# Learning to Improve Recommender Systems

**LING, Guang**

A Thesis Submitted in Partial Fulfilment
of the Requirements for the Degree of
Doctor of Philosophy
in
Computer Science and Engineering

The Chinese University of Hong Kong
March 2015

## Thesis Assessment Committee

Professor YIP Yuk Lap Kevin (Chair)

Professor LYU Rung Tsong Michael (Thesis Supervisor)

Professor KING Kuo Chin Irwin (Thesis Co-supervisor)

Professor LEUNG Kwong Sak (Committee Member)

Professor ZHANG Lianwen Nevin (External Examiner)

Abstract of thesis entitled:
Learning to Improve Recommender Systems
Submitted by LING, Guang
for the degree of Doctor of Philosophy
at The Chinese University of Hong Kong in March 2015

With the rapid development of e-commerce websites, music and video streaming websites and social sharing websites, users are facing an explosion of choices nowadays. The presence of unprecedentedly large amount of choices leads to the information overload problem, which refers to the difficulty a user faces in understanding an issue and making decisions that are caused by the presence of too much information. Recommender systems learn users' preferences based on past behaviors and make suggestions for them. These systems are the key component to alleviate and solve the information overload problem. Encouraging progress has been achieved in the research of recommender systems from neighborhood-based methods to model-based methods. However, recommender systems employed today are far from perfect. In this thesis, we propose to improve the recommender systems from four perspectives motivated by real life problems.

First and foremost, we develop online algorithms for collaborative filtering methods, which are widely applicable to recommender systems. Traditionally batch-training algorithms are developed for collaborative filtering methods. They enjoy the advantage of easy to understand and simple to implement. However, the batch-training algorithms fail to consider the dynamic scenario where new users and new items join the system constantly. In order to make recommendations for these new users and on these new items, batch-

training algorithms need to re-train the model from scratch. During the training process of batch-training algorithms, all the data have to be processed in each iteration. This is prohibitively slow given the sheer size of users and items faced by a real recommender system. Online learning algorithms can solve both of the problems by updating the model incrementally based on a rating point.

Secondly, we question an assumption made implicitly by most recommender systems. Most existing recommender systems assume that the rating distribution of collected ratings and that of the unobserved ratings are the same. Using data collected from a real life recommender system, we show that this assumption is unlikely to be true. By employing the powerful missing data theory, we develop a model that drops this unrealistic assumption and makes unbiased predictions.

Thirdly we examine the spam problem confronted by recommender systems. The ratings assigned by spam users contaminate the data of a recommender system and lead to deteriorated experience for normal users. We propose to use a reputation estimation system to keep track of users' reputations and identify spam users based on their reputations. We develop a unified framework for reputation estimation that subsumes a number of existing reputation estimation methods. Based on the framework, we also develop a matrix factorization based method that demonstrates outstanding discrimination ability.

Lastly, we integrate content-based filtering with collaborative filtering to alleviate the cold-start problem. The cold-start problem refers to the situation where a system has too little information concerning a user or an item to make accurate recommendations. With the readily available rich information embedded in review comments, which are generally discarded, we can alleviate the cold-start problem. Additionally, we can tag the black box collaborative filtering algorithm with interpretable tags that help a recommender system to provide reasons on why items are being recommended.

In summary, we solve some of the major problems faced by recommender systems and improve them from various perspectives in this thesis. Extensive experiments on real life large-scale datasets confirm the effectiveness and efficiency of proposed models.

論文題目：改善推荐系統的學習

作者　　：凌光

學校　　：香港中文大學

學習　　：計算機科學與工程學習

修讀學位：哲學博士

摘要　　：

隨着在線電子商務網站，音樂視頻網站和社會性共享推薦網站的迅速發展，網站用戶面臨爆炸性增長的選擇。前所未見的大量選擇導致信息過載問題。信息過載問題是指由於存在數量巨大的信息，用戶不能有效的理解並做出選擇的問題。推薦系統是解決信息過載問題的一個關鍵組成部分。過去數十年，推薦系統技術有了長足的進步。研究重點又基於臨近用戶的方法向基於模型的方法過度。然而，推薦系統仍然不夠成熟完善。在本論文中，我們基於真實生活中遇到的問題提出改善推薦系統的方法。

首先，我們提出推薦系統的在線學習算法。傳統推薦系統使用批量式學習算法進行訓練。這些方法容易理解並且容易實現。然而批量式學習算法不能有效應對當今推薦系統所面臨的動態情況。新的用戶和新的物品不斷加入推薦系統。在批量式學習算法框架下，要將這些新用戶新物品納入系統，需要對所有數據進行重新學習。另外，在批量式學習算法的每一個步驟中，我們需要處理所有的數據。在現今推薦系統規模下，這通常是非常耗時的。在線學習算法可以通過對每一個數據點調整模型而解決上述兩個問題。

其次，我們深入調查大量推薦系統所作的一個假設。該假設默認推薦系統蒐集的打分數據的分佈和未蒐集到的打分數據的分

佈是完全一致的。我們使用在真實推薦系統中蒐集的數據證明這個假設極不可能爲真。使用丟失數據理論的方法，我們提出一個不基於改假設的模型。我們的模型放棄了這個假設並且能夠得到公正的推薦。

再次，我們詳細調研推薦系統中的垃圾用戶問題。垃圾用戶的打分會污染正常用戶的數據並導致正常用戶的體驗受到影響。我們提出使用用戶聲譽系統去記錄用戶的聲譽並利用用戶的聲譽去區分垃圾用戶和正常用戶。我們提出一個聲譽生成系統的框架。許多聲譽生成系統是我們聲譽生成系統框架的一個實做。基於該框架，我們還提出一個基於矩陣分解的用戶聲譽生成系統。該系統擁有出衆的分辨垃圾用戶的能力。

最後，我們將基於內容的推薦和協同過濾推薦有機結合以便減輕乃至解決冷啓動問題。冷啓動問題是指推薦系統中關於某個用戶或物品的信息是如此之少以至於系統不能對該用戶或改物品做出有效的推薦。用戶的文字性評價中通常包含大量用戶喜好和物品屬性信息。但用戶的文字性評價通常都被直接丟棄。我們提出一個同時使用基於內容的方法去處理用戶文字性評價信息，使用協同過濾方法處理用戶打分的整合式推薦模型。我們的模型能有效減輕冷啓動問題的影響並且對黑盒協同過濾算法提供可理解的詞彙標籤。這些標籤有助於幫助推薦系統提供推薦的原因。

綜上所述，在本論文中我們解決了推薦系統面臨的實際問題並從各個方面對傳統推薦系統進行改進。大量真實數據上的實驗驗證我們提出方法的有效性和高效性。

# Acknowledgement

I would like to express my sincere gratitude and appreciation to the people who assisted me in the research presented in this thesis. First and foremost, I would like to thank my supervisors, Prof. Irwin King and Prof. Michael R. Lyu, for their guidance during my study at CUHK. Their support and encouragement always guide me as a beacon when I was in need of help. I learn from them not only on knowledge, but also on attitude in doing research. I am very grateful for their high standard requirement on writing and presentation, which helped me enormously.

I am grateful to my thesis committee members, Prof. Yuk Lap Yip and Prof. Kwong Sak Leung for their helpful comments and suggestions about this thesis. I appreciated the invaluable guidance Prof. Kwong Sak Leung provided in choosing my research topic when he was my final year project supervisor during my undergraduate study. My special thanks to Prof. Lianwen Zhang who kindly served as the external committee for this thesis.

I would like to thank my mentors, Dr. Wei Xiang and Dr. Qian Xu during my internship at Baidu. Their help and advice are invaluable to me.

I thank Haiqin Yang, Chao Zhou, Hao Ma, Hongbo Deng, Xin Xin and Zibin Zheng for their effort and constructive discussions in conducting the research work in this thesis. Over the years, I was very happy to share office with my talented colleagues, Yu Kang, Shouyuan Chen, Qirun Zhang, Chen Cheng, Tong Zhao, Hongyi Zhang, Shenglin Zhao, Jieming Zhu, Pinjia He, Cuiyun Gao, Yuanyuan

Man, Yuxin Su, Xixian Chen, Priyanka Garg and Negin Rahimi. Life would not be as nearly as enjoyable without them.

Last but not least, I would like to thank my parents, and my girlfriend. Without their deep love and constant support, this thesis would never have been completed.

To my family.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

With the rapid development of e-commerce websites, online music and video streaming services and online social network, we are facing an explosion of choices nowadays. We often have trouble deciding which movie to watch, which book to read, which course to take and where to go for a travel. We are drowning in the overwhelming choices, i.e., we are beset with the information overload problem [83]. The information overload problem refers to the difficulty a person face in understanding a topic and making decisions caused by the presence of too much information. Recommender systems are the key component to alleviate and ultimately solve the information overload problem [2].

In the past decades, recommender systems have improved noticeably to produce more accurate recommendations, accommodate more users and items, and help users locate interesting products [80, 114, 115, 121, 116, 56, 58]. However, recommender systems are far from perfect. They are unable to handle new users and new items joining the system gracefully without re-training the whole system, struggle to offer recommendations for *cold* users and are ignorant of rich information embedded in the response patterns and review comments and easily manipulated by spam users. In this thesis, we present four studies on improving recommender systems from different perspectives.

We provide a brief introduction to recommender system in Sec-

tion 1.1, motivating the problems that are faced by recommender systems in Section 1.2. We introduce our contribution in Section 1.3 and present an overview of the thesis in Section 1.4.

## 1.1 Recommender System

A recommender system is an information filtering system that collects users' preferences over time and try to make predictions on which items that a user might like in the future. They are especially useful for users facing too many choices and help users make better decisions. Recommender systems can vary considerably according to the form of data collected, the underlying principals employed and the potential applications. We are going to discuss each of these factors and how they impact the recommender systems.

### 1.1.1 Data Collection

Various forms of data can be collected and used for recommendation purpose. Often recommender systems are tailored for one specific type of data.

#### Click Through Data

The most basic type of data that could be useful for a recommender system is the click through data. Given many hyperlinks on a webpage, a user is most likely to click the links according to his/her preferences. Shown in Figure 1.1 is the resulting page of googling the word "apple". There are many hyperlinks in this result page and whether a user clicks a link or not and the order of the user's click matters. This kind of data can be collected to better predict what the user wants. A user searching for "Apple" might want to buy a new Apple product such as iPhone or iPad. He could also simply want to know more about the fruit apple, although this is much less likely. Such click through data collected can be used to build a user model

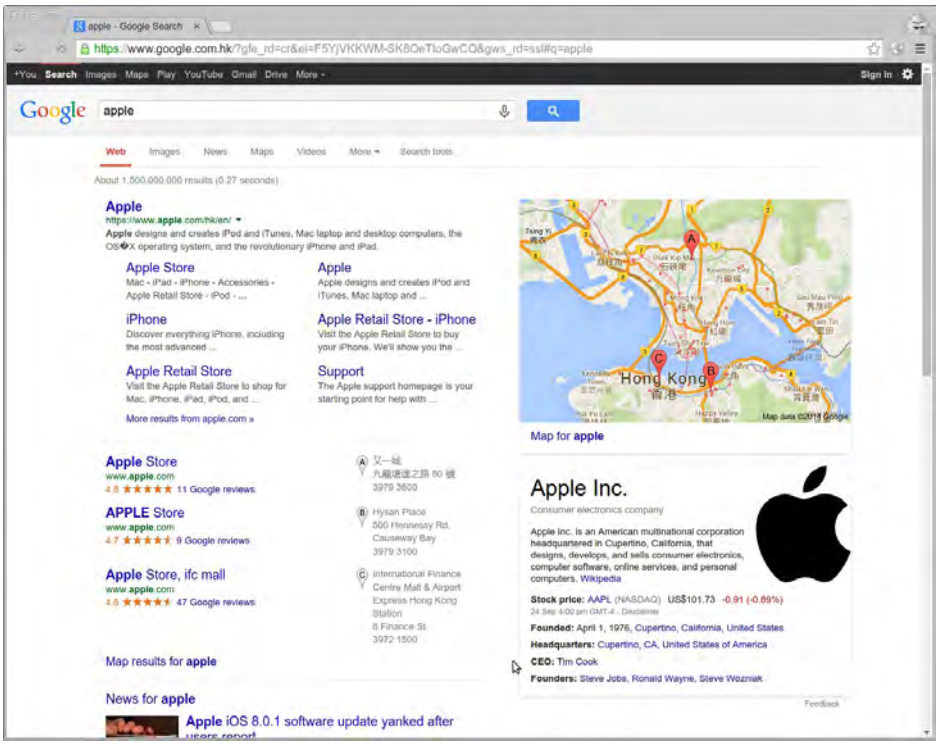



Figure 1.1: Result of Googling “Apple”

to provide suitable advertisement for this user, which in essence is a kind of recommendation. It can also be used to make the search result more suitable for the user. For example, if we know that a user is the owner of a large apple farm, then the result when the user searches for the word “apple” should be more on the fruit apple. On the contrary, for an ordinary user, the result should be the result as shown in Figure 1.1.

The click through data might have other very similar forms that are essentially the same. For example, for TV programs, a user watching a program can be seen as having “clicked” the program. In an online book sharing and recommendation website, a user might declare that he/she has read a set of books, which can be seen as having the user “clicked” the books.

Click through data has two distinct features. The first one is that click through data are binary that a link can either be clicked or not

clicked, a TV program can either be watched or not watched, a book can either be read or not read. The binary feature makes the data to have no indication of the level of fondness. The second feature is that the links not clicked do not necessarily mean the user does not like the corresponding item. It is possible that the user might have overlooked this item.

#### Rating Data

The second type of data that are suitable for use in recommender systems is rating data. This type of data is the prevailing form that is used for today's recommender system. In online e-commerce websites such as Amazon[1], eBay[2] or dedicated recommendation websites such as IMDb[3] or Douban[4], users can assign an integer score for items listed in the website to indicate their feedback on these items. The range of integer rating is mostly pre-defined and different from site to site. Shown in Figure 1.2 are the users' ratings on an earphone in the rating format on Amazon. As we can see, the rating range in Amazon is 5 and the average rating for this particular earphone is 4.3 out of 5. The rating indicates how much a user likes the item. Take 5-scale rating for example, a rating of 1 or 2 might mean that the user is not satisfied with the item and considers it below the average. A rating of 3 might indicate that the user thinks the item is mediocre. A rating of 4 could mean that the user is quite satisfied with the item, though there is room for improvement. A rating of 5 could mean that the user finds the item perfect and is very satisfied with it.

As we can see, the data collected in rating format, different from the click through data, are on a predefined *ordered* range. Rating data also have two distinct features. The first one is that rating data

[1] http://www.amazon.com
[2] http://www.ebay.com
[3] http://www.imdb.com
[4] http://www.douban.com

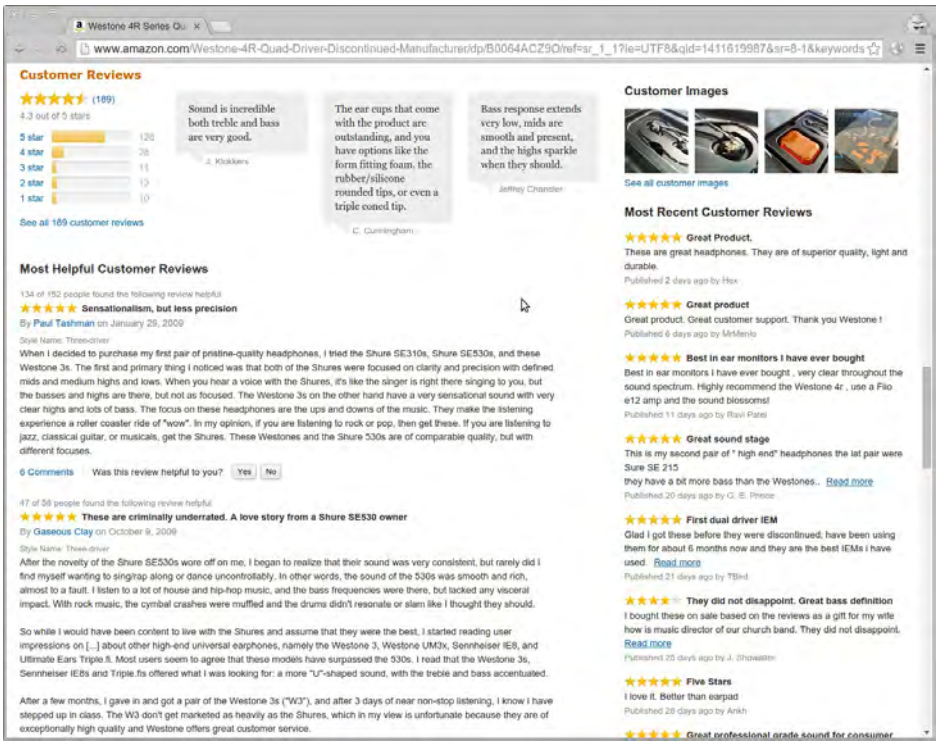



Figure 1.2: Rating feedback

are *ordered*. At least for the same user, he/she most likely prefers the item that was assigned a high rating. However, this rule does not generalize to multiple users since different users have different interpretations of the ratings. For example, a stern user might be very conservative that he/she seldom assigns the rating 5 to any item. For this user, any item has room for improvement. On the contrary, another optimistic user might never use the ratings below 3 and he/she would assign 4 or 5 for most items. The second feature is that unlike click through data, in which we do not know the user's preference on a not clicked item, we know exactly how a user likes an item once the item is rated. We are not only getting the preferred items (i.e., those items with high ratings) but also the disliked items (i.e., those items with low ratings).

Due to the innate richer features of rating data, this form of data is preferred when performing recommendation task. However, there

are also drawbacks of rating data. The first one is the choice of the range faced by the developer of the website. If the range is too small, we may not know the preference of a user to the granularity we desire. If the range is too large, users may exhibit inconsistent rating behavior. For example, given a range of 100, it's very hard for a user to tell the difference between say, 90 and 91, which might cause the user to rate inconsistently throughout time and thus violate the first feature of rating data. The second problem is the difficulty faced by the end user. To assign a rating in a pre-defined range is harder than giving a binary response, especially when the range is large. A user might have trouble maintaining the consistency over time. The particular rating might be affected by the mood, the weather and the time of the decision. Due to the above problems, rating range is generally quite small, usually 3, 5, 6, or 10.

#### Thumb Up/Down Data

Due to the difficulty of rating data discussed in the previous section, a new format of data was made popular by the online social network website Facebook[5], which is the thumb up/ thumb down data, or simply the like/dislike data. Shown in Figure 1.3 are the icons used by Facebook to indicate like and dislike. This form of data is similar to the click through data except one major difference. In click through data, we do not know the preference of the not clicked items while the dislike button is a clear indication of preferences. The like/dislike data is also binary with no granularity of fondness available. The like/dislike data is not unique to Facebook. Other websites have similar mechanism. For example, shown in Figure 1.4 is the Thumb Up/ Thumb Down feedback available in the popular video streaming website YouTube[6].

The thumb up/down data can be seen as either binary or ternary, depending on whether no response is seen as a response or not. If

[5] www.facebook.com
[6] www.youtube.com

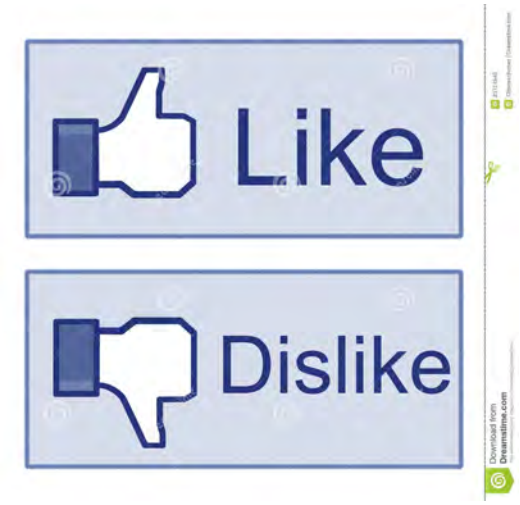



Figure 1.3: Like/Dislike

we are sure that a user has inspected an item yet not responded, then this behavior can be seen as an indication of neutrality toward the item. This might be the case of YouTube, where the site knows whether the streaming of the video has begun or not and how long the user stayed watching the video. On the other hand, in the case of Facebook, a no response can be interpreted the same as the click through data. The user might be neutral toward the item or never paid attention to it.

However, Facebook tends to hide the thumb down button on their website recently and this reduces the thumb down data to the click through data. Although in click through data we might have invalid clicks and the thumb up is much less likely to be invalid. The decision to hide the thumb down button might be due to the concern of hurting the feelings of the users who shared the item. Google provided a similar "plus one" button in Google Plus[7], another social network website.

[7]http://plus.google.com


Figure 1.4: Like/Dislike in YouTube

**Description/Review Data**

The data format that we discussed so far can be collectively called "numerical response data", in which a user indicates the fondness toward an item directly through action that can be represented as an integer (be it a 0/1 or on a ordered range). There are other formats of data that are also suitable for recommendation tasks. These formats of data, unlike the numerical response data, can be classified as description data. Meta data of a movie such as director, actors, actresses, genre, text content of a book, raw wave of a song, text of a news report, etc. are all in broad sense, description of items. In addition, text reviews that are usually collected alongside the rating data, also describe the items and users' interests. Even the video content of a movie or TV series can also be taken as description data.

One property of such description data is that usually the data is hard to use directly in a recommender system. There are so many different kinds of such data. The music information embedded in a wave file, the video contents of a movie, the text of a book are all very different data formats and each of them requires carefully

devised methods to handle. Even if we can parse the content data of items, how to match it with the taste of a user is another challenge. As far as we know, there is no single method that can take all formats of the above-mentioned description data and conduct meaningful recommendation.

Although the information embedded in video or image is hard to harness, the well-developed natural language processing techniques enable us to utilize the information in text format. The review data is an ideal text format that is suitable for recommender systems. Like the rating data, the review data contains information on both the properties of an item as well as the preferences of a user. These review text are readily available in a lot of recommender systems. Refer to Figure 1.2, we see that alongside the rating, we are also equipped with users' description of their experience on the purchased item. These reviews are usually based on personal feelings such that it not only depicts the item from the user's perspective but also gives us a hint on the preferences of the user who wrote the review. This could be a powerful source of information for a recommender system. Recently, there have been several works attempting to combine the collective power of ratings and reviews to improve the recommender system. In this thesis, we refer to review data as textual comments or textual-based content of items, that is, we consider only text as review data.

### 1.1.2 Assumptions

For a recommender system to work, we need to make a few assumptions based on which we can predict the preferences of a user. We avoid using the work "principles" here because these assumptions are often subjective and not everyone agrees with them. However, without making assumptions, it is impossible to make any recommendation. We describe two assumptions that are made by most works in recommender systems and justify them.

#### Preference Consistency Through Time

The first assumption we made in recommender systems is that *most* users' preferences are more or less consistent throughout a *short* period of time. In other words, if a user enjoys a certain type of music today, we assume that she will enjoy the same genre in the near future. This assumption is needed so that the collected past information can be utilized to produce recommendations for the future. If all users change their preferences, there is no way for a recommender system to make any meaningful recommendations. Please note that the assumption is made on the whole user base instead of a particular user. Another point is that the consistency is assumed for at least a *short* period of time. Only for those users who do not change their preferences dramatically over a period of time can we utilize their collected preferences in that period to produce recommendations.

In real life scenarios, users do change preferences throughout time. The change might due to the awareness of new things, change of attitude etc. In a recommender system, we usually decay the weight of old data and place more weight on the recent data. In this way, we can keep the recommender system up to date.

#### Preference Consistency Across Users

Another assumption made in most recommender systems is that users who shared preferences in the past tend to share similar preferences in the future. This assumption may have different expressions. For example, the following two assumptions are in essence the same.

- Items liked by a user tend to be enjoyed by the users similar to the user.
- The user-item-rating matrix is not a full rank matrix.

The user-item-rating matrix is a way of organizing the collected rating data in the form of a sparsely filled matrix. The $i$th row and $j$th

column of the matrix contains the rating assigned by user $i$ to item $j$.

This assumption is the key to power many recommender system methods. Without this assumption, the modeling of a user's preference would be independent of data collected from other users. In such situation, since every item is independent of one another, we can only leverage past behavior of *the* user to predict his/her preferences, which is hard due to the limited data. With this assumption, we assume that there is a low-rank structure of the user item rating matrix and mathematical techniques can be used to fill this matrix and thus make recommendations.

### 1.1.3 Applications

Recommender systems have wide applications in many online services. Depending on the scenario, a recommender system can be installed in movie or music streaming or recommendation websites, E-commerce websites, social sharing websites and content subscription services.

Movie and music recommendation are one of the most studied areas of recommender systems. There are dedicated recommendation websites such as Internet Movie Database (IMDb) and MovieLens [8], where the main purpose is to share knowledge about watched movies and discover new interesting movies. Users pay a subscription fee or watch ads in video streaming services such as Netflix[9] and YouTube[10]. For these websites, recommender systems, whose performance has huge impact on the potential income, play a vital role. Music recommendation is another major testbed for recommender systems. Online music streaming services, such as Spo-

[8] http://movielens.org
[9] http://www.netflix.com
[10] http://www.youtube.com

tify[11], LastFM[12], Pandora[13] and iTunes, usually provide services such as instance mix, which select songs that are similar to a given song and make a playlist and artist recommendations. These websites usually adopt a subscription business model and a good user experience is key for the business to survive, which can be greatly affected by the quality of recommendations. Recommender systems in movie and music recommendation can collect data in the rating, implicit and sometimes review format. The rating format is usually a 5-scale star rating provided by user. Implicit feedbacks are more common in music streaming websites, where a user might listen to a song, an album or an artist many times without providing any rating feedback. These feedbacks can be categorized into the click through data. Review data are more universal in movie recommendation sites, where the user would describe the feeling of watching a movie and sharing knowledge on the movie.

E-commerce websites are another major user of recommender systems. E-commerce websites such as Amazon[14] and Taobao[15] have personalized recommendations based on a user's past purchase history, rating data and review data. Shown in Figure 1.5 is a screen shot of Amazon's recommendations. Compared with movie and music recommendations, recommendations in E-commerce are different in the following ways. First, the items in movie and music recommendations are homogeneous while the items in an E-commerce website might fit in a complex taxonomy. Secondly, the number of items in movie and music recommendations is on a different (much smaller) scale compared with E-commerce websites. Take music recommendation for example, if we only consider artists, then there are only about thousands or tens of thousands of artists to consider while on a typical large online E-commerce website, the number of

[11] http://spotify.com
[12] http://lastfm.com
[13] http://pandora.com
[14] http://www.amazon.com
[15] http://www.taobao.com


Figure 1.5: Recommendations of Amazon

items could be as large as millions. Thirdly, unlike movie and music preferences, where a user's preferences are unlikely to change dramatically in a short period of time, purchase behaviors on an E-commerce website are dependent on the nature of the item itself. For example, some consumer products such as food, drinks and other consumables might have a repeated purchase pattern while the purchase of other items such as TV, piano, are one-time events.

Social sharing websites are a blend of social network and item recommendations. Famous social sharing websites include Douban[16] and Epinions[17]. Different from traditional recommender system websites where users are all independent, in social sharing websites, users can have strong (friend) or weak (follow) relationships and they share views and feelings on items. In Douban, the items are often movies, music and books while on Epinions the items are often consumer products and electronic devices. The social information presented in such websites may help them to recommend more accurately potentially interesting items to relevant users.

Recently we have seen a rise in online content subscription services. Yahoo[18] provides personalized news services for registered

[16]http://www.douban.com
[17]http://www.epinions.com
[18]http://www.yahoo.com

users. Feedly[19] is an RSS content subscriber with personalized recommendations. Toutiao[20] is a personalized news website in China. All these websites provide different content for users based on their past reading history and habits and are attracting more and more users compared with traditional non-personalized news websites. Toutiao claims it has over 50 million active users. The distinct feature of the recommender systems used in such websites is their ability to handle cold-start items. Unlike movie recommendation, where a movie can receive a lot of ratings before it was recommended to a potential user, in content subscription services, the items to recommend are breaking news that have no or little previous data and thus the cold-start is a severe problem.

We provided a few examples where recommender systems have been successfully applied and played a pivot role, but the above list is by no means a complete list of all applications of recommender systems.

## 1.2 Improving Recommender Systems

Although recommender systems have evolved considerably in the past decades, there are still several problems that are confronted by today's recommender systems. These problems affect the performance and effectiveness of recommender systems. The problems we consider in this thesis include the following:

1. Most existing model-based recommender systems are trained using batch-training algorithms. Batch-training algorithms are not suitable for the dynamic scenario faced by recommender systems today. New users and new items join the system constantly. To incorporate these new users and new items, batch-training algorithms have to re-train the model. Besides, batch-

[19] http://www.feedly.com
[20] http://www.toutiao.com

training algorithms are relatively slow and the training in large real life systems can be prohibitive.

2. Most model-based recommender systems are trained by feeding the ratings assigned by users. The trained model is then used to predict the score that users would assign to the uninspected items to make recommendations. However, there is an implicit assumption made by this paradigm of recommendation, which is that the rating distribution of the collected ratings and the unknown ratings are the same. Without this assumption, the predictions produced by a recommender system would be systematically biased. This assumption may not hold in real systems because users do filter what items to rate and what items not to rate. For example, users might rate items they like or hate with very high probability and only rate items they find mediocre occasionally. In other words, the fact that an item is rated by a user alone, irrelevant of the rating assigned by the user, contains information on how the user prefers this item. Failing to consider such response patterns can lead to biased and incorrect prediction and recommendation.

3. As is in any real life system, the spam user problem is faced by recommender systems. There are evidences showing the existence of spam users [41]. The ratings assigned by spam users have malicious purpose and they contaminate the data collected by the recommender system. The malicious ratings assigned by spam users affect the normal functioning of recommender system and lead to deteriorated experience of normal users. Identifying spam users and eliminating their impact on the system is a serious problem faced by most commercial recommender systems.

4. *Cold-start* [118] problem refers to the situation where the recommender system has too little information concerning a user or an item that it cannot make recommendations accurately.

This problem can lead to deteriorated experience for new users. Also it makes the recommendation of new items very hard. In applications such as news recommendation, where all items are "cold", the problem is fatal. Also, recommender systems with collaborative filtering make recommendation using a black-box algorithm and the system does not provide reasons as to why the items are recommended. The reason for recommendation can help users decide whether to take further actions and improve user experience in using the system.

In this thesis, we present studies that improve recommender systems by solving the above mentioned problems. To solve the first problem, we develop online learning algorithms that can accommodate new items and new users easily. To solve the second problem, we employ the powerful missing theory and incorporate it with collaborative filtering to make predictions without bias. To solve the third problem, we propose a unified framework for user reputation estimation that subsumes many existing reputation estimation methods, and propose matrix factorization based methods that show outstanding discrimination ability. To solve the fourth problem, we propose to combine content-based filtering with collaborative filtering to exploit the combined power of rating data and review data.

## 1.3 Thesis Contribution

In this thesis, we make contributions to improve recommender systems in the following ways:

1. **Online collaborative filtering**

   Collaborative filtering (CF), aiming at predicting users' unknown preferences based on observational preferences from some users, has become one of the most successful methods for building recommender systems. Various approaches to CF have been proposed in this area, but seldom do they consider

the dynamic scenarios: 1) new items arriving in the system, 2) new users joining the system; or 3) new rating updating the system are all dynamically obtained with respect to time. To capture these changes, we develop an online learning framework for collaborative filtering. Specifically, we construct this framework consisting two state-of-the-art matrix factorization based CF methods: the probabilistic matrix factorization and the top-one probability based ranking matrix factorization. Moreover, we demonstrate that the proposed online algorithms bring several attractive advantages: 1) they scale linearly with the number of observed ratings and the size of latent features; 2) they obviate the need to load all ratings in memory; 3) they can adapt to new ratings easily; and 4) they present sparse solutions by explicitly imposing specific regularizations. Finally, we conduct a series of detailed experiments on real-world datasets to demonstrate the merits of the proposed online learning algorithms under various settings.

Practically, the online learning algorithms attain impressive performance to their batch ones while achieving some properties, e.g., sparse solutions. In practice, our online algorithms achieve comparable results as batch-training algorithm very quickly while attaining some additional features, i.e. a sparse solution.

2. **Response aware collaborative filtering**

   Previous work on recommender systems mainly focus on fitting the ratings provided by users. However, the response patterns, i.e., some items are rated while others not, are generally ignored. We argue that failing to observe such response patterns can lead to *biased* parameter estimation and sub-optimal model performance. Although several works have tried to model users' response patterns, they miss the effectiveness and interpretability of the successful matrix factorization collaborative filtering approaches. To bridge the gap, we unify explicit response mod-

els and PMF to establish the Response Aware Probabilistic Matrix Factorization (RAPMF) framework. We show that RAPMF subsumes PMF as a special case. Empirically we demonstrate the merits of RAPMF from various aspects.

3. **User reputation estimation**

   Online rating systems are now ubiquitous due to the success of recommender systems. In such systems, users are allowed to rate the items (movies, songs, commodities) in a predefined range of values. The ratings collected can be used to infer users' preferences as well as items' intrinsic features, which are then matched to perform personalized recommendation. Most previous work focuses on improving the prediction accuracy or ranking capability. Little attention has been paid to the problem of spammers or low-reputed users in such systems. Spammers contaminate the rating system by assigning unreasonable scores to items, which may affect the accuracy of a recommender system. There are evidences supporting the existence of spammers in online rating systems. Reputation estimation methods can be employed to keep track of users' reputation and detect spammers in such systems. We propose a unified framework for computing the reputation score of a user, given only users' ratings on items. We show that previously proposed reputation estimation methods can be captured as special cases of our framework. We propose a new low-rank matrix factorization based reputation estimation method and demonstrate its superior discrimination ability.

4. **Combine ratings with reviews**

   Most existing recommender systems focus on modeling the ratings while ignoring the abundant information embedded in the review text. We propose a unified model that combines content-based filtering with collaborative filtering, harnessing the information of both ratings and reviews. We apply topic modeling

techniques on the review text and align the topics with rating dimensions to improve prediction accuracy. With the information embedded in the review text, we can alleviate the cold-start problem. Furthermore, our model is able to learn latent topics that are interpretable. With these interpretable topics, we can explore the prior knowledge on items or users and recommend completely "cold" items. Empirical studies on 27 classes of real-life datasets show that our proposed model lead to significant improvement compared with strong baseline methods, especially for datasets which are extremely sparse where rating-only methods cannot make accurate predictions.

## 1.4 Thesis Organization

The rest of the thesis is organized as follows:

- **Chapter 2**

  In this chapter, we review the background knowledge of recommender systems, reputation estimation in online rating systems and topic modeling. More specifically, we introduce *content-based filtering* and *collaborative filtering*. We focus more on collaborative filtering, which is the dominating method in recommender systems nowadays. We briefly survey rating oriented methods and ranking oriented methods, neighborhood-based methods and model-based methods.

- **Chapter 3**

  In this chapter, we examine the batch-training method for collaborative filtering, which requires all the data presented before the training. Incremental changes to the training data require re-training from scratch. This batch-training paradigm does not fit in today's dynamic scenarios where new users and new items keep joining the system. To solve the problem, we

develop online learning methods for Probabilistic Matrix Factorization and Ranking Matrix Factorization, which take a single data point at a time and update model parameters according to that data point. Our online learning methods are much faster at training and accommodate the dynamic case. We show the effectiveness of our online learning algorithm on large real life datasets.

- **Chapter 4**

  In this chapter, we take a close look at an assumption made by most work in collaborative filtering, which is that the rating distribution of the observed ratings are the same as the underlying rating distribution. Using real data collected from Yahoo Music, we show that it is highly possible that the assumption does not hold in real recommender systems and show mathematically that failing to consider such discrepancy between the two distributions could lead to biased and incorrect predictions in recommendation. Equipped with missing data theory, we develop a matrix factorization method that drops the incorrect assumption. Empirical studies on the Yahoo Music dataset show that our method performs much better under a more realistic setup.

- **Chapter 5**

  Spam users are becoming a more and more serious problem in online rating systems. Spam users' motivation includes promoting certain items while suppressing other items, which could be financially rewarding in markets such as AppStore. A user reputation estimation system can be employed to track the reputation of users and identify spam users. Excluding the contaminated ratings assigned by spam users from the system allow recommender systems to make more accurate recommendations. In this chapter, we propose a user reputation estimation framework in online rating systems, where a lot of pre-

vious methods can be subsumed as instances. Based on the framework, we also propose a matrix factorization based user reputation estimation method, which shows improved discrimination power under various spamming strategies.

- **Chapter 6**

  Review comments complement rating data in a lot of online rating systems, but the review data are generally discarded despite the rich information they contain. In this chapter, we propose a model that exploits the information in both the rating data and review data. When the data are extremely sparse, the review data contains information that can help alleviating the cold-start problem and allow us to make better recommendations for new users and new items. In addition, with combined power of ratings and reviews, we can learn the physical meaning of the latent topics. This allows us to recommend items with reasons, which helps users understand why these items are recommended.

- **Chapter 7**

  The last chapter summarizes this thesis and addresses some research directions that can be explored in the future.

To make the chapters self-contained, we briefly reiterate the critical definitions and models that are related in the following chapters.

---

□ **End of chapter.**

# Chapter 2

# Background Study

In this chapter, we introduce the background knowledge on recommender systems, reputation estimation and topic modeling. We review both content-based filtering, which makes recommendation by analyzing the content of the item and match it with the interests of users, and collaborative filtering, in which we analyze the past rating behavior of users on items to discover users' interest, items' features and match them to make recommendations. These two topics together describe the recommender systems. We review traditional methods of reputation estimation in online rating systems and its close relationship with recommender systems. We explore more on reputation estimation in Chapter 5. Topic modeling refers to the area of study that tries to discover topics or themes from text. Topic modeling has wide application in automatic text clustering, summarization.We will apply topic modeling techniques to review text in Chapter 6 and combine it with collaborative filtering to improve recommender systems.

This chapter is organized as follows. In section 2.1, we briefly talk about the classification of recommender systems and the basic working principles of each type. In section 2.2, we briefly introduce content-based filtering methods and their applications in news and music recommendation. In section 2.3, we present a detailed survey on collaborative filtering techniques, covering neighborhood-based methods as well as model-based methods. In section 2.4, we briefly

talk about existing work on reputation estimation in online rating systems and Lastly in section 2.5, we review several topic modeling methods that have wide applications in text classification and content-based filtering.

## 2.1 Recommender systems

Recommender systems are a subclass of information filtering system that seek to predict the rating or preference that a user would assign to an item so that given a user, the system can make recommendations on which items to inspect.

According to the way recommender systems work, they can broadly be classified into two types: *content-based filtering* [103] and *collaborative filtering*.

Content-based filtering recommends an item to a user by matching up the features of the item with the preferences of the user, both of which are learnt by analyzing the contents and profiles. The features of the content as well as the preferences of the user have to be learnt. This is usually very hard because the inherent diversity of contents of items. Content-based filtering has been successfully applied in areas such as news recommendation [72, 55, 105] and music recommendation [20, 123].

Collaborative filtering (CF), on the other hand, approaches the recommendation problem by analyzing the co-occurrence patterns of user-item pair, which is often attached with an integer rating. There are extensive investigations on collaborative filtering, from neighborhood-based methods [117, 65] to model-based methods [57, 115]. Recently, some methods concentrated on ranking [59, 110] the items better. Other approaches leveraged social [81, 82] and side information [136] to improve the performance.

## 2.2 Content-based filtering

Content-based filtering techniques approach the recommendation problem by analyzing the contents of items and match the features of the item with the preferences of a user. There are three major components in a content-based filtering system [78].

- Content Analyzer
  This is the module that perform pre-processing step that transforms the raw content of an item (e.g. text, wave, video content) into structured relevant information, usually in the form of feature representation. For example, text can be represented as keyword vectors, topic distributions or TFIDF[1].

- Profile Learner
  This component collects data concerning the past preference history of a user and construct a user profile that is suitable for recommending items of the format produced by the content analyzer. The user profile could be a prototypical item feature vector that is most representative of the user's preferences. For example, a naive strategy to generate a user profile could be the average features of the items liked by the user in the past. In real systems, the items liked and disliked by a user are considered to produce a prototypical item feature that is most representative of the user.

- Filtering Component
  This component is in charge of producing a relevance score for items given a user, based on the items' features and user profile. The relevance score could be a binary relevance or an integer indicating the level of match. For naive user profile approach, the relevance score could be the similarity between the item feature and the prototypical item feature representing

[1] Term Frequency-Inverse Document Frequency, which is a numerical static that is intended to reflect how important a word is to a document in a corpus

the user. Similarity measure can be cosine similarity, Pearson Correlation Coefficient or other similarity measures. We defer the introduction to similarity measure methods to section 2.3, where we will investigate how to measure the similarity of two vectors in a general setting.

Compared with collaborative filtering, content-based filtering enjoys the following advantages.

- User Independence
  In content-based filtering methods, the build of a user's profile does not depend on other users' behavior. User profile is build based solely on this user's past preference histories. This separation of concern allows parallel construction of user profile in large clusters simultaneously. Whereas in collaborative filtering, every user's past preference histories has impact on the system and parallel learning algorithms is not a trivial task. Given the large user base in online service providers, ease of parallel implementation could be a huge plus.

- Explainability
  Another advantage enjoyed by content-based filtering is that usually they can provide explanations on why certain items are recommended. For example we recommended a news article on Health Care Act to a user who is an avid reader on public health policy related article. In traditional collaborative filtering, the recommendations for a user are produced by a black-box algorithm that has no explanations. Explanation on why a particular item is recommended can help user decide whether to take further actions.

- New Items
  In Chapter 1, when we introduced the applications of recommender systems, we mentioned that news recommendation has the distinct feature of extremely large set of items and breaking news has no previous ratings. Content-based filtering is

well suited for this situation. To recommend a news article, in content-based filtering, we analyze the content of the article and match it with the preferences of a user, without the need to collect users' ratings on this article. Note that collaborative filtering methods cannot recommend an item who received no ratings.

Nonetheless, content-based filtering has several downsides.

- Domain Dependent
  Content analyzer of a content-based filtering system usually involves domain dependent experts who select and devise the features suitable for the description of the items. For example, the features used for music recommendation are very different from the features used for news article recommendation. In addition, the content analyzer usually requires domain knowledge. For example, content-based music recommendation often extract features related to instrumentation, rhythm and harmony from music audio signals, which require an expert in audio wave analysis as well as in music comprehension.

- Over-Specialization
  Another drawback of content-based filtering is its inability at discovering new unexpected items. This drawback is inherent in the way the content-based filtering operates. The system is only able to find items that are highly correlated with a user's profile. This problem is known as serendipity problem, which refers to the situation where a recommender system can only produce recommendations with limited degree of novelty. Take news article recommendation for example, a content-based filtering system can only recommend politics related article to a user who has only read politics related articles. The user may as well be interested in sports but the system is never going to recommend sports related articles due to the lack of previous data. In collaborative filtering, this problem is much less sev-

erer because as long as a similar user has expressed interests in other fields, items in those fields can be potential recommendations.

- New Users
  Content-based filtering solves half of the cold-start problem because it can recommend items with no previous ratings. However, similar to collaborative filtering, content-based filtering is unable to handle the new users which has no or few preferences available. With no or few previous preferences, we cannot construct a reliable user profile, a key component in successful recommendation.

### 2.2.1 State-of-the-Art Content-based Filtering Systems

Content-based filtering, due to its unique advantages, has seen wide application in news recommendation [72, 55, 105] and music recommendation [20, 123].

For personalized news recommendations, content-based filtering has been adopted in personal news agents [9], news reader for wireless devices [23, 50] and web-based news aggregators [8]. These systems build user profiles based on information collected from past preference histories or explicit feedback during an interactive initialization step. In [8], the authors let the users select a set of keywords describing their interest when they first join the system and then refined through users click behavior during the use of the recommender system. In a recent study, the authors of [72] improved Google News recommendation by analyzing users' click behavior. The news articles are classified into categories (using topic modeling or available meta data) and user profile is built as a Bayesian model of click probability given the category of the news article. The filtering component is implemented as the probability of the user click the news article using the learnt news classification and user specific click probability.

Music recommendation is another area where content-based filtering has wide adoption. Using collaborative filtering alone in music recommendation may lead to a few problems. The similarity recommendations created using collaborative filtering do not necessarily correspond to actual musical similarity. They are subject to popularity bias [19]. Domain specific knowledge exists that allow us to analyze the music audio content of a song to discover music features such as instrumentation, rhythm and harmony. These features are more suitable at finding similar songs from musical perspectives. Most studies [6, 39, 18, 19, 20, 123] in content-based music recommendation focus on content analyzer component of the content-based filtering.

## 2.3 Collaborative filtering

Collaborative filtering, unlike content-based filtering which use content analyzer to construct items features and match them with a user's profile to make recommendations, rely solely on the past ratings assigned by users to items. In collaborative filtering, we try to extract intrinsic insights such as users who buy diaper often buy beer at the same time, be it explicit such as the ones in neighborhood-based methods or implicit ones as is in model-based methods. The working mechanism of collaborative filtering is best explained as is in user-based methods. In user-based methods, we analyze the similarities between users according to the past ratings they assign to items. If two users tend to assign similar ratings to co-rated items, then we consider the similarity between them is high. Then in order to make recommendations for one of the users, we take the item set that the other user assigned a high rating and select those that the first user has not inspected before. The assumption made by this simple method is that users who shared similar tastes in the past tend to have similar preferences in the future.

Historically, collaborative filtering starts with neighborhood-based

(also known as memory-based) methods, which directly manipulate the known ratings to predict the unknown ratings. In neighborhood-based methods, the similarity between users (or items) are calculated using the raw ratings and then a weighted average of other users' (items') ratings are calculated as the predicted rating that a user (item) would assign (receive). Although simple to interpret and easy to implement, neighborhood-based methods are often sub-optimal in prediction accuracy and time consuming to make a prediction. Collaborative filtering methods have shifted from neighborhood-based methods to model-based methods. Successful model-based methods include Probabilistic Latent Semantic Analysis, Matrix Factorization based methods, etc.

In the following of this section, we first describe the notations that will be used throughout this thesis and then define the problem of (rating oriented) collaborative filtering. We then review some of the most important work in collaborative filtering, including both neighborhood-based methods and model-based methods.

### 2.3.1 Notations and Problem definition

The notations used in this thesis are described as follows. Please note in later chapters, we may define additional notations when needed in context to better describe the model. Suppose that we are given a set of $N$ users $U = \{u_1, u_2, \cdots, u_N\}$ and a set of $M$ items $I = \{i_1, i_2, \cdots, i_M\}$. Users' rating on the items are arranged in a $N \times M$ matrix $R$ where entry $r_{u,i}$ denote the rating that user $u$ gives to item $i$ and $r_{u,i} = 0$ if $u$ have not rated $i$. The set of all items that user $u$ have rated is denoted by $I_u$ and the set of all users who have rated item $i$ is denoted by $U_i$. Alternatively, we denote the set of all observed triplet $(u, i, r) \in \mathcal{Q}$, where $u$ is the user id, $i$ is the item id and $r$ is the rating given by $u$ to $i$. The whole set is denoted by $\mathcal{Q}$. Other notations used which are model specific will be discussed in the context.

Given $M, N, R$ or $M, N, Q$, which are equivalent, rating oriented collaborative filtering tries to produce prediction for the rating that is likely to be assigned to $i$ by $u$, i.e. $r_{u,i}$. We denote the prediction by $\hat{r}_{u,i}$. Instead of giving prediction of ratings, ranking oriented collaborative filtering output a rank $\pi$ of the items in decreasing order of preference.

### 2.3.2 Neighborhood-based methods

Neighborhood-based methods manipulate the ratings assigned by users directly in making predictions. Currently there are rating-oriented methods, which try to minimize squared error between prediction and true rating, and ranking-oriented methods, which try to rank the items in correct order. For rating-oriented approaches, user-based methods [14, 43] and item-based methods [30, 65, 117] are mostly studied. They utilize similar users' and similar items' ratings to make prediction, respectively. User based methods take the weighted average of the ratings assigned by a set of neighbor users who share similar tastes in the past as the predicted rating. The weight is determined by the similarity between active user and his neighbor user. More accordant ratings that a neighbor user assigned with the active user, more weight this user's rating would carry when predicting the rating for active user. Item based methods make the observation that a user tends to assign similar ratings to similar items. Weighted average of the active user's past ratings is taken as the prediction. Popular similarity measure of users and items include Pearson Correlation Coefficient (PCC) [113], Vector Similarity (VS) [14] and Adjusted Vector Similarity (AVS) [117]. PCC similarity generally attains better accuracy for user based methods because it is in essence a normalized correlation measure, which describes the kind of similarity we seek for. Sarwar et al. reported [117] that AVS in item based methods can achieve better performance than user based methods while promoting efficiency since we usually

have far more users than items. Due to their simplicity, these methods are very popular and are applied widely in commercial websites [65, 113].

Besides rating-oriented methods, Liu et al. proposed [74] a pairwise ranking-oriented method called EigenRank, and reported it attains more credible ranking scores. It computes an active user's pairwise preferences over all items based on the neighbor users' preferences in a way that is analogous to user based methods. Similarities between users are measured using Kendall Rank Correlation Coefficient (KRCC) [85], which considers only pairwise preference. Then these pairwise preferences are fused to produce a rank of items in decreasing order of preference. Liu et al. reported [74] that EigenRank attains more credible ranking score than rating-based methods.

Although memory-based methods are easy to implement and understand, they impose several limitations. First, the accuracy of memory-based approaches is often sub-optimal [46, 76, 114]. On top of that, they are susceptible to the data sparsity problem because in order to measure the similarities, two users need to rate at least some items in common. Furthermore, as they manipulate the ratings directly, the time complexity and memory consumption can be potentially very expensive. Finally, they solve the problem heuristically without clearly maximizing or minimizing an objective [46].

We now take a deeper look at user-based methods and item-based methods.

#### User-based methods

In user based methods, the unknown ratings $\hat{r}_{u,i}$'s is predicted using a set of other users' rating for item $i$. Let $s_{u,v}$ denote the similarity measure between $u$ and $v$. We defer the discussion of various similarity measures to later part of this section. For now, let us assume that we have similarity measure that would assign larger value for a pair of users who share more interests in common. Then the

prediction can be calculated as:

$$\hat{r}_{u,i} = \frac{\sum_{v \in N_u \cap U_i} s_{u,v} \cdot r_{v,i}}{\sum_{v \in N_u \cap U_i} s_{u,v}}. \tag{2.1}$$

Note, in equation 2.1 $N_u$ is the set of all neighbor users of user $u$ and $U_i$ is all the users who have rated $i$.

Equation 2.1 gives a simple method to calculate the prediction. However, there are several issues associated with this method, which we will now explain.

The first problem is that users may have very different rating behaviors. A conservative user might never give more than 3 while an optimism user would rate the items on the scale from 3 to 5. That is, the ratings given by a user could be biased. It is easy to see that the simple user based method could give rating prediction wildly deviate from the true rating. As a simple solution for this is to adjust a user's ratings with his mean rating [14, 113]. This would give a slightly modified prediction:

$$\hat{r}_{u,i} = \bar{r}_u + \frac{\sum_{v \in N_u \cap U_i} s_{u,v} \cdot (r_{v,i} - \bar{r}_v)}{\sum_{v \in N_u \cap U_i} s_{u,v}}. \tag{2.2}$$

In equation 2.2, $\bar{r}_u$ denotes the mean ratings given by $u$. Note that this equation only considers the rating behavior problem in the prediction phase. However, to obtain a good result, the similarity measurement should also take this problem into consideration.

Instead of using the simple approach given in equation 2.2, several different approaches have been proposed to alleviate the bias problem by normalizing the rating prior the similarity calculation. Jin et al. proposed a technique to normalizing the user ratings based on the halfway accumulative distribution [74, 53].

Another problem associated with user based approach is that the user item rating matrix $R$ is usually very sparse. This could lead to

serious problems. First of all, fewer ratings can lead to poor similarity measure. Consider two users who happened to rate an item with the same score and no other co-rated items. This will lead to a large similarity measure between these two users in most of the similarity measure methods. These two users could have very different tastes about other items and they just happen to assign *this* item the same rating. So in summary, the first problem caused by sparse user item rating matrix is randomness in similarity measure. The second problem associated with data sparsity is that we may end up with no candidate rating in the known dataset to make prediction for certain entries. Consider the case when $N_u \cap U_i$ is $\emptyset$, obviously we cannot make prediction for $u$'s rating for $i$. Goldberg et al. use dimensionality reduction method to alleviate this problem [38]. Other data smoothing techniques have also been proposed targeting at this problem [130, 79].

#### Item-based methods

Another type of approach that utilizes similarity measure is item-based methods. In item-based approaches, we calculate the similarities between items and make prediction for a user based on his past rating on similar items. The intuition is that users would assign similar scores for similar items. In application scenarios such as movie or book recommendation, there are generally much less items than users, item-based methods are less prone to data sparsity problem than user-based methods. This is because similarities between items can be calculated more accurately since items are much less likely to be rated by only a few users. The chance that two items happen to share only a few similar ratings is small. Predictions are made in a similar fashion as is in user-based methods:

$$\hat{r}_{u,i} = \frac{\sum_{j \in N_i \cap I_u} s_{i,j} \cdot r_{u,j}}{\sum_{j \in N_i \cap I_u} s_{i,j}}. \tag{2.3}$$

In equation 2.3, $s_{i,j}$ denotes the similarity between $i$ and $j$. $N_i$ is all the neighbor items of $i$ and $I_u$ is all the items that $u$ have rated. Note that this item based prediction is also more robust against the bias problem. This is because only $u$'s past ratings are used to predict the score that $u$ would assign to $i$. So the prediction given by equation 2.3 is well adapted to $u$'s bias. Sarwar et al. have reported [117] that item based method could give more accurate prediction. As a side bonus, item based methods are usually more efficient than user based methods [117].

#### EigenRank

Both user-based methods and item-based methods are rating oriented methods and they output a rating for each user item pair. We now review a ranking oriented method called EigenRank, which outputs a ranking $\pi$ of the items for a given user. EigenRank [74] adopt a method that is essentially neighborhood-based, i.e. utilizing neighbor users' rating directly.

The advantages of ranking oriented approach are briefly talked about in Chapter 1. Liu [77] categorized learning to rank into point-wise, pair-wise and list-wise approaches. Point-wise approach corresponds to calculate a ranking score for each item and then rank the items using these ranking score. This is in essence rating oriented methods in collaborative filtering. As is pointed out, evaluation for traditional rating oriented methods is how well the predicted rating matches the true rating. Popular evaluation methods that serve this purpose include MAE and RMSE, which we introduce later. However, methods targeting at the rating prediction problem alone could obtain suboptimal ranking even though it might perform well in rating evaluation metric. That is, the rating metric is hard to interpret directly as the ranking quality. Rating metric and ranking metric are correlated but are not monotonically correlated. Pair-wise method usually has a preference function defined for each pair of items. EigenRank is a pair-wise learning to rank method. When

trying to produce a ranking for a specific user, a pair-wise method would have to consider all the possible pair-wise preference for the user. The computation complexity is $O(M^2)$, where $M$ is number of items in the system. This can be very slow in practice and does not scale well to large datasets. However, pair-wise methods can give better ranking result than point-wise methods and thus is worth studying. List-wise approaches usually involve a ranking model that assigns a score to a ranking of items (permutation) according to a predefined mathematical form. We consider one such model-based list-wise approach in later part of this section.

**Preference function** To model user's preference over different items, we define a preference function $\Psi : I \times I \rightarrow \mathbb{R}$. $\Psi(i, j) > 0$ means that user prefer $i$ to $j$ and the magnitude $|\Psi(i, j)|$ denotes the strength of this preference. $\Psi(i, j) = 0$ then indicates that there is no preference between $i$ and $j$. We dictate that $\Psi$ have the following additional properties for easier mathematical manipulation: $\Psi$ should be anti-symmetric, i.e. $\Psi(i, j) = -\Psi(j, i)$. $\Psi(i, i) = 0$ for all $i \in I$. However, $\Psi$ does not have to be transitive, i.e. $\Psi(i, j) > 0$ and $\Psi(j, k) > 0$ does not imply $\Psi(i, k) > 0$. Note that the last properties follow the work [34].

Having stated the desired properties of a preference function, we can now relate this definition to the collaborative filtering problem. For each user $u$, we have a list of rated items $I_u$ and their corresponding ratings $R_u$. That a higher rating for $i$ than $j$, i.e. $r_{u,i} > r_{u,j}$, should serve as an evidence that user $u$ prefer $i$ to $j$. Since we do not have all the rating information for all items, we leverage the ratings assigned to the item pairs by neighboring users. The fundamental idea here is the same as is in user-based methods. We use $u$'s neighboring users' ratings on $i$ and $j$ as an estimation of the preferences of $u$ on $i$ and $j$. This observation leads to the following definition of

$\Psi$ in collaborative filtering:

$$\Psi(i,j) = \frac{\sum_{v \in N_u^{i,j}} s_{u,v} \cdot (r_{v,i} - r_{v,j})}{\sum_{v \in N_u^{i,j}} s_{u,v}} \tag{2.4}$$

The $N_u^{i,j}$ denotes $u$'s neighbors who have rated both $i$ and $j$. So in effect, we are taking the weighted average of neighbors' preference as $u$'s preference.

**Objective function** Equipped with the definition of the preference function $\Psi$, we now define the objective function $\mathcal{O}$ for a rank $\rho$. Intuitively we want to choose a ranking $\rho$ so that it agrees with the preference function $\Psi$ as much as possible. Let $\rho(i)$ be the position of item $i$ in the rank $\rho$. So that $\rho(i) < \rho(j)$ means $i$ is ranked higher than $j$. Then the objective function $\mathcal{O}$, which measure how well the ranking $\rho$ agrees with $\Psi$ can be defined as:

$$\mathcal{O}^{\Psi}(\rho) = \sum_{i,j:\rho(i)<\rho(j)} \Psi(i,j). \tag{2.5}$$

Following this definition, we want to find $\rho^*$ that maximize $\mathcal{O}$, that is:

$$\rho^* = \text{argmax}_{\rho} \mathcal{O}^{\Psi}(\rho). \tag{2.6}$$

Solving this problem in equation 2.6 exactly has been proved to be NP-complete problem[26] based on reduction from Cyclic-Ordering problem[36]. We now show two methods proposed in [74] to solve this problem heuristically.

**Greedy algorithm** The greedy algorithm for solving 2.6 appeared first in [26]. Its complexity is linear with the number of items $M$. But the preference for each possible item pair still needs to be calculated. So the overall time complexity is still $O(M^2)$. The algorithm is given in Algorithm 1. This greedy algorithm assigns a utility value

**Algorithm 1** Greedy Order

```
for all each i ∈ I do
    π(i) = Σ_{j∈I} Ψ(i, j) − Σ_{j∈I} Ψ(j, i)
end for
while I is not empty do
    t = argmax_{i∈I} π(i)
    ρ̂(t) = |I|
    I = I − t
    for all each i ∈ I do
        π(i) = π(i) + Ψ(t, i) − Ψ(i, t)
    end for
end while
```

$\pi$ for each $i \in I$. This could be interpreted as the overall preference of $i$ compared to all other items. So a higher $\pi(i)$ would imply that user prefers $i$ to other items. The algorithm works by selecting the item $t$ with the largest $\pi(t)$ value and then eliminates the effect of $t$ from all the following $\pi$s. As this is done for all items, we would obtain a ranking of items $i \in I$. Cohen et al. analyzed the performance of this algorithm and proved that a ranking $\hat{\rho}$ produced by this algorithm is within a factor of 2 of the optimal, i.e. $\mathcal{O}^{\Psi}(\hat{\rho}) \geq \frac{1}{2}\mathcal{O}^{\Psi}(\rho^*)$ in [26].

**Random walk algorithm** In this section, we introduce another method to solve the optimization problem in Eq. 2.6 that is proposed by Liu et al. in [74]. Different from the greedy ordering approach that tries to define a utility value for each item and then rank the items accordingly, random walk algorithm tackles this problem from another perspective. Inspired by PageRank[15], the random walk algorithm takes the ranking problem as a Markov Chain where the states correspond to the items and the transition probability depends on user's preference function $\Psi$.

Markov chain model is a tool that is used a lot in machine learning and it is very effective at aggregating partial ordering information to produce an overall rank. If we define the states of the Markov

chain to be the items to be ranked, we could model the preference of the items by the stationary distribution overall the items. That is, the more probable the stationary state is at $i$, the more likely that user would prefer $i$. The transition from $i$ to $j$, $p(j|i)$ is governed by user's preference on $j$ over $i$. In other words, the more that a user prefers $j$, the more likely it is for the state to transit from $i$ to $j$ so that the stationary distribution probability for $j$ would be higher. Then the ranking procedure could be seen as a random walk on this Markov chain and the stationary distribution explains user's preference over the items. This is intuitively appealing in the sense that the most preferred an item is, the more frequent that user would visit that state.

We want to define transition probability depending on the preference function $\Psi$. Since $\Psi$ could give negative values, we should filter this preference value through a function that always gives positive values. It turns out that exponential function is a good choice. We define the transition probability from $i$ to $j$ to be

$$p(j|i) = \frac{e^{\Psi(j,i)}}{\sum_{j' \in I} e^{\Psi(j',i)}}. \tag{2.7}$$

Let $P$ be the transition probabilities denoted in matrix form, where each entry $P_{i,j}$ is the transition probability $p(j|i)$. Define $\pi_t = [p_t(1), p_t(2), \cdots, p_t(m)]$ to be the distribution at step $t$, where $p_t(i)$ denote the probability of being at item $i$ at step $t$. Then $\pi_t$'s can be calculated iteratively using the following update formula

$$\pi_{t+1} = \pi_t P. \tag{2.8}$$

The stationary distribution $\pi^*$ can be computed as $\pi^* = \lim_{t \to \infty} \pi_t$. Note that $\pi$ is guaranteed to converge to a unique stationary distribution given that $P$ is an irreducible and aperiodical. However, the mixing time, i.e. the steps needed until converge, is unknown beforehand. Alternatively we can calculate the eigenvalues and eigenvectors associated with $P$ and the principle eigenvector would be

the stationary distribution $\pi^*$. However, eigenvector calculation for a large matrix is a computation demanding operation. So in practice, we would simply compute $\pi$ using the equation 2.8 and watch for convergence.

To avoid the reducibility of $P$ and improve numerical stability, the author of EigenRank adopted a trick used in PageRank, which interpolates a "teleport" probability matrix $E$ with the transition probability $P$. $E = ev^T/n$ where $e$ is the vector with all components equal to 1 and $v$ is a personalization vector. The new transition probability is defined as:

$$\bar{P} = \epsilon \cdot E + (1 - \epsilon) \cdot P \tag{2.9}$$

where $\epsilon$ is a scalar parameter controlling the mixing proportion. The intuition of adding $E$ is that users do not have to jump to a state $i$ from current state $j$ following the probability $p(j|i)$. He could also change state to $j$ by "teleporting" to $i$, following the probability defined in $v$. The personalization vector $v = [p^u(1), \cdots, p^u(m)]$ for each $u$ is defined based on his known ratings on the items

$$p^u(i) = \begin{cases} \frac{e^{r_{u,i}-\bar{r}_u}}{1+\sum_{i\in I_u} e^{r_{u,i}-\bar{r}_u}} & \text{if } i \in I_u \\ \frac{1}{n-|I_u|} \cdot \frac{1}{1+\sum_{i\in I_u} e^{r_{u,i}-\bar{r}_u}} & \text{otherwise.} \end{cases} \tag{2.10}$$

Following the definition 2.10, the higher the rating $r_{u,i}$ that $u$ assign to $i$, the more probable that user would teleport to $i$, thus increasing the stationary distribution at $i$. For the unknown ratings, the teleport probability is evenly distributed.

#### Similarity measure

One of the most important factors that affect the result of memory based methods for collaborative filtering is how the similarity between users or items is measured. In this part, we review several useful similarity measures and discuss their merits and drawbacks. Note that the similarity measure is between two vectors.

Ideally, for similarity between two users, two vectors both of length $M$ are used as input. However, in collaborative filtering, it is often the case that not all of the entries for both vectors are known. In fact, it may even be the case where only a few entries of these two vectors are known. Note that we now face the choice of two possible input vectors. One is to fill all the unknown entries with $0$ and then measure the similarity between these two length $M$ vectors. Another choice is to select the entries that both users have rated and use these two shortened vectors as the input vectors.

In collaborative filtering, we generally use the second approach. The first approach would overestimate the similarity between two vectors because of the filled entries are all $0$ and this is generally not desirable. The problem associated with the second approach is, however, the impact of sparsity problem is more severe. Take the user similarity for example, only the items that are rated by both users can be used as input. Given that there might be thousands of items to rate while users generally rate only a few items, the number of items that two users rate in common could be very small. This is essentially data sparsity problem. One extreme case would be two users only rate one item in common. If these two users happen to give the same ratings for this item, their similarity would be $1$. It could be the case that these two users are indeed share common interests. However, there is also a possibility that they are have quite different tastes and in this case, the similarity is overestimated and could lead to inaccurate predictions. We defer the discussion of possible methods to alleviate this problem to later parts. We now introduce various similarity measures. We assume that we are talking about user similarity unless otherwise specified.

**Pearson Correlation Coefficient** Pearson correlation coefficient (PCC) is a similarity measure between two vectors. In essence, it measures the correlation between two vectors with respect to the product of

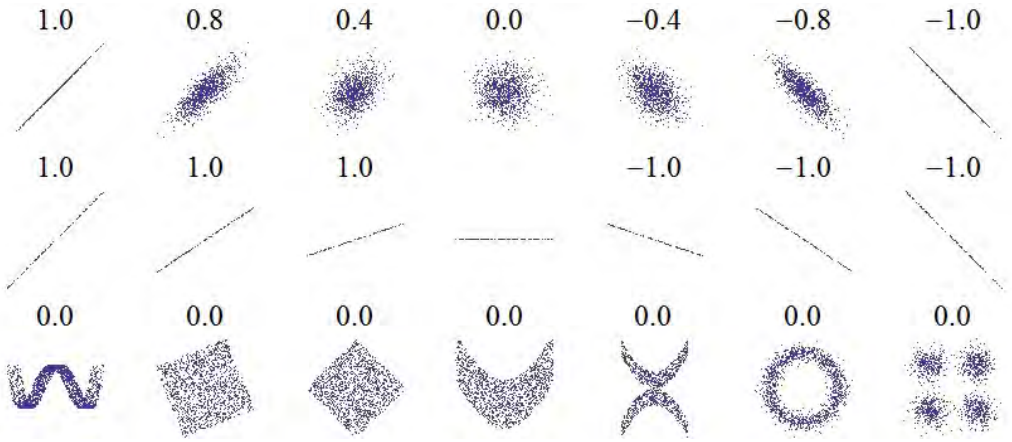



Figure 2.1: Pearson correlation coefficient

their standard deviation. In our setting, it is defined as following:

$$s_{u,v} = \frac{\sum_{i \in I_u \cap I_v} (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\Big[ \sum_{i \in I_u \cap I_v} (r_{u,i} - \bar{r}_u)^2 \sum_{i \in I_u \cap I_v} (r_{v,i} - \bar{r}_v)^2 \Big]^{1/2}}. \tag{2.11}$$

From the definition in equation 2.11, we see that PCC measure the adjusted correlation between two vectors. If two vectors were not correlated, their PCC would be $0$. However, refer to Figure 2.1[2], we see that the reverse is not always true. Note that on the third row of figure 2.1, there is correlation between the datasets but the correlation would still be $0$. PCC is a widely used measurement for calculating user similarities in neighborhood-based methods. PCC alleviates the bias problem by adjusting ratings with their mean.

**Vector similarity** Vector similarity is also known as cosine similarity. It measures the cosine value of the angle between two vectors in high

[2]This figure is adopted from Wikipedia, http://en.wikipedia.org/wiki/Pearson_product-moment_correlation_coefficient

dimensional space. Its definition is given in equation 2.12

$$s_{u,v} = \frac{\sum_{i \in I_u \cap I_v} r_{u,i} \cdot r_{v,i}}{\Big[\sum_{i \in I_u \cap I_v} r_{u,i}^2 \sum_{i \in I_u \cap I_v} r_{v,i}^2\Big]^{1/2}}. \tag{2.12}$$

This is essentially the dot product divided by the product of the lengths vectors.

**Adjusted vector similarity** Adjusted vector similarity is the cosine similarity adjusted with the mean. It has been shown that adjusted vector similarity is the most effective measure in item based methods [117]. It is computed as:

$$s_{u,v} = \frac{\sum_{i \in I_u \cap I_v} (r_{u,i} - \bar{r}_i) \cdot (r_{v,i} - \bar{r}_i)}{\Big[\sum_{i \in I_u \cap I_v} (r_{u,i} - \bar{r}_i)^2 \sum_{i \in I_u \cap I_v} (r_{v,i} - \bar{r}_i)^2\Big]^{1/2}}. \tag{2.13}$$

Similar to PCC, adjusted vector similarity alleviate bias problem by adjusting ratings with their mean.

**Kendall Rank Correlation Coefficient** Similarity measurements discussed so far, including PCC, cosine and adjusted cosine similarity, are all rating based measure. Liu et al. proposed to use a ranking oriented similarity measure in Eigenrank. The motivation to use a ranking oriented measure in ranking oriented methods is to produce more accurate similarity measure and is more robust against bias problem.

Consider the ratings that two users $u$ and $v$ assigned to a set of items. Suppose $u$ assigned $\{3, 5, 4\}$ and $v$ assigned $\{1, 4, 2\}$. The ratings assigned are different for the same item by $u$ and $v$. However, in terms of preference, we see that both users prefer item $2$ to $3$ to $1$. If we could model the similarity between $u$ and $v$ using ranking, we would be able to retain the information that they have very

similar preference even though their rating behavior is quite different. Kendall rank correlation coefficient (KRCC) [85] is a ranking similarity measure that could serve our purpose:

$$s_{u,v} = 1 - \frac{4 \times \sum_{i,j \in I_u \cap I_v} I^-((r_{u,i} - r_{u,j})(r_{v,i} - r_{v,j}))}{|I_u \cap I_v| \cdot (|I_u \cap I_v| - 1)}. \quad (2.14)$$

$I^-$ here is an indicator function defined as:

$$I^- = \begin{cases} 1 & \text{if } x < 0 \\ 0 & \text{otherwise.} \end{cases} \quad (2.15)$$

KRCC can be intuitively interpreted as a value that discounts a specific value for each pair of items on which $u$ and $v$ do not agree. As long as the preference is the same, KRCC will not discount the value.

### 2.3.3 Model-based methods

Model-based approaches provide a systematic way to train a predefined compact model in the training phase that explains observed ratings, which is then used to make predictions. Usually, model-based collaborative filtering methods can achieve better performance [114]. There are both rating-oriented methods [46, 80, 104, 114, 115, 121] and ranking-oriented methods [76, 121]

Methods falling into this category include Aspect Model [46], Bayesian Networks [104], and Matrix Factorization (MF) based approaches [80, 114, 115, 121]. The state-of-the-art model-based methods include restricted Boltzmann machines [116], SVD++ [56, 58], Probabilistic Matrix Factorization (PMF) [114], and multi-domain collaborative filtering [137], graphical models [52], pair-wise tensor factorization [111], and matrix factorization with social regularization [82], etc. Aspect model, also known as Probabilistic Latent Semantics Analysis (PLSA) is a community-based model. It models

an active user's behavior as a mixture of latent communities' behaviors to which the user belongs, according to the mixing proportions. The mixing proportions that a user belongs to each latent community as well as communities' rating behaviors are inferred during training phase. Unknown rating can then be predicted efficiently afterwards. Matrix factorization based methods use low-rank matrix to approximate user item rating matrix. The underlining assumption is that user's behavior is determined by only a few factors that affect his/her preferences. Correspondingly, item have features indicating to what extent these factors affect this item. Matrix factorization based methods are less prone to sparseness problem and scale better to large datasets. They approach the problem by learning two low-rank matrices, one captures users' latent features and the other capture items' latent features. A linear model of these two matrices is fitted to the observed ratings. Prediction is then made using the recovered matrix obtained from the product of the two learned matrices. Due to their success in prediction, variants of MF-based collaborative filtering methods have been developed [112]. Most of previously proposed model-based methods are rating-oriented. They try to fit the observed ratings directly as close as possible so as to make good prediction for the unseen ratings. However, an essential objective of recommender systems is to rank the items according to users' preferences and then to present the top-$N$ list to users. They do not directly solve the ranking problem. To remedy this shortcoming, several ranking-oriented methods are proposed. To address the ranking problem directly, several ranking-oriented models have been proposed. Probabilistic Latent Preference Analysis (PLPA) [76] is a pairwise model-based method extended from Aspect Model. It also assumes that user's behavior is determined by the mixture of latent communities that user belongs to. For each community, pairwise preferences over all items are modeled using Bradley-Terry model [49, 85]. For an active user, a ranking is produced as a fusion of the pairwise preferences. List-wise MF-based method using top-

one probability [22] has been proposed recently [121]. It employs cross entropy as the loss function to measure divergence between ranking distributions defined on observed ratings and on predictive ratings.

Model-based methods hold several appealing properties. First, they often have a clear interpretation. Secondly, once trained, they can produce predictions much more efficiently compared with memory-based methods, which is very importance to commercial applications such as online E-commerce websites. Finally, we are able to gain specific insights concerning the community structure in several latent feature-based methods. Model based methods utilize statistical techniques to build a compact model. In neighborhood-based model, when predicting a rating, we might need all known ratings in order to calculate similarity and make prediction. That is, all the known ratings have to be remembered to make a prediction. On the contrary, in model based methods, the can roughly divide the whole procedure into two phases. In the first phase, we utilize known ratings to *train* a compact model that explains the observation. In the second phase, we use the trained model to produce predictions. In previous section, we see that neighborhood-based methods are not very efficient when making prediction because of the similarity computation requires $O(N^2)$ for user based methods and $O(M^2)$ for item based methods. For EigenRank, it is even worse because we have to calculate pairwise preference for every user, which requires $O(NM^2)$ time complexity. This could be prohibitive for large online recommender system like Amazon or IMDB, which have tens of millions of users and tens of thousands of items. On the other hand, model-based methods are usually very efficient at making predictions once the model has been built. Most computation sink is on the training phase.

In the following, we will discuss two rating oriented model and two ranking oriented model.

**Probabilistic Matrix Factorization**

Probabilistic Matrix Factorization (PMF) [114] is one of the state-of-the-art methods in rating oriented collaborative filtering. It is named so because it decompose the user item rating matrix $R$ into the product of two latent feature matrices $U$ and $V$. Let $R$ be $N \times M$ matrix, where $N$ is the number of users and $M$ is the number of items. Then $U$ is a $K \times N$ matrix and $V$ is a $K \times M$ matrix. PMF tries to find $U$ and $V$ so that $U^T V$ recovers $R$ the best. Latent user feature matrix $U$ models users' latent preferences. A column $U_u$ is the latent feature for user $u$. Similarly $V$ models items' latent features. Column vector $V_i$ is the latent item feature for item $i$.

**Model assumption** In PMF, we adopt a probabilistic model that explains the observation ratings. We assume that the observed rating $r_{u,i}$ is drawn from a Gaussian distribution with mean $U_u^T V_i$ and variance $\sigma^2$. Thus the conditional probability of observing $R$ given $U$ and $V$ would be:

$$p(R|U, V, \sigma^2) = \prod_{i=1}^{N} \prod_{j=1}^{M} \left[ \mathcal{N}(R_{i,j}|U_i^T V_j, \sigma^2) \right]^{I_{i,j}}, \tag{2.16}$$

where $\mathcal{N}(x|\mu, \sigma^2)$ is Gaussian distribution with mean $\mu$ and variance $\sigma^2$. Its definition for scalar valued variable is given as

$$\mathcal{N}(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{ -\frac{(x-\mu)^2}{2\sigma^2} \right\}. \tag{2.17}$$

$I_{i,j}$ in Eq. 2.16 is an indicator function which gives 1 if $i$ have rated $j$ and 0 otherwise. We also place 0 mean spherical Gaussian priors [114, 32, 124] on $U$ and $V$:

$$p(U|\sigma_U^2) = \prod_{i=1}^{N} \mathcal{N}(U_i|0, \sigma_U^2) \tag{2.18}$$

$$p(V|\sigma_V^2) = \prod_{j=1}^{N} \mathcal{N}(V_j|0, \sigma_V^2) \tag{2.19}$$

**Model posterior** Given the definition of $p(R|U, V, \sigma^2)$, $p(U|\sigma_U^2)$ and $p(V|\sigma_V^2)$, we can write the posterior probability of the parameter $U$ and $V$ as proportional to the product of conditional probability $p(R|U, V, \sigma^2)$ and the prior probability $p(U|\sigma_U^2)$, $p(V|\sigma_V^2)$:

$$p(U, V|R, \sigma^2, \sigma_U^2, \sigma_V^2) \propto p(R|U, V, \sigma^2)p(U|\sigma_U^2)p(V|\sigma_V^2). \tag{2.20}$$

Following Eq. 2.20, the log likelihood of the posterior can be written as:

$$\begin{aligned} &\ln p(U, V|R, \sigma^2, \sigma_U^2, \sigma_V^2) \\ =& -\frac{1}{2\sigma^2}\sum_{i=1}^{N}\sum_{j=1}^{M} I_{i,j}(R_{i,j} - U_i^T V_j)^2 - \frac{1}{2\sigma_U^2}\sum_{i=1}^{N} U_i^T U_i - \frac{1}{2\sigma_V^2}\sum_{j=1}^{M} V_j^T V_j \\ &-\frac{1}{2}\Big(\sum_{i=1}^{N}\sum_{j=1}^{M} I_{i,j}\ln \sigma^2 + KN\ln \sigma_U^2 + KM\ln \sigma_V^2\Big) + C, \end{aligned} \tag{2.21}$$

where $C$ is a constant that do not depend on input parameters. Maximizing the loglikelihood defined in Eq. 2.21 is equivalent to minimizing the following loss:

$$\mathcal{L} = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{M} I_{i,j}(R_{i,j} - U_i^T V_j)^2 + \frac{\lambda_U}{2}\|U\|_F^2 + \frac{\lambda_V}{2}\|V\|_F^2. \tag{2.22}$$

The first term of Eq. 2.22 is the Root Mean Squared Error (RMSE) for all the observed ratings unnormalized. In Eq. 2.22, $\lambda_U = \sigma^2/\sigma_U^2$ and $\lambda_V = \sigma^2/\sigma_V^2$ and $\|\cdot\|_F$ denote the Frobenius norm. This loss function is unnormalized RMSE (first term) plus regularization term (second and third term) to avoid over-fitting.

**Model inference** The objective function in training phase is to minimize the loss defined in Eq. 2.22. The dot product $U_i^T V_j$ breaks the convex property of $\mathcal{L}$ and global minima is not achievable in general. However, by employing optimization methods such as gradient descend we can approach a local minimum. First compute the partial derivative of $\mathcal{L}$ with respect to $U$ and $V$

$$\frac{\partial \mathcal{L}}{\partial U_i} = \sum_{j=1}^{M} I_{i,j}(U_i^T V_j - R_{i,j})V_j + \lambda_U U_i, \tag{2.23}$$

$$\frac{\partial \mathcal{L}}{\partial V_j} = \sum_{i=1}^{N} I_{i,j}(U_i^T V_j - R_{i,j})U_i + \lambda_V V_j. \tag{2.24}$$

Fix a learning rate $\eta$, we can use the following update formula to iteratively refine $U$ and $V$:

$$U_i^{t+1} = U_i^t - \eta \frac{\partial \mathcal{L}}{\partial U_i^t}, \tag{2.25}$$

$$V_j^{t+1} = V_j^t - \eta \frac{\partial \mathcal{L}}{\partial V_j^t}. \tag{2.26}$$

Time complexity is linear with the number of observed ratings for one iteration in update. In practice, it takes dozens of iterations to converge to a local minimum. Compared with neighborhood-based methods, the training of PMF is very efficient.

**Rating prediction** Once the model is trained, predictions are calculated as dot production of two vectors:

$$\hat{r}_{u,i} = U_u^T V_i. \tag{2.27}$$

Note that $\hat{r}_{u,i}$ is the expected value of $r_{u,i}$, i.e. $\hat{r}_{u,i} = E[r_{u,i}]$, under PMF's model assumption.

**Extensions** In practice, instead of fitting $U$ and $V$ to the user item rating matrix $R$, we can filter the dot product $U_u^T V_j$ through a logistic function. Useful function would be logistic sigmoid function $g = 1/(1 + \exp(-x))$. Filter through this function would always give output in $(0, 1)$. In order for PMF to be able to fit the observations, we should map the observed ratings in $R$ from $[1, D]$ to the interval $[0, 1]$ and $t = (x - 1)/(D - 1)$ does the trick.

Since PMF give very accurate rating prediction, there have been several follow up works that try to extend PMF to more general assumptions or incorporating more information.

Bayesian Probabilistic Matrix Factorization (BPMF) [115] is a Bayesian treatment of PMF. In PMF, the prior of $U$ and $V$ is spherical Gaussian. That is, the covariance is of the form $\sigma^2 I$ where $I$ is the identity matrix. Here the assumption is that the correlation is $0$, which is not always true. BPMF generalize PMF by placing a Gaussian-Wishart prior on the user and movie hyperparameters and thus obtain a Bayesian model. However, this generalization makes the model significantly more complex and becomes intractable to solve. So the Salakhutdinov et al. proposed to use Markov Chain Monte Carlo to do the inference.

Other extensions to PMF include incorporating side information, for example social network information. Here we can do so because we have latent user feature matrix $U$. So we can utilize $U$ to explain users' relationship on social network. If two users share similar interests, they are more likely to be friends. There are methods that leveraged social [81, 82] and side information [136] to improve the performance. The basic idea is to decompose user item rating matrix $R$ and trust social matrix using the same latent user features $U$.

#### Probabilistic Latent Semantic Analysis

Probabilistic Latent Semantic Analysis (pLSA) [45] is a popular model that has seen wide adoption in topic modeling field. Hofmann adopted it to the collaborative filtering problem[46]. To explain the

observed ratings, a latent class variable $z$ is introduced and the probability of observing a rating $r_{u,i}$ is decomposed into:

$$p(r_{u,i}) = \sum_{z=1}^{K} p(r_{u,i}|i,z)p(z|u). \tag{2.28}$$

This model makes the assumption that each user $u$ is distributed on $K$ latent class (community) according to his interests. And the rating that $u$ would assign to $i$ is the weighted average of class behavior. The latent variable $z$ can be interpreted as community in user movie scenario. For example, some users may be fond of Sci-Fi movies and in general assign high ratings to such movies. These users are mainly recognized as belonging to Sci-Fi community. This could be well modeled in pLSA by assigning a large probability $p(z|u)$ to $z$, which corresponds to Sci-Fi community. And $p(r|z,i)$ are modeled in such a way that it assigns high ratings to Sci-Fi movies.

**Model assumption** In pLSA, we assume that the probability that latent class $z$ assign item $i$ rating $r$ following a Gaussian Distribution $\mathcal{N}(r|\mu_{zi}, \sigma_{zi}^2)$. $\mu_{zi} \in \mathbb{R}$ and $\sigma_{zi}^2 \in \mathbb{R}^+$ are Gaussian mean and variance. User mixing probability $p(z|u)$ follows categorical distribution:

$$\sum_{z=1}^{K} p(z|u) = 1. \tag{2.29}$$

Based on these assumptions, the log likelihood of all the observed

ratings is:

$$\begin{aligned}\mathcal{L} &= \sum_{(u,i)\in R} \log\left(p(r_{u,i}|u,i)\right) \\ &= \sum_{(u,i)\in R} \log\Big(\sum_{z=1}^{K} p(r_{u,i}|z,i)p(z|u)\Big) \\ &= \sum_{(u,i)\in R} \log\Big(\sum_{z=1}^{K} \mathcal{N}(r_{u,i}|\mu_{zi},\sigma_{zi})p(z|u)\Big).\end{aligned} \tag{2.30}$$

**Model inference** Given the definition of the log likelihood, for pLSA to best explain all the rating observed so far, we want to maximize $\mathcal{L}$ as is defined in Eq. 2.30. Note that maximizing $\mathcal{L}$ directly is intractable due to the sum inside the log, which cannot be reduced although $\mathcal{N}$ belongs to exponential family. Standard approach to solve such optimization problem is Expectation Maximization (EM) [10] algorithm, which alternates between an expectation step and a maximization step. The observation is that if we have *complete* data which includes the latent class information, i.e. $u, i, r, z$, we can write the log likelihood in a form that can be solved analytically. Since we do not have complete data, we approximate this complete data by the posterior probability of latent class variable $z$. So, in expectation step, we compute the posterior probability of $z$:

$$p(z|u,i) = \frac{p(z|u)p(r_{u,i}|z,i)}{\sum_{z'=1}^{K} p(z'|u)p(r_{u,i}|z',i)}. \tag{2.31}$$

Then in maximization step, we approximate the complete log likelihood using the expectation over the posterior:

$$E[\mathcal{L}^c] = \sum_{(u,i)\in R}\sum_{z=1}^{K} p(z|u,i)\log\Big\{p(r_{u,i}|z,i)p(z|u)\Big\} \tag{2.32}$$

Notice that there is no sum inside the log, it turns out we can solve the optimization problem in closed form. Introducing a Lagrange multiplier to enforce the $p(z|u)$ follows categorical distribution, i.e. $\sum_z p(z|u) = 1$, solving the constraint optimization problem gives us the following closed form solutions:

$$p(z|u) = \frac{\sum\limits_{(u',i)\in R:u'=u} p(z|u,i)}{\sum\limits_{z'=1}^{K} \sum\limits_{(u',i)\in R:u'=u} p(z'|u,i)} \tag{2.33}$$

$$\mu_{zi} = \frac{\sum\limits_{(u,i')\in R:i'=i} r_{ui} p(z|u,i)}{\sum\limits_{(u,i')\in R:i'=i} p(z|u,i)} \tag{2.34}$$

$$\sigma_{zi}^2 = \frac{\sum\limits_{(u,i')\in R:i'=i} (r_{ui} - \mu_{zi})^2 p(z|u,i)}{\sum\limits_{(u,i')\in R:i'=i} p(z|u,i)} \tag{2.35}$$

We train the model in an iterative manner. In each iteration, expectation step is executed first and $p(z|u,i)$s are calculated. Then in maximization step, $p(z|u), \mu_{zi}, \sigma_{zi}^2$s are calculated using Eq. 2.33, Eq. 2.34 and Eq. 2.35. EM algorithm is guaranteed to converge. And in experiments it takes less than a hundred iterations to converge.

**Rating Prediction** Once all the parameters are learnt in the training phase, we can make prediction on unknown rating using the expected value:

$$\hat{r}_{u,i} = \int r p(r|u,i) = \sum_{z=1}^{K} p(z|u) \int r p(r|u,i) = \sum_{z=1}^{k} p(z|u)\mu_{zi}. \tag{2.36}$$

Once all the parameter is learnt, we can output the prediction in $O(K)$ time.

**Probabilistic Latent Preference Analysis**

Probabilistic Latent Preference Analysis (pLPA) [76] following the idea of pLSA. Unlike pLSA, which is a rating oriented model, pLPA tries to model the preferences of users directly and thus fall into the category of ranking oriented model. pLPA is a pair-wise ranking model. Statistical tool Bradley-Terry model is used as the "core" of pLPA. So before discuss pLPA in detail, we will first review Bradley-Terry model.

**Bradley-Terry model** Bradley-Terry model defines a distribution over the rank of a set of $M$ items using a vector parameter $\gamma \in \mathbb{R}_M^+$. Let us focus on a simpler problem, where there are only two items $i$ and $j$. Following [76], we use $\delta_{ij} = 1$ to denote that $i$ is preferred to $j$ and $\delta_{ij} = 0$ otherwise. Then Bradley-Terry model models the probability of $\delta_{ij}$ to be:

$$p(\delta_{ij} = 1|\gamma) = \frac{\gamma_i}{\gamma_i + \gamma_j} \tag{2.37}$$

The parameter used in Bradley-Terry model could be interpreted as the utility associated with the item. So a higher $\gamma_i$ would imply that it is more probable that $i$ would be preferred over other items.

The Bradley-Terry model defined for a pair of items can be extended to model the probability of ranking $\pi$ over $M$ items[49, 85]. Let $\mathbf{P}_M$ denote the set of all possible permutations over integers from 1 to $M$. Let $\pi_i = 1$ indicates that item $i$ is ranked first in ranking $\pi$. Similarly, $\delta_{ij}^{\pi} = 1$ denote the fact that $i$ is ranked higher than $j$ in ranking $\pi$, i.e. $\pi_i < \pi_j$. The probability over all possible ranking $\mathbf{P}_M$ is

$$p(\pi|\gamma) = \frac{1}{C(\gamma)} \prod_{i=1}^{M-1} \prod_{j=i+1}^{M} p(\delta_{ij}^{\pi}|\gamma), \tag{2.38}$$

in which $C(\gamma)$ is the normalization term to ensure that Eq. 2.38 is a

probability measure:

$$C(\gamma) = \sum_{\pi \in \mathbf{P}_M} \prod_{i=1}^{M-1} \prod_{j=i+1}^{M} p(\delta_{ij}^{\pi} | \gamma). \tag{2.39}$$

Substitute Eq. 2.37 into Eq. 2.38 and we obtain:

$$p(\pi|\gamma) = \frac{1}{C^*(\gamma)} \prod_{i=1}^{M} \gamma_i^{M-\pi_i}, \tag{2.40}$$

where $C^*(\gamma)$ is normalization term defined as:

$$C^*(\gamma) = C(\gamma) \prod_{i=1}^{M-1} \prod_{j=i+1}^{M} (\gamma_i + \gamma_j). \tag{2.41}$$

From this definition, we see that $C^*(\gamma)$ is a constant term. So, from Eq. 2.40, we see that there is a single most probably ranking $\pi^* = \text{argmax}_\pi p(\pi|\gamma)$. It can be obtained by sorting the value of $\gamma_i$s in decreasing order and output the corresponding $i$s as the rank.

**Model assumption** The Bradley-Terry model forms the core of pLPA. In pLSA, the observation is the triplet $(u, i, r)$ and we use this observation to fit the model. Analogously we model the preferences observed. Let us define that the preference of triplet $(u, i, j)$ is observed if $r_{u,i}$ and $r_{u,j}$ is both observed and $r_{u,i} \neq r_{u,j}$. Again, we let $\delta_{ij}^u = 1$ if $r_{u,i} > r_{u,j}$ as the observed preference. Let $\mathcal{Q}$ be the set of the triplet $(u, i, j)$ for which the preferences we observed.

Similar to pLSA, each observed preference is modeled as a mixing distribution:

$$p(\delta_{ij}^u | u, i, j) = \sum_{z=1}^{K} p(\delta_{ij}^u | z, i, j) p(z|u), \tag{2.42}$$

in which $p(\delta_{ij}^u | z, i, j)$ is a Bradley-Terry model parameterized by $\gamma_z$:

$$p(\delta_{ij}^u | z, i, j) = p(\delta_{ij} | \gamma_z) = \frac{\gamma_{zi}}{\gamma_{zi} + \gamma_{zj}}. \tag{2.43}$$

The log likelihood of the model can be computed as:

$$\mathcal{L} = \sum_{(u,i,j)\in\mathcal{Q}} \log \sum_{z=1}^{K} p(\delta_{ij}^{u}|z,i,j)p(z|u). \tag{2.44}$$

**Model inference** Notice that $\mathcal{L}$ defined in Eq. 2.44 shares the same form as that in Eq. 2.30. Again we can use EM algorithm to solve this optimization problem. In expectation step, the posterior probability of latent class variable for each observed pairwise preference is computed:

$$p(z|u,i,j) = \frac{p(z|u)p(\delta_{ij}^{u}|\gamma_z)}{\sum_{z'=1}^{K} p(z'|u)p(\delta_{ij}^{u}|\gamma_z')}. \tag{2.45}$$

In the maximization step, the expected complete log likelihood $E[\mathcal{L}^c]$ is calculated as:

$$E[\mathcal{L}^c] = \sum_{(u,i,j)\in\mathcal{Q}} \sum_{z=1}^{K} p(z|u,i,j)[\log p(\delta_{ij}^{u}|\gamma_z) + \log p(z|u)]. \tag{2.46}$$

Optimize $E[\mathcal{L}^c]$ with respect to $p(z|u)$ with the constraint $\sum_{z=1}^{K} p(z|u) = 1$ gives the following update formula:

$$p(z|u) = \frac{\sum_{(u',i,j)\in\mathcal{Q}:u'=u} p(z|u,i,j)}{\sum_{z'=1}^{K} \sum_{(u',i,j)\in\mathcal{Q}:u'=u} p(z|u,i,j)}. \tag{2.47}$$

However, there is no closed form solution for $\gamma_z$ as is in pLSA. Hunter proposed an iterative algorithm for obtaining maximum likelihood estimates for parameters in Bradley-Terry model [49]. We won't dig into the details of this algorithm. The update formula for $\gamma_z$ is given below:

$$\gamma_{zi}^{(t+1)} = W_i^z \Big[ \sum_{j\neq i} \frac{N_{ij}^z}{\gamma_{zi}^t + \gamma_{zj}^t} \Big], \tag{2.48}$$

where in the equation, $W_i^z = \sum_{j=1}^{M} \omega_{ij}^z$, $N_{ij}^z = \omega_{ij}^z + \omega_{ji}^z$ and

$$\omega_{ij}^z = \sum_{(u,i,j)\in\mathcal{Q}} p(z|u,i,j) \tag{2.49}$$

is the expected number of times that $i$ is preferred to $j$ in latent class $z$.

So in each maximization step, we would use Eq. 2.48 iteratively to calculate $\gamma$ until converge.

**Ranking prediction** Once learnt all the model parameters, we can now turn to the problem of ranking prediction. Given a user $u$, we can compute the probability of $u$ giving a ranking $\pi \in \mathbf{P}_M$:

$$\begin{aligned} p(\pi|u) &= \frac{1}{C(u)} \prod_{i=1}^{M-1} \prod_{j=i+1}^{M} p(\delta_{ij}^{\pi}|u) \\ &= \frac{1}{C(u)} \prod_{i=1}^{M-1} \prod_{j=i+1}^{M} \sum_{z=1}^{K} p(\delta_{ij}^{\pi}|\gamma_z) p(z|u). \end{aligned} \tag{2.50}$$

We want to find

$$\pi^* = \text{argmax}_{\pi\in\mathbf{P}_M} p(\pi|u). \tag{2.51}$$

This turns out to be a NP-complete problem by reduction from cyclic-ordering problem [26].

Thus we have to resort to approximate methods to solve Eq. 2.51. Since the model parameter $\gamma_{zi}$ could be interpreted as the utility that latent class $z$ assign to $i$, we could approximate the utility that user $u$ would assign $i$ to be:

$$\theta_{ui} = \sum_{z=1}^{K} p(z|u)\gamma_{zi}. \tag{2.52}$$

Once we have obtained $\theta$, we can sort $\theta$ in decreasing order and output the corresponding rank as the prediction. This matches our intuition well in the sense that the item with the largest expected utility is ranked highest in the prediction.

**List-wise ranking using top one probability**

All the model talked so far, including memory based and model based, are either point-wise or pair-wise methods. Objective in point-wise models, i.e. rating oriented models, is to minimize predicted rating error. This could give suboptimal ranking output as demoed. The problems associated with pair-wise ranking models, including EigenRank and pLPA, are that the time complexity is prohibitive for it to be applied in large dataset. A good compromise would be to model ranking problem using list-wise model. In this section, we review a list-wise ranking oriented collaborative filtering model proposed by Shi et al. [121].

To model the problem on a list-wise basis, we need to define a probability over ranking on a list-wise manner. Note that Bradley-Terry model mentioned in previous model is essentially a pair-wise model. Thus, we resort to a newly proposed probability over ranking that is list-wise in nature.

**Top one probability** Top one probability models the probability that an item is ranked on the first place in the ranking. It has a generalized version, top $k$ probability, which consider the top ranking $k$ items instead. The top one probability associated with an item $i$ in a ranking $\pi$ for user $u$ is defined using the ratings that $u$ assign to $i$:

$$p_R(\pi_i = 1|u, i) = \frac{\varphi(r_{u,i})}{\sum_{i'=1}^{M} \varphi(r_{u,i'})}, \tag{2.53}$$

where $\varphi$ is a monotonically increasing and positive valued function. We use exponential function $\exp$ in this thesis consistently.

Since in the problem setting, not all ratings are observed, we instead model the top one probability over the observed items $I_u$ for user $u$:

$$p_R(\pi_i = 1|u, i) = \frac{\varphi(r_{u,i})}{\sum_{i' \in I_u} \varphi(r_{u,i'})}. \tag{2.54}$$

The subscript $R$ in $p_R$ means that this top one probability is defined using rating $R$.

**Model assumption** Like in PMF, we introduce latent user $K \times N$ matrix $U$ and latent item $K \times M$ matrix $V$. The rating $r_{u,i}$ given by $u$ to $i$, is related to the dot product $U_i^T V_j$. This model is different from PMF because it uses a different optimization function. In PMF, we try to match $U_i^T V_j$ to $r_{u,i}$ as accurate as possible. However, in list-wise top one model, we define the loss function to be the cross entropy of the top one distribution using true ratings and the top one distribution using the dot product of the latent feature vectors. Cross entropy of two distribution $p$ and $q$ is defined as:

$$H(p, q) = E_p[-\log q]. \tag{2.55}$$

This quantity could be interpreted as distance measure between two distributions and is minimized when $p = q$. The loss function of top one ranking model is defined as:

$$\begin{aligned}
&\mathcal{L}(U, V) \\
&= \sum_{i=1}^{N} \Big\{ - \sum_{j=1}^{M} p_R(r_{i,j}) \log \{p_{U,V}(g(U_i, V_j))\} \Big\} + \frac{\lambda_U}{2} \|U\|_F^2 + \frac{\lambda_V}{2} \|V\|_F^2 \\
&= \sum_{i=1}^{N} \left\{ - \sum_{j=1}^{M} I_{i,j} \frac{\exp(r_{i,j})}{\sum_{k=1}^{M} I_{ik} \exp(r_{i,k})} \log \left\{ \frac{\exp(g(U_i^T V_j))}{\sum_{k=1}^{M} I_{ik} \exp(g(U_i^T V_k))} \right\} \right\} \\
&\quad + \frac{\lambda_U}{2} \|U\|_F^2 + \frac{\lambda_V}{2} \|V\|_F^2.
\end{aligned} \tag{2.56}$$

Contrast Eq. 2.56 with PMF's loss in Eq. 2.22, we see that in Eq. 2.22, the core part is the squared loss while in Eq. 2.56, the core part is the cross entropy $H[p_R, p_{UV}]$. Both loss functions have regularization terms to avoid over-fitting. In Eq. 2.56, $I_{ij}$ is an indicator function that is 1 if $i$ have rated $j$ and 0 otherwise. $g$ is the logistic sigmoid

function $1/(1+\exp(-x))$ and used here to add "non-linearity" to the model. We have to map $r_{i,j}$ to scale $[0, 1]$.

**Model inference** As is in PMF, the loss defined in Eq. 2.56 is again non-convex and in general we cannot achieve a global optimal. However, by alternatively fixing one component, i.e. $V$, and update the other, i.e. $U$, loss would converge to a local minima. Partial derivative of $L$ with respect to $U$ and $V$ is given:

$$\frac{\partial \mathcal{L}}{\partial U_i} = \sum_{j=1}^{M} I_{ij} \left\{ \frac{\exp(g(U_i^T V_j))}{\sum_{k=1}^{M} I_{ik} \exp(g(U_i^T V_k))} - \frac{\exp(r_{i,j})}{\sum_{k=1}^{M} I_{ik} \exp(r_{i,k})} \right\} g'(U_i^T V_j) V_j + \lambda_U U_i, \tag{2.57}$$

$$\frac{\partial \mathcal{L}}{\partial U_i} = \sum_{i=1}^{N} I_{ij} \left\{ \frac{\exp(g(U_i^T V_j))}{\sum_{k=1}^{M} I_{ik} \exp(g(U_i^T V_k))} - \frac{\exp(r_{i,j})}{\sum_{k=1}^{M} I_{ik} \exp(r_{i,k})} \right\} g'(U_i^T V_j) U_i + \lambda_V V_j. \tag{2.58}$$

Then define a learning rate $\eta$, we can reuse the update formula Eq. 2.25 and Eq. 2.26 to calculate $U$ and $V$ until converge.

**Ranking prediction** Once the model is learnt, we can output the ranking that would maximize top one probability $p_{U,V}$. It is easy to see that this is equivalent to sort the items using $U_i^T V_j$ and output the corresponding rank.

## 2.4 Reputation Estimation

Reputation estimation refers to the study of systematically assign reputation scores for users (domains, email senders, etc.) to facilitate the detection of spam users and mitigate the effect of spam users. Reputation estimation systems have been successfully applied to spam email detection [37, 25, 62, 16], social network zombie user detection [126] and voice (VoIP) spam filtering [33, 27].

There are two different scenarios where reputation estimation methods come into play. In one line of research, the trustiness of a user and the propagation of the trust are studied [40, 100]. The reputation of a user is calculated based on the trustiness of the user. Such reputation estimation methods are suitable for a graph based system like social networks. Another line of study that operates on rating systems is more relevant to recommender systems. Users' reputations are estimated in an online rating environment such as the rating collection module of a recommender system. Usually an item is assumed to have an intrinsic quality that is common to all users and users' reputations are calculated based on the deviation between their ratings and the true qualities of the rated items [94, 60]. The intrinsic quality of an item is often calculated using a weighted average of ratings assigned by all users, where the weight of a user's rating depends on the user's reputation. These methods are iterative in nature and some works [94, 60] fall into this paradigm. One immediate concern of such algorithms is whether these reputation scores converge to a fixed solution or oscillate. Recently, there are investigations that guarantee the theoretical convergence of the reputations [29, 63]. These algorithms are not applicable to every rating system since the intrinsic quality assumption might not be true in all cases. For example, these methods may not perform well on music and movie recommendation. In such scenarios, users' tastes play a big role in deciding what score to assign to the item and users have different tastes. The intrinsic quality view does not apply in this personalized environment. Principle Component Analysis (PCA) and PLSA based methods are proposed in [90, 92], which exploit statistical properties to identify spam users in a rating system. PCA learns the low rank structure of the data. However, since it sticks to the principle components, the flexibility of the model is questionable. A reputation weighted strategy is adopted in [91] to improve the robustness of prediction accuracy.

The details of reputation estimation methods in online rating sys-

tems are deferred to Chapter 5, in which we developed a framework where a lot of the methods mentioned can be subsumed.

## 2.5 Topic Modeling

Topic modeling is the study of modeling text corpora and other collections of discrete data. The target of topic modeling is to find a short description of all the members in the collection to facilitate further process such as classification while preserving the essential statistical relationship [11]. In this thesis, we refer to topic modeling as the study of modeling text corpora or more specifically review comments collected in online rating systems. However, in this section, we assume that we are presented a corpus of documents, each of which is comprised of words. We now define the terms that are used when referring to topic modeling in this thesis:

- **word**
  A *word* is the basic unit of the discrete data. In this thesis, a word refers to a word in a natural language such as English or German. All words in a language form a vocabulary and words are indexed by $\{1, \cdots, V\}$.

- **document**
  A *document* is a sequence of $N$ words denoted by $\mathbf{w} = (w_1, w_2, \cdots, w_N)$, where $w_n$ is the $n$th word in the sequence.

- **corpus**
  A *corpus* is a collection of $M$ documents denoted by $D = \{\mathbf{w}_1, \mathbf{w}_2, \cdots, \mathbf{w}_M\}$.

We now review several models for topic modeling, starting from the simplest one unigram model. Then we introduce mixture of unigram model [98], probabilistic latent semantic indexing [45] and lastly Latent Dirichlet Allocation [11].

#### Unigram Model

The unigram model assumes that every word of every document is drawn independently from a single multinomial distribution:

$$p(\mathbf{w}) = \prod_{n=1}^{N} p(w_n). \tag{2.59}$$

It is easy to see that the above model does not have application in real life since it assumes all the documents in the corpus all have the same distribution on words. If this is indeed the case, then all the documents are generated from the same source and have very similar statistics. They belong to the same class and no further classification is required.

#### Mixture of Unigram Model

The Mixture of Unigram Model [98] is a slightly more complex model than unigram model. Where as in unigram model, we assume that a single multinomial distribution governs all the documents, in mixture of unigram model, we assume there are several such multinomial distributions and there is a latent variable indicating which distribution to use for each of the document. Each document still has only one multinomial distribution governs its word distribution. The probability of a document is

$$p(\mathbf{w}) = \sum_{z} p(z) \prod_{n=1}^{N} p(w_n|z). \tag{2.60}$$

Mixture of unigram model can be used for document classification. We learn the multinomial distributions that best explain the documents and each document has a most probable latent variable $z$, which can be used to perform classification. However, there are serious restrictions on the model since each document can have only one multinomial distribution. In real life documents tend to be a mixture of several themes, each of which has a distribution on words.

#### Probabilistic Latent Semantic Indexing

Probabilistic Latent Semantic Indexing (pLSI) [45] is a widely used topic model. Compared with mixture of unigram model, pLSI add another layer of mixture by allowing each document has its own mixing proportion on the latent multinomial distributions on words which we now refer to as topics. This modeling technique greatly expands the potential applicable scenario of pLSI. In mixture of unigram model, we make the restriction that each document can have only one topic (distribution on words). In pLSI, this restriction is removed and it allows each document to have its own mixing proportions of the topics. This is much more realistic and resembles the real documents better. The probability of observing a document $d$ has word $w_n$ in it in a corpus under the assumption of pLSI is

$$p(d, w_n) = p(d) \sum_z p(w_n|z)p(z|d). \tag{2.61}$$

pLSI posits that a document label $d$ and a word $w_n$ are conditionally independent given the unobserved latent topic $z$.

The major drawbacks of pLSI are that there is no assumption on the prior distribution on $p(z|d)$ and $p(w_n|z)$. There are a lot of parameters in pLSI and overfitting is a serious problem in practice [11].

#### Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA) [11] is a more sophisticated topic model that is built on pLSI. LDA choose a full Bayesian treatment of the problem and place Dirichlet distribution on both the word distribution for each topic $p(w_n|z)$ as well as the mixing proportion of each document $p(z|d)$. The basic idea of LDA is that documents are represented as random mixtures over latent topics, where each topic is a distribution over the words in vocabulary. In LDA, each document is generated according to the following procedure:

1. Choose document length $N \sim \text{Poisson}(\xi)$.

2. Choose document topic mixing proportion distribution $\theta \sim \text{Dir}(\alpha)$.

3. For each of the $N$ words $w_n$:

   (a) Choose a topic $z_n \sim \text{Multinomial}(\theta)$.

   (b) Choose a word $w_n$ according to $p(w_n|z_n, \beta)$, which is a multinomial probability conditioned on the topic $z_n$.

Given the above generative process for a document, we can obtain the probability of observing a corpus:

$$P(D|\alpha, \beta) = \prod_{d=1}^{M} \int p(\theta_d|\alpha)(\prod_{n=1}^{N_d} \sum_{z_{dn}} p(z_{dn}|\theta_d)p(w_{dn}|z_{dn}, \beta))d\theta_d. \tag{2.62}$$

The learning of the model can be performed suing Variational Inference method [11] or Collapsed Gibbs Sampler [106].

---

□ **End of chapter.**

# Chapter 3

# Online Collaborative Filtering

## 3.1 Problem and Motivation

With the emergence of large-scale online user-contributed websites and online shopping websites, e.g., Amazon, IMDB, etc., users are presented with unprecedentedly large amount of items. On one hand, users can easily get stuck in the information-overloading problem, and how to select favorite items from millions of options becomes a major bottleneck. On the other hand, it is important for vendors to find out users' preferences so as to boost sales. Recommender systems, aiming at selecting attractive items for users, become one promising technology to resolve the aforementioned problems. They can filter out less interesting items and recommend appealing ones to the corresponding users. Collaborative filtering is one of the major methods to perform recommendation tasks. Traditional collaborative filtering methods adopt batch-trained algorithms. These methods suffer from two major drawbacks. First, they scale poorly. Before training, they require that all data are available. During training, at each iteration, all ratings must be loaded into the memory to perform the algorithms. This is very expensive since real-world datasets like Yahoo!Music which contains more than $250$ million ratings, cannot be loaded into the memory easily. The second drawback of batch-trained algorithms is that they are unsuitable for dynamic ratings. A recommender system may change in one of

the following ways: 1) a new user may join the system and rate existing items; 2) a new item may appear in the system and existing users may purchase and rate it; and 3) existing users may purchase and rate existing items; in other words, ratings are collected over time. In these cases, to capture the change, the batch-trained CF methods have to rebuild the model from scratch. As recommender systems usually contain a huge amount of training data, rebuilding a model is very expensive.

Online learning algorithms emerge as a natural solution to attack the incremental rating problem. In the literature, although there are several tasks [1, 21, 28, 75, 84] investigating online learning for collaborative filtering, they did not explore the complete properties of online algorithms. In addition, none of previous work considers ranking-oriented collaborative filtering. Hence, in this chapter, we study online algorithms from various aspects to solve the issues facing batch-trained CF algorithms and previously proposed online learning algorithms. Our contributions include:

- We apply the proposed online learning framework to both rating-oriented *matrix factorization* (MF) based methods (PMF) and ranking-oriented MF-based methods (RMF). Our proposed online learning algorithms can handle new rating incrementally without retraining the models. To our best knowledge, this is the first attempt to develop online algorithms for ranking-oriented CF methods.

- We develop the online learning algorithms employing two approaches, stochastic gradient descent and the dual-average method. We provide succinct and efficient solutions to both of them. All the online algorithms scale linearly with the number of observed ratings and memory consumption is linear in the number of users and the number of items.

- We further impose different regularizations to explore the properties of the proposed models. For example, we introduce the

$L_1$-regularization to yield sparse solutions while maintaining comparable performance. The parsimony model can therefore reduce the memory cost in storing the model and time cost in predicting the results.

- Finally, we conduct a series of detailed experiments on real-world datasets to demonstrate the merits and properties of the proposed online learning algorithms.

In the literature, online learning algorithms have been extensively studied in content-based filtering [12, 132]. They can be classified into three categories: online learning for maximum margin models [31], algorithms implemented by stochastic gradient method [12], and online learning for sparse models [128, 132]. However, in collaborative filtering, there are only limited investigations. In [75], an online algorithm is developed for memory-based collaborative filtering. In [21], an online algorithm for Non-negative Matrix Factorization (NMF) is considered. In [28], an online algorithm is conducted on a mixture of memory-based and model-based algorithms, where the data are dynamically clustered. The involved model, however, is different from what we consider as the matrix factorization models. Another work is [84], which only applies online algorithms on sparse models in computer vision area. In [1], a gradient descent method on matrix factorization with (or without) features by directly minimizing the square loss is proposed to convert a batch-trained algorithm into an online version. It ignores regularization effects and may be suboptimal under certain conditions.

Although adapting the model-based collaborative filtering methods by online algorithms, e.g., stochastic gradient descent method [57, 61], can involve substantial implementation efforts, the real properties of the algorithms are still not well-investigated. This unexplored territory motivates us to study the online learning algorithms for PMF and RMF thoroughly. To be more specific, we investigate both stochastic gradient descent and dual averaging methods

on both rating-oriented and ranking-oriented MF methods. The remainder of this chapter is organized as follows. Section 3.2 discusses in more detail regarding two models for which we develop online algorithms, namely probabilistic matrix factorization and top-one probability based ranking matrix factorization. We present our online algorithms in Section 3.3. Experimental results and analysis are shown in Section 3.4 and summarize this chapter in Section 3.5.

## 3.2 Model-based Matrix Factorization

In this section, we present a probabilistic matrix factorization model, a top-one probability based ranking matrix factorization model, and their batch-trained algorithms. Before delving into these two models, we first describe the notations used in this chapter. Suppose that we are given a set of $N$ users $\mathcal{U} = \{u_1, u_2, \cdots, u_N\}$ and a set of $M$ items $\mathcal{I} = \{i_1, i_2, \cdots, i_M\}$. Users' rating on the items forms an $N \times M$ matrix $R$, where the element $r_{ui}$ denotes user $u$'s rating on item $i$. Alternatively, we denote all observed ratings in a set of triplets as $(u, i, r) \in \mathcal{Q}$, where $u$ is the user id, $i$ is the item id, and $r$ is the rating given by $u$ to $i$. Usually, the rating $r$ is a value in the range $[R_{\min}, R_{\max}]$. Often, we map it to $[0, 1]$ by $(r - R_{\min})/(R_{\max} - R_{\min})$. To avoid clutter notations, we use $g_{ij}$ to denote $g(U_i^T V_j)$ and $g'_{ij}$ to denote the derivative of the logistic function $g'(U_i^T V_j)$, where $g(x)$ is the logistic function to be defined in Eq. (3.2).

The problem of matrix factorization collaborative filtering is to learn two low-rank feature matrices, $U$ and $V$, to approximate $R$ based on the given $M, N, R$ or $M, N, Q$. The user feature matrix $U$ is a $K \times N$ matrix where each column $U_u$ is the length $K$ feature vector summarizing $u$'s rating behavior. $V$ is a $K \times M$ matrix where each column $V_i$ denotes the length $K$ feature vector inherent in the $i$th item. Generally latent feature size $K$ is much smaller than $N$ and $M$.

For rating-oriented methods, the objective is to find $U$ and $V$ to best fit $R$ so as to correctly predict $\hat{r}_{ui}$, the rating user $u$ would assign to item $i$. Whereas in ranking-oriented methods, the target is to correctly output a ranking $\pi$ of the items in decreasing order of preference for an active user.

### 3.2.1 Probabilistic Matrix Factorization

Probabilistic Matrix Factorization (PMF) adopts a probabilistic linear model with Gaussian observation noise [114]. Maximizing the posterior probability of $p(U, V|R, \sigma^2, \sigma_U^2, \sigma_V^2)$ is equivalent to minimizing a squared loss with regularization defined as:

$$\mathcal{L} = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{M} I_{ij}(r_{ij} - g_{ij})^2 + \frac{\lambda_U}{2}\|U\|_F^2 + \frac{\lambda_V}{2}\|V\|_F^2, \tag{3.1}$$

where $g$ is the logistic function used to map the value into the range of $[0, 1]$ as is in [114]:

$$g(x) = \frac{1}{1 + \exp(-x)}. \tag{3.2}$$

$\lambda_U$ and $\lambda_V$ are $L_2$-regularization strength parameter to avoid overfitting and $I_{ij}$ is an indicator function which equals $1$ if user $i$ have rated item $j$ and $0$ otherwise. Note that Eq. (3.1) implicitly indicates ratings have been mapped to the range of $[0, 1]$.

Gradient descent algorithm can be adopted to reach a local minimum of the objective given in Eq. (3.1). The partial derivative of $\mathcal{L}$ with respect to $U$ and $V$ can be computed as:

$$\frac{\partial \mathcal{L}}{\partial U_i} = \sum_{j=1}^{M} I_{ij}(g_{ij} - r_{ij})g'_{ij}V_j + \lambda_U U_i, \tag{3.3}$$

$$\frac{\partial \mathcal{L}}{\partial V_j} = \sum_{i=1}^{N} I_{ij}(g_{ij} - r_{ij})g'_{ij}U_i + \lambda_V V_j. \tag{3.4}$$

Thus, the feature matrices on users and items can be updated iteratively by

$$U_i \leftarrow U_i - \eta \frac{\partial \mathcal{L}}{\partial U_i}, \qquad V_j \leftarrow V_j - \eta \frac{\partial \mathcal{L}}{\partial V_j}, \tag{3.5}$$

where $\eta$ is the learning rate. In practice, it takes dozens of iterations for PMF to converge. Once trained, the predicted rating that user $u$ would assign to item $i$ can be computed as the expected value of the Gaussian distribution:

$$\hat{r}_{ui} = g_{ui}. \tag{3.6}$$

Note that this value may need to be converted back to the original rating range.

### 3.2.2 Ranking Matrix Factorization

Top-one probability based ranking matrix factorization (RMF) [121] also factorizes the user-item matrix $R$ into two low-rank user and item feature matrices, $U$ and $V$. Different from PMF, it minimizes the cross entropy of two top-one probability distributions defined on actual rating $r_{ij}$ and predicted rating $g_{ij}$.

Top-one probability models the probability of an item being ranked in the top position of an active user's ranking list. It can be defined using either actual rating or predicted rating. Using actual ratings $R$, the top-one probability associated with an item $i$ in a ranking $\pi$ for user $u$ is defined as:

$$p_R(r_{ui}) = \frac{\exp(r_{ui})}{\sum_{k=1}^{M} I_{uk} \exp(r_{uk})}. \tag{3.7}$$

Using predicted rating, the top-one probability of the learned model is defined as:

$$p_{UV}(g_{ui}) = \frac{\exp(g_{ui})}{\sum_{k=1}^{M} I_{uk} \exp(g_{uk})}. \tag{3.8}$$

RMF minimizes the cross entropy between $p_R$ and $p_{UV}$ [121]. Cross entropy of two distributions $p$ and $q$ is defined as:

$$H(p, q) = E_p[-\log q] = -\sum_x p(x) \log q(x). \tag{3.9}$$

This quantity measures the divergence between two distributions and is minimized when $p = q$. Hence, to find the optimal $U$ and $V$, RMF is to minimize the following objective function:

$$\mathcal{L} = \sum_{i=1}^{N} \left\{ -\sum_{j=1}^{M} I_{ij} \frac{\exp(r_{ij})}{\sum_{k=1}^{M} I_{ik} \exp(r_{ik})} \log \left\{ \frac{\exp(g_{ij})}{\sum_{k=1}^{M} I_{ik} \exp(g_{ik})} \right\} \right\} + \frac{\lambda_U}{2} \|U\|_F^2 + \frac{\lambda_V}{2} \|V\|_F^2. \tag{3.10}$$

Similarly, RMF is not a convex optimization problem and the local minima can be sought using the gradient descent method. The gradients of $\mathcal{L}$ with respect to $U$ and $V$ can be calculated by:

$$\frac{\partial \mathcal{L}}{\partial U_i} = \sum_{j=1}^{M} I_{ij} \left\{ \frac{\exp(g_{ij})}{\sum_{k=1}^{M} I_{ik} \exp(g_{ik})} - \frac{\exp(r_{ij})}{\sum_{k=1}^{M} I_{ik} \exp(r_{ik})} \right\} g'_{ij} V_j + \lambda_U U_i, \tag{3.11}$$

$$\frac{\partial \mathcal{L}}{\partial V_j} = \sum_{i=1}^{N} I_{ij} \left\{ \frac{\exp(g_{ij})}{\sum_{k=1}^{M} I_{ik} \exp(g_{ik})} - \frac{\exp(r_{ij})}{\sum_{k=1}^{M} I_{ik} \exp(r_{ik})} \right\} g'_{ij} U_i + \lambda_V V_j. \tag{3.12}$$

Hence, by defining a suitable learning rate $\eta$, we can adopt Eq. (3.5) to update $U$ and $V$ until they converge. Once trained, the model recommends the items in decreasing order of their top-one probability to an active user.

## 3.3 Online Matrix Factorization

It is noted that the gradient descent algorithm can be used to train both PMF and RMF. The batch-training algorithm assumes all the ratings are available before the training. This makes it unsuitable for many practical application scenarios. When the system receives a new rating, the model has to be retrained using *all* available data. To adapt PMF or RMF to such scenarios, it is better to train the model in an *online* manner. Using an online algorithm the model can be easily adapted to fit the new data. In the following, we present our online algorithms for both PMF and RMF to demonstrate such merits [68, 131].

### 3.3.1 Online PMF

We present two algorithms of online PMF, the stochastic gradient descent PMF (SGD-PMF) and dual averaging PMF (DA-PMF).

#### Stochastic Gradient Descent for PMF

By examining the update rules defined in Eq. (3.5), we observe that at each iteration, the low-rank matrices move toward the *average* gradient descent, $\partial\mathcal{L}/\partial U_u$ and $\partial\mathcal{L}/\partial V_i$, by a small step that is controlled by $\eta$. When there is only one rating presented, we would adjust the model solely according to this rating. If the collected ratings are coming sequentially, we could adjust the model stochastically by taking into account that rating only. This corresponds to the scheme of stochastic gradient descent.

Suppose the new coming rating is $(u, i, r) \in \mathcal{Q}$, in (3.1), the terms related to this perticular rating are:

$$\mathcal{L}_{(u,i,r)} = (r_{ui} - g_{ui})^2 + \frac{\lambda_U}{2}\|U_u\|_2^2 + \frac{\lambda_V}{2}\|V_i\|_2^2. \tag{3.13}$$

The first term in Eq. (3.13) is the squared error between the observation and predicted value, and the following two terms are the cor-

responding regularizations. Notice that here the trade-off constants, $\lambda_U$ and $\lambda_V$, are on different scale from those in Eq. (3.1).

Similarly, by adopting the gradient descent method, we obtain the following update equations:

$$U_u \leftarrow U_u - \eta((g_{ui} - r)g'_{ui}V_i + \lambda_U U_u), \tag{3.14}$$

$$V_i \leftarrow V_i - \eta((g_{ui} - r)g'_{ui}U_u + \lambda_V V_i), \tag{3.15}$$

where $\eta$ is the step size controlling how much change to make at each step. This naturally gives an online algorithm, where at each iteration, we make a small change for user $u$'s feature vector $U_u$ and item $i$'s feature vector $V_i$ when a rating $(u, i, r)$ is revealed. We call this method stochastic gradient descent PMF (SGD-PMF) and summarize it in Algorithm 2.

**Algorithm 2** Stochastic Gradient Descent for PMF

Parameter: $N$, $M$, $K$, $\eta$, $\lambda_U$, $\lambda_V$
Input: Observation triplet $(u, i, r) \in \mathcal{Q}$
Initialize $U \in \mathbb{R}^{K \times N}$ and $V \in \mathbb{R}^{K \times M}$ randomly
**for all** $(u, i, r) \in \mathcal{Q}$ **do**
  $U_u \leftarrow U_u - \eta((g_{ui} - r)g'_{ui}V_i + \lambda_U U_u)$
  $V_i \leftarrow V_i - \eta((g_{ui} - r)g'_{ui}U_u + \lambda_V V_i)$
**end for**

Algorithm 2 is *stochastic* in the sense that every time we adjust the parameter, we accommodate it to that particular rating seen at that instance. However, the rating could have some inherent noise that is not easy to eliminate. So every time we modify the parameter, we could have taken a direction that deviates a little from the average gradient as we do in the batch mode algorithm. We expect that taking several stochastic steps approximately have similar net effect as taking average steps. We do not provide the convergence of stochastic gradient descent for PMF here, as its detailed proof can be referred to [119].

**Dual-Averaging Method for PMF**

Recently, dual-averaging method [97] ignites the development of online learning algorithms. By imposing different regularizations, variants of online learning algorithms have been developed and they achieve good result in the corresponding applications [128, 132]. Due to the success and efficiency of dual-averaging method, we decide to adopt it to solve the online PMF problem.

Dual-average method for PMF absorbs previous rating information in an approximate average gradient of the loss. Then it updates the parameters by solving an analytically tractable optimization problem. More importantly, by imposing different regularizations as [128] on each iteration, we can attain different model properties. For example, by applying $L_1$-regularization, we can adjust the regularization strength and yield solutions exhibit various level of sparseness.

Hence, in DA-PMF, we keep track of the average gradient, $Y$, of the square loss with respect to $U$ and $V$. Given a newly observed triplet $(u, i, r)$, we can update the average gradient with respect to $U_u$ by the following rule:

$$Y_{U_u} \leftarrow \frac{t_u - 1}{t_u} Y_{U_u} + \frac{1}{t_u}(g_{ui} - r) g'_{ui} V_i, \tag{3.16}$$

where $t_u$ denotes the number of items $u$ has rated. Note that $Y_{U_u}$ in Eq. (3.16) is an approximation of $\{\sum_{i \in \mathcal{I}_u}(g_{ui} - r) g'_{ui} V_i\}/t_u$ which is the average gradient of the squared loss with respect to $U_u$. $\mathcal{I}_u$ denotes all the items that $u$ has rated. Similarly, we can obtain $Y_{V_i}$, the average gradient of $\mathcal{L}$ with respect to $V_i$, as follows:

$$Y_{V_i} \leftarrow \frac{t_v - 1}{t_v} Y_{V_i} + \frac{1}{t_v}(g_{ui} - r) g'_{ui} U_u, \tag{3.17}$$

where $t_v$ denotes the number of users who have rated item $v$.

Once we get the average gradient, we update $U_u$ and $V_i$ by solving

the following minimization problems:

$$U_u = \text{arg min}_w\{Y_{U_u}^T w + \lambda_U \|w\|_2^2\}, \tag{3.18}$$

$$V_i = \text{arg min}_w\{Y_{V_i}^T w + \lambda_V \|w\|_2^2\}. \tag{3.19}$$

To get an intuition of why choosing such optimization objectives defined in Eq. (3.18) and Eq. (3.19), we should note that $Y_{U_u}^T w$, the dot product of $Y_{U_u}$ and $w$, is minimized when $w$ is on the opposite direction as $Y_{U_u}$. We have shown that $Y_{U_u}$ is the average gradient, whose direction will lead to a larger $\mathcal{L}$. By taking an opposite direction of $Y_{U_u}$, we expect $\mathcal{L}$ to be decreased. However, the minimum value of $Y_{U_u}^T w$ is unbounded if we allow an arbitrary $w$. Adding a regularization term $\lambda_V \|w\|_2^2$ can effectively limit the value that $w$ can assume. Thus using such an optimization objective, we can get $U_u$ and $V_i$ that lead to a decreased $\mathcal{L}$. Convergence of such formulation is referred to [128]. The solution to Eq. (3.18) and Eq. (3.19) can be found analytically by taking the derivative to $0$, which is summarized in Algorithm 3.

DA-RMF commands an explicit regularization effect. Other than the $L_2$-regularization used in Eq. (3.18) and (3.19), we can incorporate other types of regularizations to achieve solutions with different properties. For example, we can adopt an $L_1$-regularization, $\rho\|w\|_1$, and add it to the objective function defined in Eq. (3.18) and (3.19). This can yield sparse solutions for PMF while maintaining a comparable performance. Hence, the corresponding update rules in Eq. (3.20) can be changed to:

$$U_u(k) \leftarrow \begin{cases} 0 & \text{if } |Y_{U_u}(k)| \leq \rho \\ \frac{1}{2\lambda_U}(\rho \text{ sign}(Y_{U_u}(k)) - Y_{U_u}(k)) & \text{otherwise} \end{cases}, \tag{3.21}$$

$$V_k(k) \leftarrow \begin{cases} 0 & \text{if } |Y_{V_k}(k)| \leq \rho \\ \frac{1}{2\lambda_V}(\rho \text{ sign}(Y_{V_i}(k)) - Y_{V_i}(k)) & \text{otherwise} \end{cases}, \tag{3.22}$$

**Algorithm 3** Dual-Averaging Method for PMF (DA-PMF)

Parameters: $N, M, K, \lambda_U, \lambda_V$
Input: Observation triplet $(u, i, r) \in \mathcal{Q}$
Initialize $U \in \mathbb{R}^{K \times N}$ and $V \in \mathbb{R}^{K \times M}$ randomly
Initialize average gradient matrix to $\mathbf{0}$

$$Y_U \in \mathbb{R}^{K \times N} \leftarrow \mathbf{0}; \quad Y_V \in \mathbb{R}^{K \times M} \leftarrow \mathbf{0}$$

Initialize index vector $T_U \in \mathbb{Z}^N \leftarrow \mathbf{0}$ and $T_V \in \mathbb{Z}^M \leftarrow \mathbf{0}$
**for all** $(u, i, r) \in \mathcal{Q}$ **do**
Increase index $T_{U_u} \leftarrow T_{U_u} + 1$ and $T_{V_i} \leftarrow T_{V_i} + 1$
$t_u \leftarrow T_{U_u}, \quad t_v \leftarrow T_{V_i}$
Update average gradient $Y_U$ and $Y_V$

$$Y_{U_u} \leftarrow \frac{t_u - 1}{t_u} Y_{U_u} + \frac{1}{t_u}(g_{ui} - r) g'_{ui} V_i$$
$$Y_{V_i} \leftarrow \frac{t_v - 1}{t_v} Y_{V_i} + \frac{1}{t_v}(g_{ui} - r) g'_{ui} U_u$$

Update latent user and item feature $U_u$ and $V_i$

$$U_u \leftarrow -\frac{1}{2\lambda_U} Y_{U_u}, \quad V_i \leftarrow -\frac{1}{2\lambda_U} Y_{V_i} \tag{3.20}$$

**end for**

where $U_u(k)$ refers to the $k$-th element of vector $U_u$. Detailed derivation of the above equation is referred to [128]. Note that by increasing the parameter $\rho$, we can increase the sparseness of the solutions.

#### Time Complexity and Memory Cost

Both SGD-PMF and DA-PMF are efficient in terms of time complexity and memory cost. For SGD-PMF, only user feature matrix and item feature matrix have to be stored in memory, so the memory cost is $O((N + M)K)$. $K$ is generally on the scale of tens even for a very big dataset. For each observation $(u, i, r) \in \mathcal{Q}$, only $O(K)$ steps are needed to update the model. Since $K$ is quite small, it can be taken as constant time. So SGD-PMF scales linearly with the number of observed ratings. For DA-PMF, beside user feature matrix and item feature matrix, average gradient matrix $Y_U, Y_V$ and index vector $T_U, T_V$ are also stored. Total memory cost is still $O((N + M)K)$. Similar to SGD-PMF, DA-PMF needs $O(K)$ steps for each observation and thus it scales linearly with the number of observed ratings. So both SGD-PMF and DA-PMF can be effective to large-scale datasets.

### 3.3.2 Online RMF

In this section, we present online algorithms for RMF. Similarly, we consider both the stochastic gradient descent method (SGD-RMF) and the dual-averaging method (DA-RMF) for RMF.

#### Stochastic Gradient Descent for RMF

The loss $\mathcal{L}$ of RMF is defined in Eq. (3.10). Let $\mathcal{H} = \mathcal{L} - \frac{\lambda_U}{2}\|U\|_F^2 + \frac{\lambda_V}{2}\|V\|_F^2$ be the loss without the regularization, i.e., it only captures the cross entropy between $p_R$ and $p_{U,V}$.

Unlike the squared loss used in PMF, which can be easily dissected when a new rating is observed, cross entropy is adopted as the

loss in RMF. It measures the divergence between the top-one probability distribution defined using actual rating and that defined using predicted rating. The top-one probability indicates the probability of an item being ranked in the top position. It is essentially a categorical distribution and normalization is engaged to ensure it is a proper probability measure. When a newly observed rating $(u, i, r)$ is revealed, this probability mass function will include one more term indicating the probability of the new item being ranked in the top position. Due to normalization, the top-one probability corresponds to other items are further decayed. $\mathcal{H}$, the cross entropy of these two distributions, also changes accordingly. Note that these changes are coupled together and are thus very difficult to dissect.

We resort to algorithms that approximate the gradient descent of $\mathcal{H}$ with respect to $U_u$ and $V_i$ given in Eq. (3.23) and Eq. (3.24):

$$\frac{\partial \mathcal{H}}{\partial U_i} = \sum_{j=1}^{M} I_{ij} \left\{ \frac{\exp(g_{ij})}{\sum_{k=1}^{M} I_{ik} \exp(g_{ik})} - \frac{\exp(r_{ij})}{\sum_{k=1}^{M} I_{ik} \exp(r_{ik})} \right\} g'_{ij} V_j, \quad (3.23)$$

$$\frac{\partial \mathcal{H}}{\partial V_j} = \sum_{i=1}^{N} I_{ij} \left\{ \frac{\exp(g_{ij})}{\sum_{k=1}^{M} I_{ik} \exp(g_{ik})} - \frac{\exp(r_{ij})}{\sum_{k=1}^{M} I_{ik} \exp(r_{ik})} \right\} g'_{ij} U_i. \quad (3.24)$$

Now, we consider the update of the gradients as the observed rating appears one by one. Denote $Y_{U_u}^{t_u}$ as the gradient of $\mathcal{H}$ with respect to $U_u$ when the $t_u$-th rating is assigned by user $u$ to item $i$. Denote $Y_{V_i}^{t_v}$ as the gradient of $\mathcal{H}$ with respect to $V_i$ when item $i$ receives its $t_v$-th rating. To simplify the expression, we let $\mathcal{I}_u^{t_u}$ denote the set of items rated by user $u$ when the $t_u$-th rating is observed, i.e., $|\mathcal{I}_u^{t_u}| = t_u$. We update the corresponding gradients by the following

rules:

$$Y_{U_u}^{t_u+1} \leftarrow \frac{\sum_{k\in\mathcal{I}_u^{t_u}} \exp(r_{uk})}{\sum_{k\in\mathcal{I}_u^{t_u+1}} \exp(r_{uk})} Y_{U_u}^{t_u} + \left\{ \frac{\exp(g_{ui})}{\sum\limits_{k\in\mathcal{I}_u^{t_u+1}} \exp(g_{uk})} - \frac{\exp(r_{ui})}{\sum\limits_{k\in\mathcal{I}_u^{t_u+1}} \exp(r_{uk})} \right\} g'_{ui} V_i, \tag{3.25}$$

$$Y_{V_i}^{t_v+1} \leftarrow (1-\alpha^{c\times t_v}) Y_{V_i}^{t_v} + \left\{ \frac{\exp(g_{ui})}{\sum\limits_{k\in\mathcal{I}_u^{t_u+1}} \exp(g_{uk})} - \frac{\exp(r_{ui})}{\sum\limits_{k\in\mathcal{I}_u^{t_u+1}} \exp(r_{uk})} \right\} g'_{ui} U_u. \tag{3.26}$$

It should be noted that we initialize $Y_U^0 = \mathbf{0}, Y_V^0 = \mathbf{0}$ for all users and items. Index vector $T_U$, whose $u$-th entry is $t_u$, and $T_V$, whose $i$-th entry is $t_v$, are on a per-user and per-item basis respectively.

Here, we argue that Eq. (3.25) and Eq. (3.26) approximate the ideal gradients in Eq. (3.23) and Eq. (3.24), respectively. It is obvious that Eq. (3.25) recovers Eq. (3.23) provided that $g_{ui}$ approximates $r_{ui}$ well at each iteration. To see that Eq. (3.26) indeed approximates Eq. (3.24), consider what might happen between two ratings item $i$ receives. Note that the summation in Eq. (3.24) is over all the users who have rated item $i$. Since top-one probability is on a per-user basis, we cannot find a single value that properly describes the decay of previously rated item as is the case in Eq. (3.25). Consider the event that might happen between the observations of $(u_1, i, r_1)$ and $(u_2, i, r_2)$. Let the set of users who have rated $i$ be $\mathcal{U}_i$. The change of the top-one probability happens when a $u \in \mathcal{U}_i$ rate another movie $i'$ and thus causing the top-one probability with respect to $i$ to decay due to the normalization. However, as more and more ratings are revealed, the change will be smaller and smaller. So we need to decay previous gradient to reflex this change. Therefore, $\alpha \in (0, 1)$ in Eq. (3.26) controls the initial decay rate and $c$ controls how this rate drops.

**Algorithm 4** Stochastic Gradient Descent for RMF

Parameter: $N$, $M$, $K$, $\eta$, $\lambda_U$, $\lambda_V$, $\alpha$, $c$
Input: Observation triplet $(u, i, r) \in \mathcal{Q}$
Initialize $U \in \mathbb{R}^{K \times N}$ and $V \in \mathbb{R}^{K \times M}$ randomly
Initialize average gradient matrix to $\mathbf{0}$

$$Y_U \in \mathbb{R}^{K \times N} \leftarrow \mathbf{0}; Y_V \in \mathbb{R}^{K \times M} \leftarrow \mathbf{0}$$

Initialize index vector $T_V \in \mathbb{Z}^M \leftarrow \mathbf{0}$
Initialize sum vector $S_R \in \mathbb{R}^N \leftarrow \mathbf{0}$ and $S_{UV} \in \mathbb{R}^N \leftarrow \mathbf{0}$
**for all** $(u, i, r) \in \mathcal{Q}$ **do**
  $t_v \leftarrow T_V(v)$
  $s_r \leftarrow S_R(u)$
  $s_{uv} \leftarrow S_{UV}(u)$
  $s'_r \leftarrow s_r + \exp(r)$
  $s'_{uv} \leftarrow s_{uv} + \exp(g_{ui})$
  $Y_{U_u} \leftarrow \frac{s_r}{s'_r} Y_{U_u} + \{\frac{\exp(g_{ui})}{s'_{uv}} - \frac{\exp(r)}{s'_r}\} g'_{ui} V_i$
  $Y_{V_i} \leftarrow (1 - \alpha^{ct_v}) Y_{V_i} + \{\frac{\exp(g_{ui})}{s'_{uv}} - \frac{\exp(r)}{s'_r}\} g'_{ui} U_u$
  $U_u \leftarrow U_u - \eta(Y_{U_u} + \lambda_U U_u)$
  $V_i \leftarrow V_i - \eta(Y_{V_i} + \lambda_V V_i)$
  $S_R(u) \leftarrow s'_r$
  $S_{UV}(u) \leftarrow s'_{uv}$
  $T_V(v) \leftarrow t_v + 1$
**end for**

Once how to calculate the gradient in an online manner is figured out, adjusting the parameters along the inverse direction of the gradient in every step leads to an algorithm similar to SGD-PMF. We present the complete gradient descent online RMF in Algorithm 4.

#### Dual-Averaging Method for RMF

The gradient we calculated as in Eq. (3.25) and Eq. (3.26) is average gradient. By solving the same optimization objective as defined in Eq. (3.18) and Eq. (3.19) at each iteration, we obtain the Dual-Averaging RMF (DA-RMF) algorithm. This algorithm is summarized in Algorithm 5.

**Algorithm 5** Dual-Averaging method for RMF (DA-RMF)

Parameter: $N$, $M$, $K$, $\lambda_U$, $\lambda_V$, $\alpha$, $c$
Input: Observation triplet $(u, i, r) \in \mathcal{Q}$
Initialize $U \in \mathbb{R}^{K \times N}$ and $V \in \mathbb{R}^{K \times M}$ randomly
Initialize average gradient matrix to $\mathbf{0}$

$$Y_U \in \mathbb{R}^{K \times N} \leftarrow \mathbf{0}; Y_V \in \mathbb{R}^{K \times M} \leftarrow \mathbf{0}$$

Initialize index vector $T_V \in \mathbb{Z}^M \leftarrow \mathbf{0}$
Initialize sum vector $S_R \in \mathbb{R}^N \leftarrow \mathbf{0}$ and $S_{UV} \in \mathbb{R}^N \leftarrow \mathbf{0}$
**for all** $(u, i, r) \in \mathcal{Q}$ **do**
  $t_v \leftarrow T_V(v)$
  $s_r \leftarrow S_R(u)$
  $s_{uv} \leftarrow S_{UV}(u)$
  $s'_r \leftarrow s_r + \exp(r)$
  $s'_{uv} \leftarrow s_{uv} + \exp(g_{ui})$
  $Y_{U_u} \leftarrow \frac{s_r}{s'_r} Y_{U_u} + \{\frac{\exp(g_{ui})}{s'_{uv}} - \frac{\exp(r)}{s'_r}\} g'_{ui} V_i$
  $Y_{V_i} \leftarrow (1 - \alpha^{ct_v}) Y_{V_i} + \{\frac{\exp(g_{ui})}{s'_{uv}} - \frac{\exp(r)}{s'_r}\} g'_{ui} U_u$
  $U_u \leftarrow -\frac{1}{2\lambda_U} Y_{U_u}$
  $V_i \leftarrow -\frac{1}{2\lambda_V} Y_{V_i}$
  $S_R(u) \leftarrow s'_r$
  $S_{UV}(u) \leftarrow s'_{uv}$
  $T_V(v) \leftarrow t_v + 1$
**end for**

**Time Complexity and Memory Cost**

The memory cost for both SGD-RMF and DA-RMF includes the user and item feature matrix $U, V$, approximate average gradient $Y_U, Y_V$, index vector $T_V$, and sum vector $S_R, S_{UV}$. The total memory cost is still $O((N + M)K)$, which is independent of the number of observed ratings. In terms of time complexity, the steps needed for each observed triplet $(u, i, r)$ is still $O(K)$. Namely, both SGD-RMF and DA-RMF scale linearly with the number of observed ratings.

## 3.4 Experiments

In this section, we conduct experiments to compare the performance of our online algorithms with batch-trained algorithms. The questions we want to address include:

1. How is the stochastic gradient descend and dual-averaging algorithms compared with the batch mode algorithms?
2. How do the online algorithms perform under different settings?
3. How well do stochastic gradient descend and dual-averaging methods scale to large datasets?
4. In which way do model parameters, i.e., $\lambda, \eta$, affect the algorithms' performance?
5. For dual-averaging methods, we can attain sparse solution by incorporating $L_1$-regularization. How well does the sparse solution perform?

### 3.4.1 Data Sets

We choose MovieLens[1], Yahoo!Music[2] and Jester [38] to study empirical performance of our algorithms. Table 3.1 shows the basic

[1] http://www.cs.umn.edu/Research/GroupLens
[2] http://kddcup.yahoo.com

Table 3.1: Statistics of datasets

| | **MovieLens** | **Yahoo!Music** | **Jester** |
|---|---|---|---|
| **No. ratings** | 1,000,209 | 252,800,275 | 1,810,455 |
| **No. users** | 6,040 | 1,000,990 | 24,938 |
| **No. items** | 3,952 | 624,961 | 100 |
| **Rating range** | $[1, 5]$ | $[0, 100]$ | $[-10, 10]$ |
| **Rating type** | Integer | Integer | Floating |

statistics of each dataset.

Experiments studying the effect of parameters are performed on MovieLens. Comparisons of online algorithms versus their batch-trained algorithms are also conducted on MovieLens. We select Yahoo!Music to evaluate how the online algorithms scale to large datasets. Yahoo!Music is a very large dataset containing more than 250 million ratings. Yet this data set is extremely sparse, where only 0.4% entries are known. However, RMF experiments do not utilize Yahoo!Music. This is because even though the ratings are on a range from 0 to 100, we found most users apply only a few values to indicate their fondness. The problem with many items sharing the same rating value is that the user's actual preferences over these items are implicit and thus rendering the NDCG metric insensitive to different ranking. Besides MovieLens, we select Jester to evaluate RMF algorithms. Jester consists of floating point ratings up to two decimal places, where users are unlikely to give the same rating for items and the preferences are explicit.

### 3.4.2 Evaluation Metrics

We adopt Root Mean Square Error(RMSE)[3] to evaluate rating-oriented algorithms, i.e. PMF, SGD-PMF and DA-PMF. RMSE evaluates the root of average square error between true rating and predicted rating.

[3]The lower the RMSE, the better the performance.

Denote the test set by $\mathcal{T}$, the definition of RMSE is given below:

$$\text{RMSE} = \sqrt{\frac{\sum_{(u,i,r)\in\mathcal{T}} (\hat{r}_{u,i} - r)^2}{|\mathcal{T}|}}. \tag{3.27}$$

To evaluate the ranking accuracy, we adopt Normalized Discounted Cumulative Gain (NDCG)[4] [51] as the metric. Let $\pi(i)$ be the item ranked on the $i$th position in ranking $\pi$ and the actual rating assigned to $i$ is $r_{\pi(i)}$. Then NDCG at $n$ is defined as:

$$\text{NDCG@n} = \sum_{i=1}^{n} \frac{2^{r_{\hat{\pi}(i)}} - 1}{\log(1+i)} \Big/ \sum_{i=1}^{n} \frac{2^{r_{\pi^*(i)}} - 1}{\log(1+i)}, \tag{3.28}$$

where $\pi^*$ is the optimal ranking.

Here in NDCG, the score that an item gets is $2^r$ divided by $\log(1+i)$, where $i$ is the position in ranking. That is, rating of a higher ranking item will impact NDCG more than that of a lower ranking item. The second sum in Eq. (3.28) is a normalization constant so that the best ranking would have an NDCG value of 1. The range of NDCG is $(0, 1]$ and a higher value means a better ranking. An appealing property of NDCG is that it gives more weight on the items ranked higher than the item ranked lower. This is consistent with our experience that user seldom looks past the first few recommended items.

### 3.4.3 Evaluation Protocol

To better understand the behavior of our online algorithms under different settings, we conduct experiments with the following three settings:

1. T1: Randomly choose 10% of all $(u, i, r)$ triplets for training, and use remaining 90% for evaluation.

2. T5: Randomly choose 50% of all $(u, i, r)$ triplets for training, and use remaining 50% for evaluation.

[4]The higher the NDCG, the better the performance.

3. T9: Randomly choose 90% of all $(u, i, r)$ triplets for training, and use remaining 10% for evaluation.

### 3.4.4 Comparisons

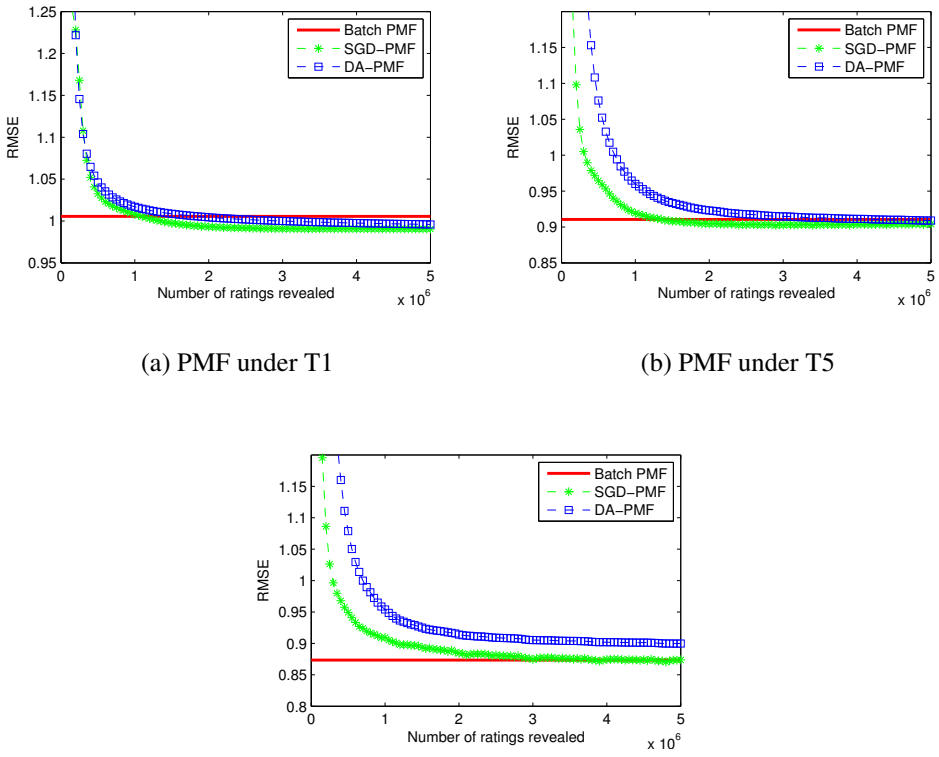
In this section, we compare our online algorithms with full-fledged batch-trained PMF and RMF and investigate how they scale to large datasets. The tuning of the parameters for both the batch-trained algorithms and the online algorithms are performed using a subset of the training set. We reserved 10% of the training set for parameter tuning and grid search for the best performing parameters. Then using these parameters and all the training set as training and obtain the algorithms' performance on the test set. The performances of the batch-trained algorithms are consistent with the results in [1, 121]. The impact of parameters on the performance in online algorithms is referred to Sec. 3.4.5.

#### Online versus Batch

Figure 3.1 shows that DA-PMF and SGD-PMF perform comparable as batch-trained PMF in MovieLens. Under T1, online algorithms even outperform batched-trained algorithms a little bit. This may be due to the fact that under the scenario of few training samples, online learning algorithms are less likely to be trapped in a local optimum. Under T9, DA-PMF's performance is sub-optimal compared with PMF and SGD-PMF. Overall, our online PMF algorithms perform as well as batch-trained algorithm.

Figure 3.2 shows the comparison of various RMF algorithms under different settings. Compared to batch-trained RMF, our online algorithms' performance is off by about 1% in all settings evaluated by NDCG@5. The performance gap is probably due to the approximation when computing the gradient with respect to $V$.

The ordering of data points that are fed to our online algorithms are chosen randomly. We have repeated the experiments several




(a) PMF under T1

(b) PMF under T5

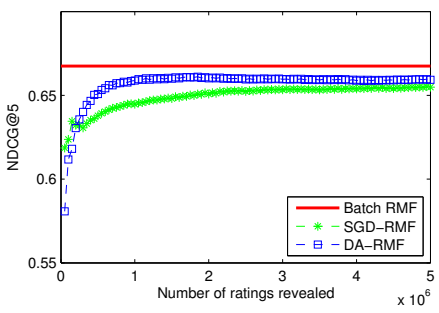
(c) PMF under T9

Figure 3.1: Comparison of PMF algorithms in MovieLens

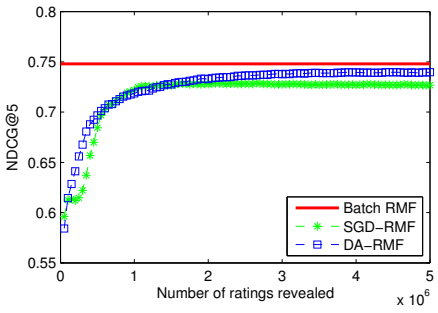
times and the ordering does not impact the overall performance. It is possible that in extreme cases, for example, all the ratings for user $u$ are revealed in the beginning of the training process, the prediction accuracy might be impacted. However, we argue that in real recommender systems, random ordering provides a good approximation of the true incoming sequence and thus the problem of order is negligible.

### Scaling to Large Dataset

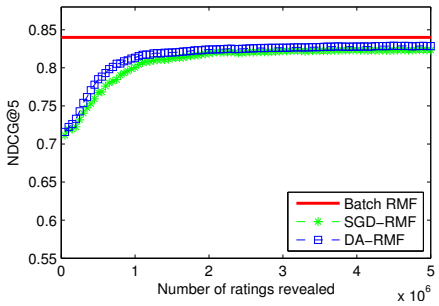
Figure 3.3 shows the comparison of online and batch PMF in Yahoo!Music. Note that we evaluate the performance on the evaluation set, which contains 4 million ratings. Due to the huge size




(a) RMF under T1




(b) RMF under T5

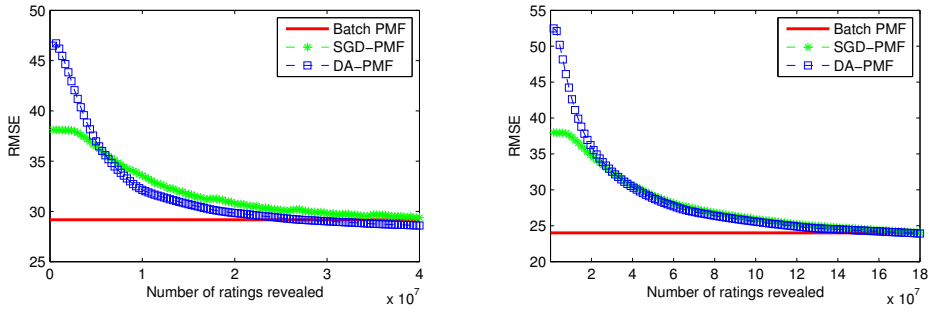



(c) RMF under T9

Figure 3.2: Comparison of RMF algorithms in MovieLens

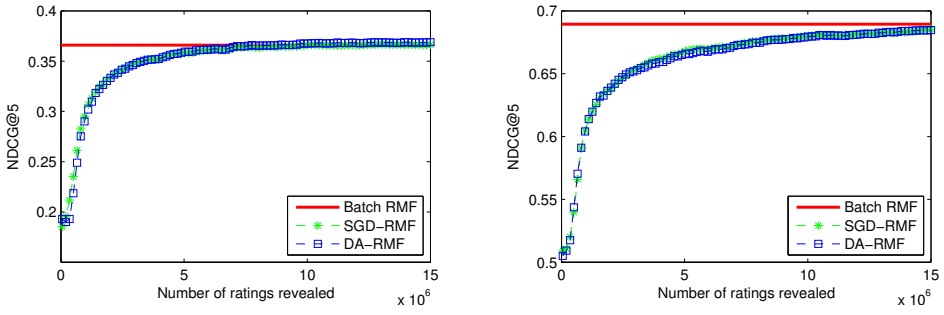
of Yahoo!Music dataset, we were unable to perform batch-trained PMF using T9 setting. Under T5, batch-trained PMF takes more than 8 hours to finish 120 iterations (to converge) using a C++ implementation in a Linux workstation with Xeon Dual Core 2.4GHz processor and 32GB memories. The online algorithms take only about 10 minutes to finish processing all 180 million ratings to reach a similar performance. The time saving is phenomenal. The memory saving is even more dramatic. By loading the data on-demand, our online algorithm takes only several hundred MB memories. The batch-trained PMF can take up a dozen GB memories under T5. As shown, the performance of our online algorithms remains comparable to batch-trained PMF.

Yahoo!Music dataset gives us a peek at the size of the real commercial recommender system in use today. The efficiency of batch-trained algorithms may not be an issue for relatively small size dataset. But for real commercial recommender systems like Yahoo!Music, the efficiency in terms of both time and memory cost become a challenging issue. As being demonstrated, our online algorithms, which scale linearly with the number of ratings, converge much faster and consume much less memory than their batch-trained alternatives, yet maintain a comparable performance.




(a) PMF under T1 (b) PMF under T5

Figure 3.3: Comparison of PMF in Yahoo!Music




(a) RMF under T5 (b) RMF under T9

Figure 3.4: Comparison of RMF in Jester

We choose Jester to verify the online RMF algorithms because

the preferences of users are explicit in this dataset. Figure 3.4 shows the comparison between online RMF and batch-trained RMF in Jester. We observe that both DA-RMF and SGD-RMF deliver comparable performance as batch-trained RMF. The NDCG value obtained in Jester is smaller than that in MovieLens because its ratings are on a wider range.

The dataset is extremely asymmetric in the sense that there are far more users than items in this dataset. We have to scale two separate learning rate $\eta_U$ and $\eta_V$ properly in batch-trained RMF for it to converge. In batch-trained algorithm, using the same learning rate for $U_u$ and $V_i$ would result in too big a step for $V_i$ and too small a step for $U_u$ in batch-trained RMF. So we use $\eta/t_u$ and $\eta/t_v$ as the true learning rate when update the parameter for $U_u$ and $V_i$ respectively, where $t_u$ is the number of items rated by $u$ and $t_v$ is the number of ratings received by $V_i$. Without scaling the learning rate properly, batch-trained RMF would not even converge. For SGD-RMF and DA-RMF, no modification is needed because online algorithms update parameters on a *per rating* basis. The result shows that our online algorithm accommodates to asymmetric dataset easily.

This asymmetric dataset exposes another issue associated with batch-trained algorithms. Using the same learning rate would not be suitable for extremely unbalanced dataset and using separate learning rate for $U$ and $V$ would require one more parameter to be tuned. Online algorithms solve this issue naturally. In real commercial recommender system, we suspect that it is often the case that we have far more users than items in the system. In Fig. 3.4(a), we reveal $50,000$ ratings to online algorithms every 3 iterations and we reveal $50,000$ ratings every 2 iterations in Fig. 3.4(b).

Please note that, in the experiment, we have cycled through the ratings and fed them randomly into the algorithms. The results are reported in Table 3.2 and 3.3.

Table 3.2: Online and batch PMF results(RMSE)

| | MovieLens | | | Yahoo!Music | |
|---|---|---|---|---|---|
| | **T1** | **T5** | **T9** | **T1** | **T5** |
| **PMF** | 1.005 | 0.910 | 0.873 | 29.16 | 24.00 |
| **DA-PMF** | 0.996 | 0.909 | 0.900 | 28.59 | 23.92 |
| **SGD-PMF** | 0.991 | 0.904 | 0.874 | 29.38 | 24.02 |

Table 3.3: Online and batch RMF results(NDCG@5)

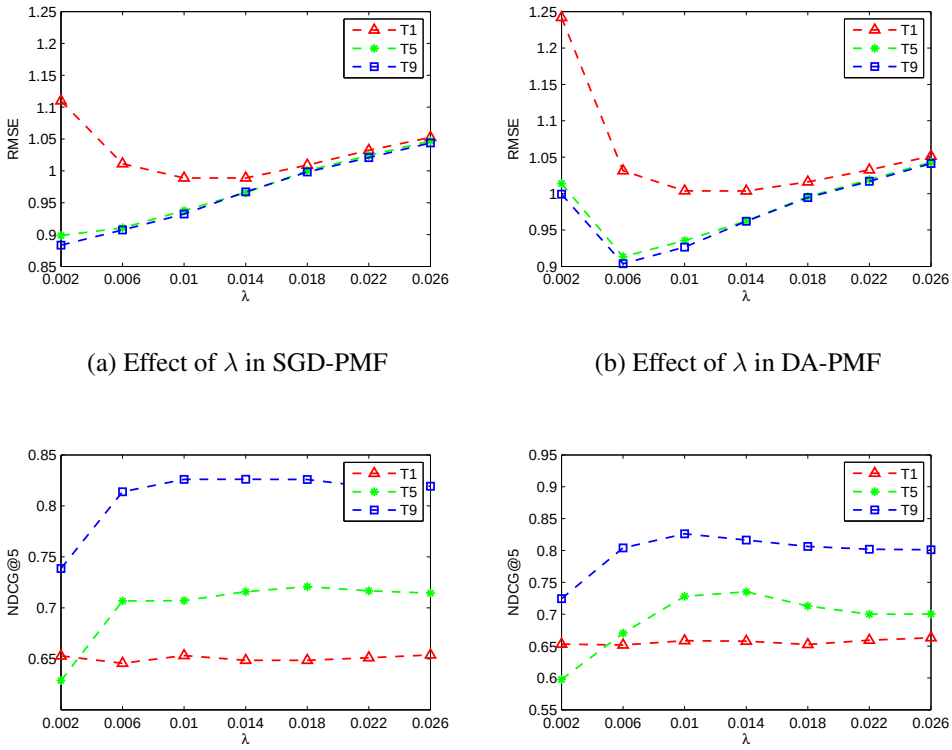
| | MovieLens | | | Jester | |
|---|---|---|---|---|---|
| | **T1** | **T5** | **T9** | **T5** | **T9** |
| **RMF** | 0.667 | 0.748 | 0.840 | 0.367 | 0.687 |
| **DA-RMF** | 0.659 | 0.740 | 0.827 | 0.369 | 0.685 |
| **SGD-RMF** | 0.655 | 0.727 | 0.824 | 0.366 | 0.685 |

### 3.4.5 Impact of Parameters

We analyze the impact of parameters in this section. We employ latent feature dimension $K = 10$ consistently for all algorithms, which is a suitable value according to our empirical results. Since all algorithms we consider are matrix factorization based, employing the same latent feature size is fair for comparison. The results reported are obtained on MovieLens.

#### Impact of $\lambda$

The parameter $\lambda$ controls the trade-off between the regularization and the model loss. We set $\lambda_U = \lambda_V = \lambda$ for simplicity [114, 121]. Figure 3.5 shows the impact of $\lambda$ in four online algorithms under different settings. The range of $\lambda$ is selected by trail-and-error as shown in Figure 3.5, where the range of $\lambda$ gives reasonable performance. As we can see, there is a clear trend that the more data we have, the smaller $\lambda$ we need. Since the model complexity is fixed (i.e., we choose dimension = 10 for all experiments), we need a larger $\lambda$ to




(a) Effect of $\lambda$ in SGD-PMF

(b) Effect of $\lambda$ in DA-PMF

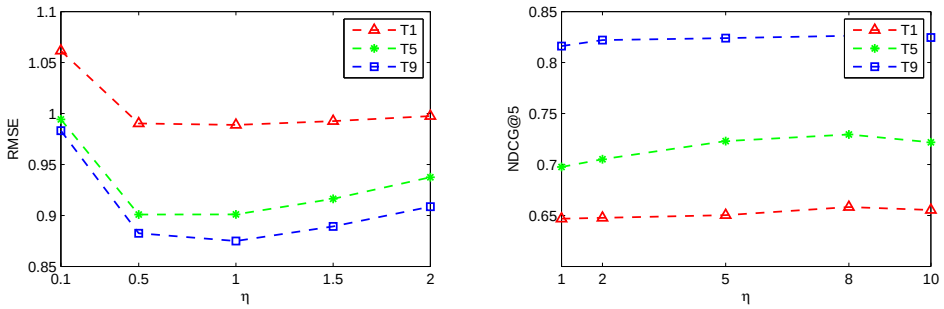
(c) Effect of $\lambda$ in SGD-RMF

(d) Effect of $\lambda$ in DA-RMF

Figure 3.5: Effect of $\lambda$ in various online algorithm

avoid over-fitting when there is only limited data. In [1], the author reported that stochastic gradient descent without regularization performs quite well compared to batch-trained algorithm. This is true only when we have access to abundant data. A proper regularization is vital for a model with limited data.

Figure 3.5(c) and Figure 3.5(d) show that both ranking-oriented online algorithms are quite stale in the setting T1. This is due to the insufficiency of training data in T1. Only 10% of training data to train the model would not make it well-fitted. This causes the insensitivity of the effect of $\lambda$.




(a) Effect of $\eta$ in SGD-PMF

(b) Effect of $\eta$ in SGD-RMF

Figure 3.6: Effect of $\eta$ in various online algorithm

**Impact of $\eta$**

The parameter $\eta$ denotes the learning rate, which is only applied in SGD-PMF and SGD-RMF. It will impact the convergence rate and the performance of the algorithms. Figure 3.6 shows the performance of SGD-PMF and SGD-RMF under T1, T5 and T9 with respect to various $\eta$ values. When performing the experiments, we employ the best $\lambda$ we learnt in previous sections. We see that $\eta = 1.0$ is optimal for SGD-PMF and $\eta = 8.0$ is optimal for SGD-RMF.

There is a subtle difference between online algorithms and batch-trained algorithms. The optimal learning rate $\eta$ depends on the size of the training set in batch-trained algorithms. On the contrary, it is independent for online algorithms. The reason is that we update $U$ and $V$ with respect to only *one* rating in online algorithms each time.

**Impact of other parameters**

In the two online RMF algorithms, there are two extra parameters $\alpha$ and $c$ that control the decay and drop-rate of decay respectively. In practice, they do not affect the model performance as significant as $\lambda$. Hence, we do not present their sensitivity analysis here. In

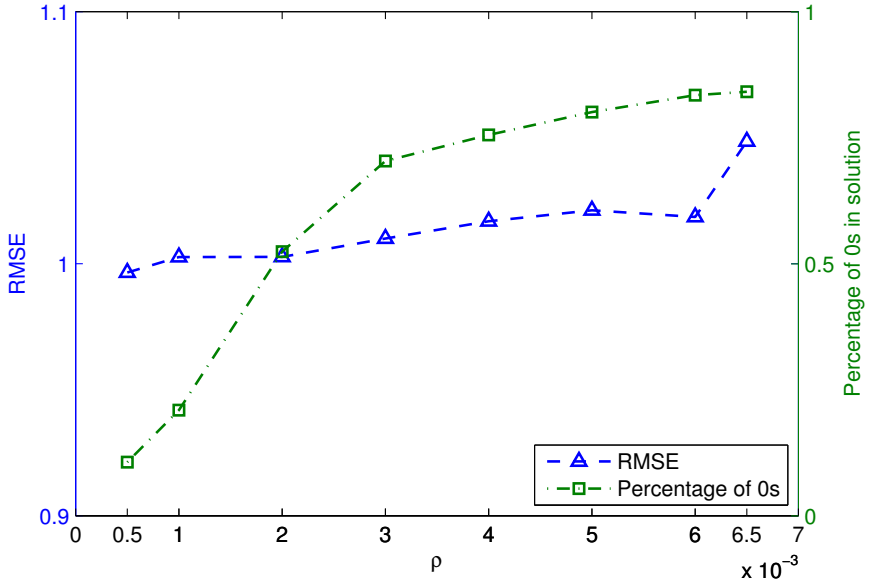



Figure 3.7: Effect of $\rho$ on sparseness and accuracy

the experiment, we set $\alpha = 0.8, c = 0.2$ since they deliver good performance empirically.

### 3.4.6 Sparse Solution

In Dual-Averaging algorithms, adding an $L_1$-regularization to the optimization objective can lead to a sparse solution. We demonstrate this in DA-PMF with Eq. (3.21) and Eq. (3.22). Figure 3.7 demonstrates the effect of $\rho$, the parameter controlling the trade-off between $L_1$-regularization and model loss, on accuracy (RMSE) and percentage of zeros in solution under T1 using MovieLens dataset. As $\rho$ increases, percentage of zeros in solution increases and the accuracy of the model decreases marginally. Using $\rho = 0.003$, we can obtain a solution that achieves 70.4% of zero entry while still maintaining an RMSE of 1.01.

It is important to emphasize again that a sparse solution can maintain the model performance with less information. It can reduce the memory cost in storing the model and the time cost in producing the

results. Sparse solution is also a good approach to control the overall complexity of the model and reduce over-fitting. In a real world recommender system, we could increase the dimensionality of the low-rank matrix to account for the large amount of users and items. Sparse solution is especially useful in such situations to avoid over-fitting and control the model complexity. The constraint that most of the entries in the user and item feature matrices should be zero effectively limits the model complexity while allow different groups of users and items to have distinct features. Such effects are very hard to achieve in a non-sparse solution.

## 3.5 Summary

In this chapter, we have thoroughly investigated the online learning algorithms for rating-oriented CF model, PMF, and ranking-oriented CF model, RMF. More specifically, we developed Stochastic Gradient Descent and Dual-Averaging methods for both models. Our proposed algorithms scale linearly with the number of observed ratings. Furthermore, they obviate the need to hold all data in memory and thus can be applied to large-scale applications. Using Dual-Averaging method with $L_1$-regularization, we can achieve sparse solution while maintaining comparable performance. Experimental results show that our online algorithms achieve comparable performance as their batch-trained algorithms while dramatically boosting efficiency.

---

□ **End of chapter.**

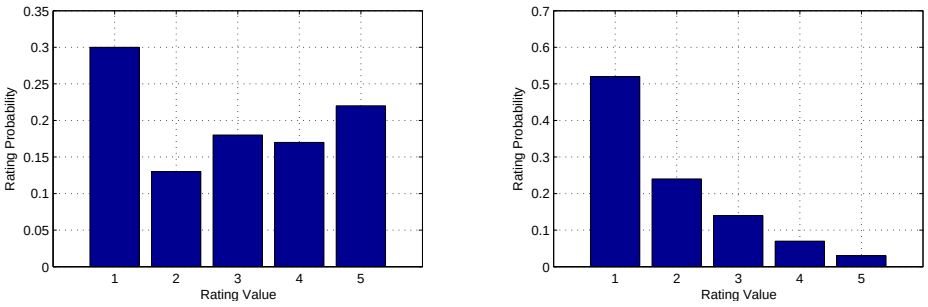
# Chapter 4

# Response Aware Collaborative Filtering

## 4.1 Problem and Motivation

In real-world online rating systems, users' ratings carry twofold information. Firstly the rating value indicates a user's preference on a particular item as well as an item's inherent features. The scores that a user assigns to different items convey information on what the user likes and what the user dislikes. The rating values that an item received from different users also carry information on intrinsic properties of the item. Second, the ratings also reveal users' response patterns, i.e., some items are rated while others not. This information can be utilized to improve the model performance. However, previously proposed methods usually assume that all the users would rate *all* the inspected items, or more generally, *randomly* select inspected items to rate. These methods fit the users' ratings directly and ignore the key factor, users' response patterns. The ignorance will degrade the model performance. In this chapter, we explore previously ignored response information to further boost recommender system's quality.

Practically, the assumption of *all inspection* or *randomly rate* is not true in real-world rating systems. Users are unlikely to rate all the inspected items or randomly select the inspected items to rate.




(a) User selected ratings

(b) Randomly selected ratings

Figure 4.1: Distribution of ratings in a music website [87].

Shown in Figure 4.1(a) is the rating value distribution of the items that users *choose to rate*, while Figure 4.1(b) shows the distribution of ratings for *randomly* selected songs from the same group of users. Clearly these two distributions are very different. In the user selected ratings, there are far more items with high ratings than those in the randomly selected songs. This is compelling evidence showing that the assumption that all the users would rate all the inspected items or select random items to rate is unlikely to be true. The investigation of the Yahoo!LaunchCast data indicates that users are more likely to rate items they do love and hate, but not neutral [87, 122].

To further demonstrate the risk of *incorrect* parameter estimation and *biased* rating prediction when ignoring the response in-

Table 4.1: Skewed ratings on 5 items from 5 users

| | item1 | item2 | item3 | item4 | item5 |
|---|---|---|---|---|---|
| user1 | 5 | 4 | | | |
| user2 | | 5 | | 4 | |
| user3 | 4 | | | 4 | |
| user4 | 5 | | 5 | | |
| user5 | | 4 | | | 5 |

formation, we show an intuitive example of five users' rating on five items in Table 4.1, where the ratings are skewed to either 4 or 5. Clearly, user-based approaches [14, 43] and item-based approaches [30, 65, 117] are more likely to predict rating values in the range of 4 to 5. Similarly, ignoring response information will cause the one-class issue for model-based approaches [64, 101, 102]. In a real-world recommender system, the case may not be as extreme as is in Table 4.1. Nevertheless, the effect is similar. By ignoring the response information, we will learn a model that has *bias*.

Currently, there are two main streams of work trying to solve the above response ignorance problem. One line of work tries to model the above phenomena as a one-class collaborative filtering task [64, 101, 102]. A heuristic weight in the range of 0 to 1 is introduced to calibrate the loss on those unseen ratings, where the rating scores are set to zeros [101, 102]. Embedding user information is also adopted to optimize the weight on the unseen ratings via users' similarity [64]. However, these methods do not model the users' missing response information together with the ratings. The other line of work model the response ignorance through missing data theory [70]. The multinomial mixture model is adopted to model the non-random response [87]. The work is also extended for collaborative ranking [86]. These methods model users' response patterns and ratings via multinomial mixture model, but they discard the effectiveness and interpretability of the matrix factorization approaches [56, 114].

To bridge this gap, we are the first to integrate the users' response patterns into PMF to establish a unified framework, which we refer to as Response Aware PMF (RAPMF) [69]. The response models we propose include the rating dominating response model, and a generalized one, the context-aware response model. We demonstrate the advantages of our proposed RAPMF through detailed and fair experimental comparison.

The rest of this chapter is organized as follows. In Section 4.2,

we motivate the explicit modeling of user responses from a probabilistic point of view. In Section 4.3, we present how to incorporate response models into PMF and elaborate the proposed RAPMF model. Empirical study and comparison with previous work is conducted in Section 4.4. We summarize this chapter in Section 4.5.

## 4.2 Response and Missing Theory

Modeling response patterns have a strong incentive from statistical missing data theory [70]. The response patterns can be hidden [24, 48] or explicit. In recommender system case, it is explicit. In the following, we show that without modeling the response patterns properly, we may learn a *bias* model.

### 4.2.1 Setup and Notation

Assume that we are given a partially observed $N \times M$ matrix $X$, where $N$ is the number of users and $M$ is the number of items, the $(i, j)$ element of $X$ denotes the rating assigned by user $i$ to item $j$ in the scale of 1 to $D$. Collaborative filtering approaches try to recover the original full matrix $X_{full}$ to predict users' preferences.

In the matrix $X$, $0$ denotes an unobserved entry. Alternatively, we denote all the observations as a set of triplets $(i, j, x) \in \mathcal{Q}$. Moreover, we define a companion response indicator matrix $R$ to denote whether the corresponding rating is observed in $X$. If $X_{ij} \neq 0$, i.e., we have observed user $i$'s rating on item $j$, then $R_{ij} = 1$. Otherwise $R_{ij} = 0$. Note that $X$ is partially observed while $R$ is fully observed.

### 4.2.2 Missing Data Theory

Following missing data theory in [70], we model the collaborative filtering data as a two-step procedure. First, a data model $P(X|\theta)$ generates the full data matrix $X_{full}$. Then, a response model $P(R|X, \mu)$

determines which elements in $X_{full}$ are observed. Hence, we can take a parametric joint distribution on the observed data matrix $X$ and the response matrix $R$, conditioned on the model parameters, $\theta$ and $\mu$.

$$P(R, X|\mu, \theta) = P(R|X, \mu, \theta)P(X|\mu, \theta) \tag{4.1}$$
$$= P(R|X, \mu)P(X|\theta), \tag{4.2}$$

where $P(R|X, \mu)$ is also referred to as the missing data model. In the following, we use response model and missing data model interchangeably.

According to the missing data theory [70], there are three kinds of missing data assumptions: 1) Missing Completely At Random (MCAR); 2) Missing At Random (MAR), and 3) Not Missing At Random (NMAR). MCAR has the strongest independence assumption. Under the MCAR assumption, the missing mechanism cannot depend on the data in any way. Whether we will observe a response is fully determined by the parameter $\mu$ and is irrelevant to the users' rating, i.e.,

$$P(R|X, \mu) = P(R|\mu) \tag{4.3}$$

One typical example where MCAR holds is that given an inspected item, whether it will be observed is a Bernoulli trail with probability $\mu$.

The MAR assumption is slightly different from the MCAR assumption. Let $X_{full} = (X_{obs}, X_{mis})$, i.e., the full data matrix $X_{full}$ is separated into observed data matrix $X_{obs}$ and missing data matrix $X_{mis}$. Under the MAR assumption, the response probability depends on the *observed* data and $\mu$, i.e.,

$$P(R|X, \mu) = P(R|X_{obs}, \mu). \tag{4.4}$$

Marlin and Zemel [86] refer to this as the probability of observing a particular response only depending on the observed elements of the data vector. The assumption made by MAR may seem bizarre. However, it comes up naturally if we want to ignore response model

and still learn *unbiased* data model parameters. We demonstrate this in the following.

Let $\mathcal{L}(\mu, \theta | X_{obs}, R)$ be the likelihood of $\mu$ and $\theta$ given the observation $X_{obs}$ and $R$. Under the MAR assumption, we have

$$\begin{aligned}
& \mathcal{L}(\mu, \theta | X_{obs}, R) = P(R, X_{obs} | \mu, \theta) \\
= & \int_{X_{mis}} P(R, X | \mu, \theta) dX_{mis} \\
= & \int_{X_{mis}} P(R | X, \mu) P(X | \theta) dX_{mis} \\
= & \int_{X_{mis}} P(R | X_{obs}, \mu) P(X | \theta) dX_{mis} \\
= & P(R | X_{obs}, \mu) \int_{X_{mis}} P(X | \theta) dX_{mis} \\
= & P(R | X_{obs}, \mu) P(X_{obs} | \theta) \\
\propto & P(X_{obs} | \theta).
\end{aligned} \tag{4.5}$$

The key to marginalize the missing data is that the missing data model depends only on the observed data, i.e., MAR assumption in Eq. (4.4). Under the MCAR assumption, we can simplify Eq. (4.5) similarly, which only depends on $\mu$. Note that the assumption made by MAR appears naturally in the derivation. This is the independence assumption we cannot release anymore without losing the ability to marginalize the complete data model independently of missing data model.

If both MCAR and MAR fail to hold, then NMAR assumption is made. Unlike MCAR and MAR, NMAR requires an explicit response model in order to learn unbiased model parameters. Otherwise, maximizing $P(X_{obs} | \theta)$ directly can yield a *biased* $\theta$. With only a few exceptions [86, 87], nearly all the previous work on recommender systems try to maximize the data model directly [46, 82, 104, 114, 115]. In a typical recommender system, the data collected can easily violate the MAR assumption. The distinct distribution of rating values on user selected items and randomly selected items

hints that the response pattern depends on not only the observed data. Also, a survey on the Yahoo! LanuchCast provides evidence that the response probability might depend on the fondness of particular items [87, 122].

## 4.3 Models and Analysis

In the following, we first review the Probabilistic Matrix Factorization (PMF). After that, we present the response aware PMF and show how it can incorporate PMF with the response models. More specifically, we introduce two response models, the rating dominant response model and the context-aware response model. The updating rules and complexity analysis are provided correspondingly.

### 4.3.1 Probabilistic Matrix Factorization

PMF [114] is one of the most famous matrix factorization models in collaborative filtering, which decomposes the partially observed data matrix $X$ into the product of two low-rank latent feature matrices, $U$ and $V$, where $U \in \mathbb{R}^{K\times N}$, $V \in \mathbb{R}^{K\times M}$, and $K \ll \min(N, M)$.

By assuming Gaussian distribution on the residual noise of observed data and placing Gaussian priors on the latent feature matrices, PMF tries to maximize the log-likelihood of the posterior distribution on the user and item features as follows:

$$\mathcal{L}_{PMF} = -\sum_{(i,j,x)\in\mathcal{Q}} \frac{(x - U_i^T V_j)^2}{2\sigma^2} - \frac{\|U\|_F^2}{2\sigma_U^2} - \frac{\|V\|_F^2}{2\sigma_V^2}. \tag{4.6}$$

This is equivalent to minimizing a squared loss with regularization defined as follows:

$$\mathcal{E} = \frac{1}{2}\sum_{(i,j,x)\in\mathcal{Q}} (x - U_i^T V_j)^2 + \frac{\lambda_U}{2}\|U\|_F^2 + \frac{\lambda_V}{2}\|V\|_F^2, \tag{4.7}$$

where $\lambda_U = \sigma^2/\sigma_U^2$ and $\lambda_V = \sigma^2/\sigma_V^2$ are positive constants to control the trade-off between the loss and the regularization terms. $\|\cdot\|_F^2$ denotes the Frobenious norm.

After training the PMF model via gradient descent or stochastic gradient algorithms [114], the predicted rating that user $i$ would assign to item $j$ can be computed as the expected mean of the Gaussian distribution $\hat{x}_{ij} = U_i^T V_j$.

### 4.3.2 Response Aware PMF

In Sec. 4.2, we have demonstrated that by neglecting response patterns, not only do we lose the potential information that might boost the model performance, but also can it lead to incorrect or biased parameters estimation. Due to the effectiveness and interpretability of PMF, we will unify it with explicit response models, which we refer to as Response Aware PMF (RAPMF).

Replacing $\theta$ in Eq. (4.2) by the low-rank latent feature matrices in PMF, we have

$$P(R, X|U, V, \mu, \sigma^2) = P(R|X, U, V, \mu, \sigma^2)P(X|U, V, \sigma^2). \quad (4.8)$$

The probability of full model, $P(R, X|U, V, \mu, \sigma^2)$, is decomposed into data model $P(X|U, V, \sigma^2)$ and the missing data model $P(R|X, U, V, \mu, \sigma^2)$.

### 4.3.3 Response Model

Modeling the missing data successfully requires a correct and tractable distribution on the response patterns. Bernoulli distribution is an intuitive distribution to explain data missing phenomena [87]. Depending on whether users' and items' features are incorporated, we propose two response models, *rating dominant response model* and *context-aware response model*.

### 4.3.4 Rating Dominant Response Model

For the sake of simplification, we assume the probability that a user chooses to rate an item follows a Bernoulli distribution given the rating assigned is $k$. Hence, for a scale of 1 to $D$, the rating dominant response model has $D$ parameters $\mu_1, \mu_2, \cdots, \mu_D$.

If $X$ is fully observed, then the response mechanism can be modeled as [87]:

$$\begin{aligned} &P(R|X, U, V, \mu, \sigma^2) = P(R|X, \mu) \\ &= \prod_{i=1}^{N}\prod_{j=1}^{M}\prod_{k=1}^{D}(\mu_k^{[r_{ij}=1]}(1-\mu_k)^{[r_{ij}=0]})^{[x_{ij}=k]}, \end{aligned} \tag{4.9}$$

where $[r = 0]$ is an indicator variable that outputs $1$ if the expression is valid and $0$ otherwise. It is noted that Eq. (4.9) adopts the "winner-take-all" scheme, i.e., a hard assignment scheme, to model users' response on a particular rating.

However, in real-world recommender system, the data is not fully observed. The "winner-take-all" scheme brings the risk of deteriorating assignment probability when the data is recovered based on the learned model. Hence, we adopt a soft assignment using probability of the possible rating values in the response model as follows:

$$\begin{aligned} &P(R|X, U, V, \mu, \sigma^2) = P(R|U, V, \mu, \sigma^2) \\ &= \prod_{i=1}^{N}\prod_{j=1}^{M}\sum_{k=1}^{D}(\mu_k^{[r_{ij}=1]}(1-\mu_k)^{[r_{ij}=0]})P(x_{ij}=k|U, V, \sigma^2), \end{aligned} \tag{4.10}$$

where $P(X|U, V, \sigma^2)$, the probability of $X$ being assigned to $k$, can be set to $\mathcal{N}(k|U^T V, \sigma^2)$ as is in [114].

To relieve the inaccuracy issue when recovering the original model, we further introduce a discount parameter $\beta$ on the assignment prob-

ability

$$P(R|X,U,V,\mu,\sigma^2) = P(R|U,V,\mu,\sigma^2) \tag{4.11}$$

$$\propto \prod_{i=1}^{N}\prod_{j=1}^{M}(\sum_{k=1}^{D}(\mu_k^{[r_{ij}=1]}(1-\mu_k)^{[r_{ij}=0]})\mathcal{N}(k|U^TV,\sigma^2))^{\beta}, \tag{4.12}$$

where the parameter $\beta$, in the range of 0 to 1, can be interpreted as the faith we have on the response model relative to the data model. As $\beta$ decreases, the effect of the response model decreases correspondingly. When $\beta = 0$, the RAPMF collapses to PMF.

More importantly, the expectations of Bernoulli distributions, $\mu_k$'s should be in the range of 0 to 1. With the performance consideration, the logistic function is usually adopted to constrain the range of $\mu_k$'s [86],

$$g(\mu_k) = \frac{1}{1+\exp(-\mu_k)}, \quad k = 1,\ldots,D. \tag{4.13}$$

Similarly, we place a zero mean Gaussian prior on $\mu_k$ to regularize it.

Note that in Eq. (4.10), we use only one parameter for each possible rating value, so all the users and items share the same probability as long as the rating values are the same. This is a simple approach to capture the intuition that the rating assigned to an item may influence the chance that it got rated. This motivates us to name this model as *rating dominant response model*. We refer to PMF with rating dominate response model as RAPMF-r.

By incorporating the response model in Eq. (4.12) and the PFM model in Eq. (4.6) into RAPMF in Eq. (4.8), we obtain the log-

likelihood of the RAPMF-r as follows:

$$\begin{aligned}
&\mathcal{L}(U, V, \sigma^2, \mu) \\
&= \beta \sum_{i=1}^{N} \sum_{j=1}^{M} \log(\sum_{k=1}^{D} \alpha_{kij} \mathcal{N}(k|U^T V, \sigma^2)) - \frac{1}{2\sigma_\mu^2} \|\mu\|^2 - \\
&\sum_{(i,j,x) \in Q} \frac{(x_{ij} - U_i^T V_j)^2}{2\sigma^2} - \frac{1}{2\sigma_U^2} \|U\|_F^2 - \frac{1}{2\sigma_V^2} \|V\|_F^2 + C, \qquad (4.14)
\end{aligned}$$

where $C$ denotes the constant terms and $\alpha_{kij}$ is defined as

$$\alpha_{kij} = (g(\mu_k)^{[r_{ij}=1]} (1 - g(\mu_k))^{[r_{ij}=0]}). \qquad (4.15)$$

The gradient of $\mathcal{L}$ with respect to $U_i$ is:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial U_i} =& - \beta \sum_{j=1}^{M} \frac{\sum_{k=1}^{D} \alpha_{kij} \mathcal{N}(k|U^T V, \sigma^2)(U_i^T V_j - k) V_j}{\sum_{k=1}^{D} \alpha_{kij} \mathcal{N}(k|U^T V, \sigma^2)} \\
&- \sum_{j=1}^{M} (U_i^T V_j - x_{ij})[r_{ij} = 1] V_j - \lambda_U U_i. \qquad (4.16)
\end{aligned}$$

Similarly, the gradient of $\mathcal{L}$ with respect to $V_j$ is:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial V_j} =& - \beta \sum_{i=1}^{N} \frac{\sum_{k=1}^{D} \alpha_{kij} \mathcal{N}(k|U^T V, \sigma^2)(U_i^T V_j - k) U_i}{\sum_{k=1}^{D} \alpha_{kij} \mathcal{N}(k|U^T V, \sigma^2)} \\
&- \sum_{j=1}^{M} (U_i^T V_j - x_{ij})[r_{ij} = 1] U_i - \lambda_V V_j. \qquad (4.17)
\end{aligned}$$

Both Eq. (4.16) and Eq. (4.17) consist of three terms. The first term corresponds to the change due to the response model, the second term is the change due to the data model and third is a regularization to avoid overfitting. Note that by adjusting $\beta$, we effectively alter the weight of the response model when updating parameters.

Finally, the gradient of $\mathcal{L}$ with respect to $\mu_l$ is

$$\frac{\partial \mathcal{L}}{\partial \mu_l} = \sum_{i=1}^{N} \sum_{j=1}^{M} \frac{\mathcal{N}(l|U^T V, \sigma^2) g'(\mu_l) (-1)^{[r_{ij}=0]}}{\sum_{k=1}^{D} \alpha_{kij} \mathcal{N}(k|U^T V, \sigma^2)} - \lambda_\mu \mu_l, \qquad (4.18)$$

where $g'(x)$ is the derivative of the sigmoid function $g(x)$. In Eq. (4.16), (4.17), (4.18), $\lambda_U = \sigma^2/\sigma_U^2, \lambda_V = \sigma^2/\sigma_V^2$ and $\lambda_\mu = \sigma^2/\sigma_\mu^2$ and a multiplicative constant $1/\sigma^2$ is dropped in all three equations.

To learn model parameters, we alternatively update $U, V$ and $\mu$ using the gradient algorithm with a learning rate $\eta$ by maximizing the log-likelihood. First we update $U, V$ by

$$U_i \leftarrow U_i + \eta \frac{\partial \mathcal{L}}{\partial U_i}, \qquad V_j \leftarrow V_j + \eta \frac{\partial \mathcal{L}}{\partial V_j}. \tag{4.19}$$

Then using the updated $U, V$, we update $\mu_l$ by

$$\mu_l \leftarrow \mu_l + \eta \frac{\partial \mathcal{L}}{\partial \mu_l}. \tag{4.20}$$

Similar to PMF [114], we linearly map the rating values in $[1, D]$ to $[0, 1]$ and pass $U_i^T V_j$ through the sigmoid function as defined in Eq. (4.13). To avoid cluttered notations, we drop all the logistic function in our derivation process. After obtained the trained model, we convert the expected value, $g(U_i^T V_j)$, back to the scale of 1 to $D$ and set it as the predicted score of user $i$'s rating on item $j$. We provide an algorithm for learning model parameters in Algorithm 6.

### 4.3.5 Context aware response model

In real-world recommender systems, the probability of an item being rated may not only depend on users' rating score. Many factors affect the response probabilities. For example, in a movie rating system, some popular movies such as Titanic, Avatar, may have much higher probability of being rated than a mediocre movie. Moreover, the features of users and items may contain group structure [134]. One may argue this might be caused by the higher inspection rate, i.e., it is likely that a reputable movie is being watched more than an obscure one. Nevertheless, it still makes sense that some items

**Algorithm 6** Response Aware PMF(RAPMF)

Parameters: $N, M, D, K, \lambda_U, \lambda_V, \lambda_\mu, \beta$
Input: Rating and Response Matrix $(X, R)$
Initialize $U \in \mathbb{R}^{K \times N}$ and $V \in \mathbb{R}^{K \times M}$ randomly
Initialize average gradient matrix to $\mathbf{0}$

$$Y_U \in \mathbb{R}^{K \times N} \leftarrow \mathbf{0}; \quad Y_V \in \mathbb{R}^{K \times M} \leftarrow \mathbf{0}$$

**while** Stop criteria not met **do**
$\hat{X} \leftarrow U^T V$
Increase index $T_{U_u} \leftarrow T_{U_u} + 1$ and $T_{V_i} \leftarrow T_{V_i} + 1$
$t_u \leftarrow T_{U_u}, \quad t_v \leftarrow T_{V_i}$
Update average gradient $Y_U$ and $Y_V$

$$Y_{U_u} \leftarrow \frac{t_u - 1}{t_u} Y_{U_u} + \frac{1}{t_u}(g_{ui} - r) g'_{ui} V_i$$
$$Y_{V_i} \leftarrow \frac{t_v - 1}{t_v} Y_{V_i} + \frac{1}{t_v}(g_{ui} - r) g'_{ui} U_u$$

Update latent user and item feature $U_u$ and $V_i$

$$U_u \leftarrow -\frac{1}{2\lambda_U} Y_{U_u}, \quad V_i \leftarrow -\frac{1}{2\lambda_U} Y_{V_i}$$

**end while**

may have higher chance of receiving a rating due to the high quality that a user will not hesitate to rate it. In addition, different user may have distinct rating habits. Some users might be more willing to provide ratings in order to get high quality recommendation. This is supported by the fact that the number of ratings received from different users can differ wildly in real-world deployed recommender systems.

To capture such factors, we generalize the rating dominant response model by including both item features and user features. To keep the model tractable and efficient, we introduce a linear combination of the item features, user features and a constant related to the rating scores and pass it through the logistic function to model the response probability,

$$\mu_{ijk} = \frac{1}{1 + \exp(-(\delta_k + U_i^T \boldsymbol{\theta}_U + V_j^T \boldsymbol{\theta}_V))}. \tag{4.21}$$

We refer to Eq. (4.21) as context-aware response model, in which the response probability is on a per-user-item-rating basis. More sophisticated relationship definition can be referred to [129, 133, 135]. The PMF integrated with the context-aware response model is named RAPMF-c. Note that by setting $\boldsymbol{\theta}_U$ and $\boldsymbol{\theta}_V$ to zero, we can recover the rating dominant response model in Eq. (4.13).

The log-likelihood of RAPMF-c is in the same structure as RAPMF-r. We only need to substitute $\mu_k$ in Eq. (4.15) by $\mu_{ijk}$ defined in Eq. (4.21). Similarly, the gradients of $\mathcal{L}$ with respect to $U_i$ and $V_j$ are

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial U_i} =& \beta \sum_{j=1}^{M} \frac{\sum_{k=1}^{D} t_{Ukij} \mathcal{N}(k|U^T V, \sigma^2)}{\sum_{k=1}^{D} \alpha_{kij} \mathcal{N}(k|U^T V, \sigma^2)} \\ & - \sum_{j=1}^{M} (U_i^T V_j - x_{ij})[r_{ij} = 1] V_j - \lambda_U U_i,\end{aligned} \tag{4.22}$$

$$\frac{\partial \mathcal{L}}{\partial V_j} = -\beta \sum_{i=1}^{N} \frac{\sum_{k=1}^{D} t_{Vkij} \mathcal{N}(k|U^T V, \sigma^2)}{\sum_{k=1}^{D} \alpha_{kij} \mathcal{N}(k|U^T V, \sigma^2)} - \sum_{j=1}^{M} (U_i^T V_j - x_{ij})[r_{ij} = 1] U_i - \lambda_V V_j, \tag{4.23}$$

where the $t_{Ukij}$ and $t_{Vkij}$ is defined as following

$$t_{Ukij} = g'(\mu_{kij})(-1)^{[r_{ij}=0]} \boldsymbol{\theta}_U - \alpha_{kij}(U_i^T V_j - k) V_j, \tag{4.24}$$

$$t_{Vkij} = g'(\mu_{kij})(-1)^{[r_{ij}=0]} \boldsymbol{\theta}_V - \alpha_{kij}(U_i^T V_j - k) U_i. \tag{4.25}$$

Correspondingly, the gradients of $\mathcal{L}$ with respect to $\delta_l$, $\boldsymbol{\theta}_U$ and $\boldsymbol{\theta}_V$ are

$$\frac{\partial \mathcal{L}}{\partial \delta_l} = \sum_{i=1}^{N} \sum_{j=1}^{M} \frac{\mathcal{N}(l|U^T V, \sigma^2) g'(\mu_{kij})(-1)^{[r_{ij}=0]}}{\sum_{k=1}^{D} \alpha_{kij} \mathcal{N}(k|U^T V, \sigma^2)} - \lambda_\mu \delta_l, \tag{4.26}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_U} = \sum_{i=1}^{N} \sum_{j=1}^{M} \frac{\sum_{k=1}^{D} \mathcal{N}(k|U^T V, \sigma^2) g'(\mu_{kij})(-1)^{[r_{ij}=0]} U_i}{\sum_{k=1}^{D} \alpha_{kij} \mathcal{N}(k|U^T V, \sigma^2)} - \lambda_\mu \boldsymbol{\theta}_U, \tag{4.27}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_V} = \sum_{i=1}^{N} \sum_{j=1}^{M} \frac{\sum_{k=1}^{D} \mathcal{N}(k|U^T V, \sigma^2) g'(\mu_{kij})(-1)^{[r_{ij}=0]} V_j}{\sum_{k=1}^{D} \alpha_{kij} \mathcal{N}(k|U^T V, \sigma^2)} - \lambda_\mu \boldsymbol{\theta}_V. \tag{4.28}$$

To learn RAPMF-c, we adopt the alternatively updating scheme to maximize the log-likelihood, where the updating rules of $U$ and $V$ are the same as those in Eq. (4.19). After updating $U$ and $V$, we update $\delta_l$, $\boldsymbol{\theta}_U$ and $\boldsymbol{\theta}_V$ by

$$\boldsymbol{\vartheta} \leftarrow \boldsymbol{\vartheta} + \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{\vartheta}},$$

where $\boldsymbol{\vartheta}$ is replaced by $\delta_l$, $\boldsymbol{\theta}_U$ and $\boldsymbol{\theta}_V$, respectively.

### 4.3.6 Complexity and Parallelization

The training complexity of RAPMF, $O(MN)$, can be quite time consuming compared with the PMF, which is linear in number of observations, $O(|\mathcal{Q}|)$. However, we argue that the time spent on training is worthy since it can boost the model performance. More importantly, the prediction complexity of RAPMF is the same as PMF, $O(K)$, which can be taken as a constant time given a moderate sized $K$. Since the training procedure can be performed offline, RAPMF can accommodate the hard response time constraint in real-world deployed recommender systems due to the succinct prediction cost.

In addition, RAPMF can be speedup by parallelization. The intensive computation cost, calculating the gradients, can be decoupled and distributed to a cluster of computers. It is also possible to use online learning to speed up the training process [68].

## 4.4 Experiments and Results

We conduct empirical evaluation to compare the performance of PMF [114], CPT-v [87], Logit-vd [86], and our RAPMF. We try to answer the following questions:

1. How to collect data with benchmark response patterns to evaluate the models fairly?
2. How to design experiment protocols to evaluate the performance the models with and without response models fairly?
3. How the compared models perform on the collected data?
4. How the parameters, $\beta$ and $\lambda$, affect the performance of RAPMF?

Sections 4.4.1-4.4.4 answer the above questions, respectively.

Before discussing the protocol we use in this chapter, we will describe traditional experiment protocol used in evaluating recommender system and its limitations.

Traditionally, experiments for recommender system are usually done using existing publicly available dataset like MovieLens [113]

or Netflix [5]. These datasets contain a list of user item rating triplets and optionally some additional information on users and items. Ratings in these datasets are often collected from real life system that allow user to rate items. The protocol used to evaluate a recommender system's performance is to divide the dataset into two disjoint datasets, one for model training and the other for model testing. To evaluate a newly proposed recommender system model, usually we feed the training set into the model for parameter learning. When the training is complete, we use the learned model to make predictions for the user item pairs that appear in the hold-out testing set and compare them to the real ratings.

We argue that use traditional protocol to test the performance of a model is problematic. The items in a recommender system can have one of three different statuses: un-inspected, inspected and not rated, inspected and rated. Note that the status of an item is on a per-user basis. What we have in the publicly available datasets, are those of the third status, i.e., inspected and rated. That is, all we have in these datasets are user selected ratings. What we want to recommend is, however, those of the first status, i.e., un-inspected. As has already been demonstrated, the distribution of user selected ratings is quite different from the distribution of randomly selected ratings. We have shown that ignoring the response patterns can result a model that have bias. This chapter is about how to learn a model that take into account the response information. Similarly, using the user selected items to evaluate the performance of a model also has *bias* effect.

In fact, the traditional experiment protocol used for evaluating recommender system may hide the significance of the problem of ignoring the response model. Since the training set and the testing set are all user selected, by ignoring the response model, data model alone can handle the dataset without bias.

Ideally, we would like to evaluate a recommender system's performance by making recommendation to users and then collect users' feedback on those recommended items. A working recommender

system is required to conduct such kind of experiments. And to have statistical sound performance measure, we will need a large user base. Since we do not have access to such large scale real life recommender system, we choose to evaluate our model using synthetic dataset.

### 4.4.1 Datasets

We conduct our empirical analysis on two datasets: a synthetic dataset and a real-world dataset, the *Yahoo! Music ratings for User Selected and Randomly Selected songs, version 1.0* (Yahoo dataset)[1].

In this dataset, since we have the full data matrix, we take the opportunity to investigate the relative performance of different models under different settings. In the synthetic dataset, since we have the full data matrix, we can conduct the experiments using all the three protocols designed before and better reveal the properties of the proposed models.

**Synthetic dataset.** The data generation process consists of two steps: generating full rating matrix and generating response matrix. Generating the full rating matrix models the how users rate items. Generating the response matrix models the process that how users choose whether to respond or not. One may argue that these two processes are different from real situation, in which the first step is to have some items inspected by each user, and then among the inspected items, some of them are being rated. The difference in our settings and the inspect-rate situation is how a user inspects items. In our setting, we implicitly assume that the inspection is performed randomly for each user. Although this assumption might not be 100 percent true in real situation, it provides a good approximation which allows us to focus on the response models. The inspection model could be an interesting future work.

To generate the full rating matrix, we first generate the latent user

[1] http://webscope.sandbox.yahoo.com

Table 4.2: Parameters for generating the synthetic dataset.

| $N$ | $M$ | $D$ | $K$ | $P_{inspect}$ |
|---|---|---|---|---|
| 1000 | 1000 | 5 | 5 | 0.2 |
| $P_1$ | $P_2$ | $P_3$ | $P_4$ | $P_5$ |
| 0.073 | 0.068 | 0.163 | 0.308 | 0.931 |

features and item features from zero-mean spherical Gaussian as follows:

$$U_i \sim \mathcal{N}(\mathbf{0}_K, \sigma_U^2 \mathbf{I}_K), \quad V_j \sim \mathcal{N}(\mathbf{0}_K, \sigma_V^2 \mathbf{I}_K),$$

where $i = 1, \ldots, N$, $j = 1, \ldots, M$, $\mathbf{0}_K$ is a $K$-dimensional vector with each element being 0 and $\mathbf{I}_K$ is the $K \times K$ identity matrix. The full rating matrix $X$ is then obtained by re-scale the sigmoid value of $U^T V$ to 1 to $D$ by $X_{ij} = \lceil g(U_i^T V_j) \times D \rceil$.

To generate the response matrix $R$, we first set the inspection probability of a user inspecting an item, $P_{inspect}$. Then, the partitioning of inspected ratings and un-inspected ratings are done by the Bernoulli trails with success probability $P_{inspect}$. For all the inspected ratings, we model their response probability by a Bernoulli distribution with the success probability $P_k$, where $k \in \{1, 2, \ldots, D\}$. Table 4.2 summarizes the parameters used for generating the synthetic dataset. The parameters are selected so that they can faithfully simulate real users' ratings and response behaviors. The rating probabilities $P_k$ are chosen according to Fig. 4.1. To minimize the effect of randomness, we generate the dataset independently 10 times and report the average result in the following. On average, we provide about 3.3% of the full matrix as training set, around 3.4% as testing set for traditional protocol, around 17.3% as testing set for adversarial protocol and all the remaining 80% as testing set for realistic protocol.

**Yahoo dataset.** It provides a unique opportunity to investigate the response ignorance problem. The dataset contains 311,704 *training ratings* collected from 15,400 users on 1,000 songs during the

normal interaction between the users and the Yahoo! Music system, with at least 10 ratings for each user. During a survey conducted by Yahoo! Research, exactly 10 songs randomly selected from these 1,000 songs are presented to the user to listen and rate. In total there are 5,400 users participated this survey and these 54,000 ratings are the *testing ratings*.

For the Yahoo dataset, we have no information on the inspected-and-not-rated items. We cannot perform experiments under the adversarial protocol due to this reason. We conduct experiments using traditional and realistic protocol only.

### 4.4.2 Setup and Evaluation Metrics

In a real-world deployed recommender system, the status of an item given a user follows exactly one of the three types: *un-inspected*, *inspected-unrated*, and *inspected-rated*. Traditional collaborative filtering approaches separate the inspected-rated data into training set and test set and evaluates the model on the test set. Since both the training set and the test set belongs to the inspected-rated type, their rating distributions are the same. Thus, the traditional evaluation scheme may hide the significance of the ignoring response model. In the experiment, we first investigate two existing experimental protocols:

- **Traditional protocol:** Both the training set and the test set are randomly selected from inspected-rated items together with their rating users and assigned scores. This is exactly the traditional experiment protocol [114].
- **Realistic protocol:** The training set is randomly selected from inspected-rated items, but the test set is randomly selected from un-inspected items. This is an experimental protocol adopted in [86, 87]. This protocol captures the ultimate goal of a recommender system, i.e., recommending un-inspected items to potential users who are interested.

Moreover, we will investigate a new experimental protocol:

- **Adversarial protocol:** The training set is randomly selected from inspected-rated items, but the test set is randomly selected from inspected-unrated items. This setting tests the model's performance when the distribution of the training set and test are very divergent. It can reveal the property of the model in some real-world cases where most of inspected-rated items receive very high scores, while those inspected-unrated items have low scores. This setting also demonstrates the model performance when we have an adversary that manipulates the responses.

For the synthetic dataset, we use the same training set for all three protocols. For various protocols, different test set can reveal different properties of the PMF with and without the response models. We report the average performance on the 10 independently generated datasets.

For Yahoo dataset, we use only traditional and realistic protocol and do not evaluate the adversarial protocol due to the missing of necessary inspection information. For the traditional protocol, we perform 10 fold cross validation on the training ratings. For the realistic protocol, we train the model using training ratings and test on the testing ratings. We perform the experiment 10 times and report the average results.

In the experiment, we use Root Mean Square Error (RMSE) to evaluate the performance of various approaches [87, 114, 127], i.e., RMSE $= \sqrt{1/|\mathcal{T}| \sum_{(i,j,x)\in\mathcal{T}} (\hat{x}_{ij} - x)^2}$, where $\mathcal{T}$ is the set of $(i, j, x)$ triplets reserved for testing and $\hat{x}_{ij}$ is model prediction for user $i$'s rating on item $j$.

### 4.4.3 Model Comparison

For both the synthetic dataset and Yahoo dataset, we randomly select 10% of the testing ratings from realistic protocol as validation set

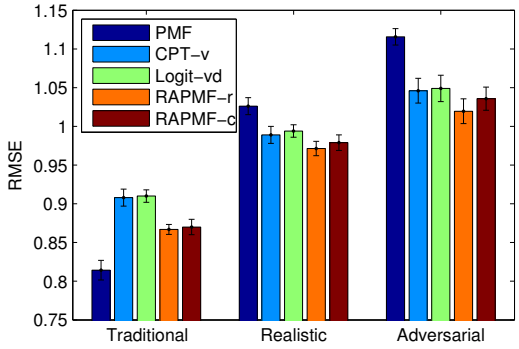
to tune the parameters (more advanced techniques can be referred to [109]):

- $\lambda_U$ and $\lambda_V$: They are tuned by the grid search scheme, i.e., first selecting from $\{10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10, 10^{2}\}$, respectively. We then fine-tune the range to achieve the best performance of PMF; see an example in Fig. 4.4.
- $\beta$: We first fix the optimal $\lambda_U$ and $\lambda_V$ obtained from PMF, we then tune it in $\{0.0, 10^{-3}, 10^{-2}, 10^{-1}, 1.0\}$ and fine-tune it further for RAPMF-r; see an example in Fig. 4.3.
- $\lambda_\mu$: As shown later in Fig. 4.5, this parameter is insensitive on a large range.

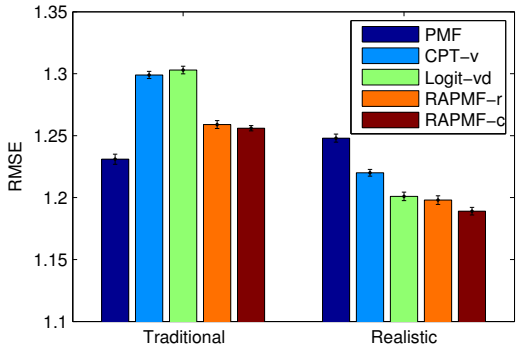
These parameters are then used across different protocols. The hyper-parameters used for CPT-v and Logit-vd follow the settings used in [86]. We choose $K = 5$ as the latent dimension size for all the experiments. All the models are trained using 500 iterations. According to our experience, the change in performance after 200 iterations is negligible.

Figure 4.2 shows the results of various models' performance under different protocols on both the synthetic and Yahoo datasets. Figure 4.2(a) shows the results on the synthetic datasets. We see that PMF performs best under traditional protocol. This is expected because under the traditional setting, the testing set and the training set have exactly the same distribution. The response model does not help. However, under realistic and adversarial protocol, the proposed RAPMF-r outperforms PMF by 5.5% and 9.2%, with 95% confidence level on the paired $t$-test, respectively. The RAPMF-c performs slightly worse than RAPMF-r. This is probably due to the reason that we does not take the user and item features into account when generating the dataset.

More importantly, the learned rating probability for a typical run of RAPMF-r is [0.0125, 0.0124, 0.0155, 0.0267, 0.105]. Comparing




(a) Synthetic dataset




(b) Yahoo dataset

Figure 4.2: Relative performance of various models. Smaller value indicates a better model performance.

this with the parameter used in Table 4.2 when generating the data, we see that although RAPMF cannot recover the rating probabilities exactly, the overall trend is captured quite precisely. This explains the significant performance boost in realistic and adversarial protocol.

Figure 4.2(b) shows the results on the Yahoo dataset. Again, PMF attains the best performance under traditional protocol. Under realistic protocol, the RAPMF-r and RAPMF-c outperforms PMF by 4.1% and 4.9%, with 95% confidence level on the paired $t$-test, re-

spectively. The performance gain is slightly less than that in the synthetic dataset, probably due to the reason that the rating probability in real-world dataset is not as dynamic as the value we choose in Table 4.2. The performance boost gained from context-awareness is not as significant as we have expected. This result hints that the rating value might impact a user's decision on whether to rate the item more than the user and item features.

### 4.4.4 Sensitivity Analysis

In the following, we investigate how the model parameters affect the performance of RAPMF. All the sensitivity analysis is done under the realistic setting in one-trial of the generated synthetic dataset.

#### Impact of $\beta$

The faith parameter $\beta$ is arguably the most important parameter in our RAPMF model. As has been discussed in Section 4.3, we adopt a soft-margin approximation of the ideal hard-margin response model. When making this approximation, we unavoidably introduce some bias in the response model because we cannot fully recover the true $U$ and $V$. Hence, we introduce $\beta$ to control the weight of the response model. Especially when the training process is just started, the probability can diverge wildly from the truth. Instead of trust the response model blindly, we use $\beta$ to control the weight we place on the response model.

Figure 4.3 plots the performance of RAPMF versus $\beta$ on the logarithmic scale. When $\beta = 0$, RAPMF fall back to PMF, whose performance is 1.015, corresponding to the most left point in Fig. 4.3. Clearly, by incorporating an explicit response model, RAPMF is able to beat PMF by a large margin (nearly 5%) by using a proper $\beta$ value. However, if we use too large a $\beta$, we quickly lose the boost provided by the response model. An observation of the experiment is that when $\beta$ is too large, the model does not converge. This is be-

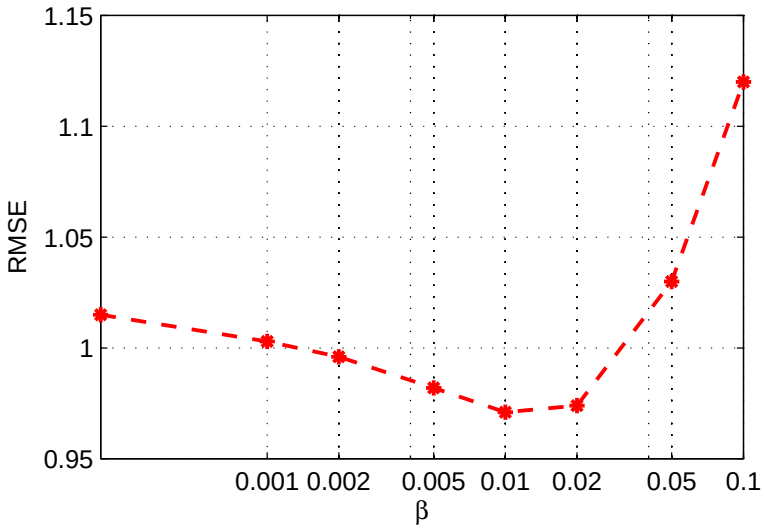



Figure 4.3: Sensitivity analysis of $\beta$ on RAPMF on one-trial test.

cause the model training starts from randomly initialized $U$ and $V$, a large weight on the response model will pull the model away from the true model and cause divergence. If we place too much weigh on the response model, the model is unlikely to converge.

We did not include the result when $\beta$ is bigger than $0.1$. In fact, when $\beta$ is too large, the model won't even converge. This is expected because we initialize $U$ and $V$ randomly. The probability measure would diverge wildly from the truth. If we place too much weigh on the response model, the model is unlikely to converge.

#### Impact of $\lambda'$s

The regularization parameters $\lambda$ are placed on $U$, $V$ and $\mu$. Since in the dataset, users and items are symmetric, we use the same regularization parameter $\lambda_{UV}$ for $U$ and $V$ and use another parameter $\lambda_\mu$ to control $\mu$.

Figure 4.4 shows the impact of $\lambda_{UV}$ on the performance of RAPMF. When $\lambda_{UV}$ is very small, although the RAPMF is able to fit the training data very well, it does not generalize well to the test set. This

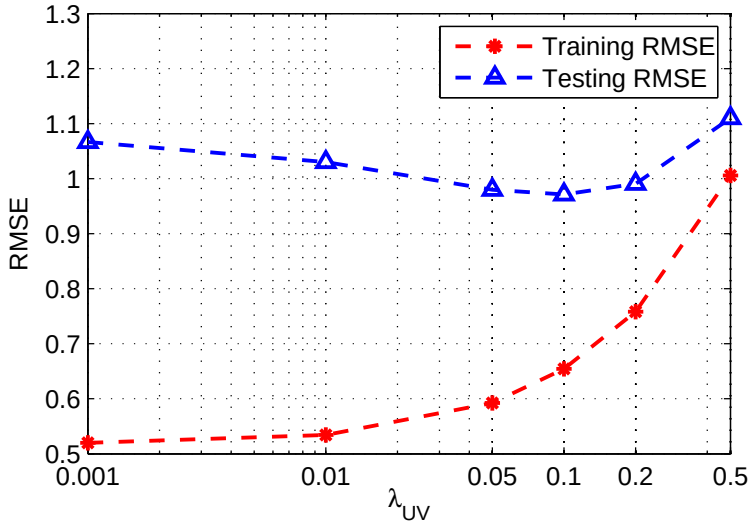



Figure 4.4: Sensitivity analysis of $\lambda_{UV}$ on RAPMF on one-trial test.

is a sign of over-fitting. As $\lambda_{UV}$ becomes larger, which limits the norms of $U$ and $V$, the training RMSE increases but the test RMSE decreases gradually. However, after a turning point, both the training RMSE and test RMSE start to increase. This is when the regularization is too stringent that it hinders the proper fitting of the model.

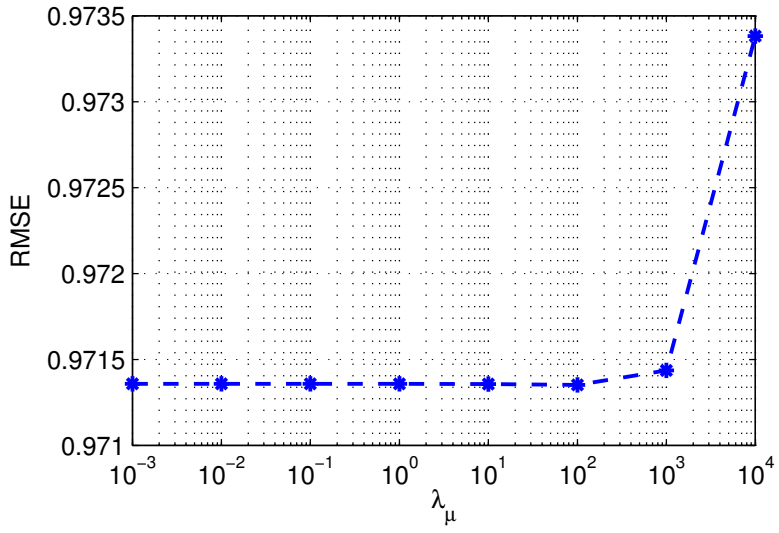
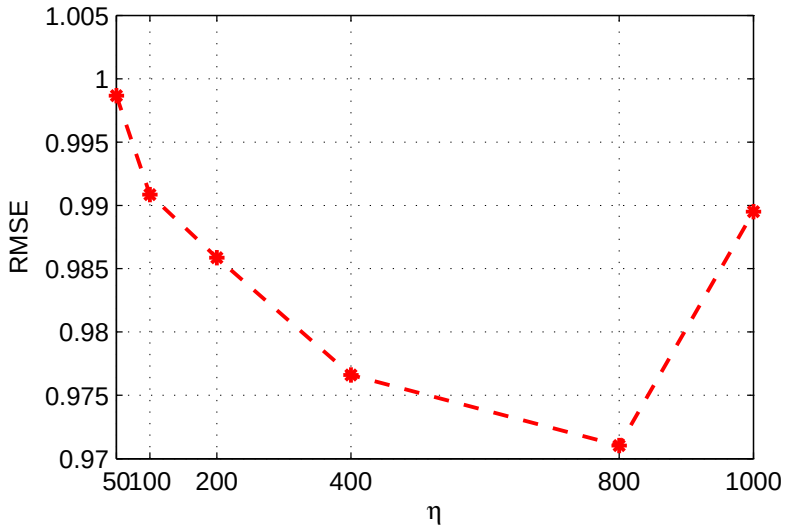



Figure 4.5: Sensitivity analysis of $\lambda_{\mu}$ on RAPMF on one-trial test.

Figure 4.5 shows the impact of $\lambda_\mu$ on the model performance. Since the number of parameters to learn in $U$ and $V$ are more than the number of parameters in $\mu$, to effectively regularize $\mu$, we need a much larger $\lambda_\mu$ than $\lambda_{UV}$. This evidence is observed in Fig. 4.5. As we can see, it is basically a straight line in a large range from $10^{-3}$ to $10^4$, while the RMSE is changed only from 0.9714 to 0.9734, a very small scale. The effect of $\lambda_\mu$ is inappreciable. This is probably due to the fact that $\mu$ is a parameter in $D$-dimension ($D$=5) and no significant over-fitting can occur.

**Impact of $\eta$**

Figure 4.6: Impact of $\eta$ in RAPMF




The parameter $\eta$ is the learning rate, deciding how large a step to take when updating the parameters. When updating $U$, the actual learning rate we use is $\eta/|V|$ and likewise $\eta/|U|$ for updating $V$. When updating $\mu$, the true learning rate is $\eta/|V|/|U|$. This scaling is done to ensure that the actual steps taken are on the same scale when updating different parameters. Figure 4.6 plots the impact of $\eta$ on the model performance. It is shown that $\eta = 800$ is an ideal choice.

## 4.5 Summary

In this chapter, we propose two response models, rating dominant and context-aware response models, to capture users' response patterns. Further, we unify the response models with one of famous collaborative filtering model-based methods, the Probabilistic Matrix Factorization, to establish the Response Aware Probabilistic Matrix Factorization framework (RAPMF). The RAPMF also generalizes PMF as its special case. Empirically, we verify the performance of RAPMF under carefully designed experimental protocols and show that RAPMF performs best when it tries to fulfill the ultimate goal of real-world recommender systems, i.e., recommending items to those who may be interested in. The empirical evaluation demonstrates the potential of our RAPMF model in real-world recommender system deployment.

---

□ **End of chapter.**

# Chapter 5

# User Reputation Estimation

## 5.1 Problem and Motivation

Recommender systems become more and more important for online service providers. Large real-life recommender systems nowadays usually adopt collaborative filtering techniques [5, 7]. These techniques are essentially social systems that predict a user's preference on an item by leveraging the judgment of other people. They are vulnerable to spammers and manipulations of ratings. There are evidences showing the existence of spammers in online rating systems. The ratings assigned by spammers can contaminate the system and affect recommendation accuracy. Detecting these spammers and eliminating their ratings from the system is thus crucial to online rating systems.

Malicious users have been found in rating systems in the literature [41]. As early as 2005, there is news reporting on a group of people managing to trick Amazon's recommender system[1]. Recently, there is an incident that a user created non-existing item receiving hundreds of reviews in a major social rating website. Though the item is deleted quickly, this incident still caused a splash in the users. The users' behavior on the fantasy item hinted that their ratings for other items could also be imaginary and malicious. From a

[1] http://news.cnet.com/2100-1023-976435.html

recommender system's point of view, such ratings do not have positive impact on the system, because users' irresponsible ratings may contaminate the features of the items that they have rated.

A user reputation system can be employed to detect and mitigate the effect of spammers in such rating systems. A reputation system can estimate the credibility of a user by analyzing the rating behavior of the user. In previous studies of user reputation estimation, an item is usually assumed to have an intrinsic utility that is common to all users [63, 29]. Then a user's rating is compared with this utility and the deviation between the rating and the utility determines the reputation score of the user. This quality view of item might be true for some rating systems, for example, an online commodity shop where the ratings are assigned based on the utility of the commodity. However, it may not be true in other types of rating systems. For example in a movie recommendation system, Star Trek might be the favorite movie for a Sci-Fi fan while it is tiring to a documentary lover. The utility score of an item can be quite different for different users in this scenario. Lower the documentary lover's reputation score because she assigned a very low score to Star Trek might be inappropriate. As long as the scores assigned to items are consistent, one's reputation score should not be penalized.

Reputation score should measure users' *consistency* and *predictability* in assigning the ratings to items. Given a reasonable model that captures the known ratings, the consistency and predictability can be measured with respect to the model prediction. If a user always assigns ratings to items in a way that is surprising and unpredictable, it is likely that he is a spammer. For example, if a user assigns a very high score to the movie Batman Dark Knight and a very low score to The Dark Knight Rises, it is inconsistent and unpredictable since these two movies are produced by the same director and have very similar characteristics. From the model's point of view, the two ratings are contradictory and cannot be trusted completely. We incorporate this idea into our framework and introduce a new method

based on the framework.

We need to point out that we cannot guarantee an error-free spam detection using the ratings alone. There could simply be a subpopulation of users who hold a view difference from other users. On the other hand, a spammer could also pretend to be a normal user by following the majority's opinions on most items except for the few items the ratings of which he/she wants to manipulate. Another potential problem with spam detection in online rating systems is spammers becoming the majority. Using the ratings alone, if the spam users become the majority under a coordinated control, then the detection methods could lead to false conclusions. In small online rating systems with few users, this situation impose a serious potential issue. However, in a large system with large user base, this is unlikely to happen. Other type of information, social network connections between users for example, can greatly increase our confidence in identifying the spam users. However, we defer the impact of other types of information in identifying the spammers to a future work. In this chapter, we deal with the case where we need to identify spammers using ratings alone.

Our contributions are three-fold. First, we propose a unified framework for reputation estimation in which the fruitful models in collaborative filtering (CF) can be readily plugged-in. We estimate the reputation of a user based on the deviation of her ratings versus the model output and penalize the deviation through a penalty function. Then this penalization quantity is transformed to the reputation through a link function. We show that previously proposed reputation estimation methods can be instantiated as special cases of our framework by choosing suitable CF models, penalize functions and link functions. Secondly, we introduce a low-rank matrix factorization based reputation estimation method by leveraging our framework. Thirdly, we conduct empirical study of various algorithms under several spamming strategies, i.e., *Random attacks*, *Bandwagon attacks* [17], *nuke* and *push* [99].

The rest of this chapter is organized as follows. In Section 5.2, we propose the framework and show how previous work can be incorporated into this framework. Section 5.3 introduces a low-rank matrix factorization based reputation estimation method. We perform empirical study in Section 5.4 and summarize this chapter in Section 5.5.

## 5.2 Reputation Estimation Framework

In this section, we introduce the reputation estimation framework and demonstrate its adaptability [66].

### 5.2.1 Problem formulation

Given a set of $N$ users $\mathcal{U} = \{u_1, u_2, \cdots, u_N\}$ and a set of $M$ items $\mathcal{I} = \{i_1, i_2, \cdots, i_M\}$, users' ratings on items form an $N \times M$ matrix $R$, where the entry on the $i$th row $j$th column $r_{ij}$ denotes $u_i$'s rating on $i_j$. Alternatively, we denote users' ratings on items by a set of triples $\mathcal{Q} = \{(i, j, r)\}$, where $r_{ij} \in R$. Let the set of items that are rated by $u_i$ be $\mathcal{I}_i$ and the set of users who have rated $i_j$ be $\mathcal{U}_j$. We assume that the ratings are integers from 1 to $D$ (real ratings can also be accommodated without a problem). Then a reputation estimation model tries to estimate the reputation $c_i$ for each of the users $u_i$. It is convenient to require that $0 \leq c_i \leq 1$, with a larger value indicating the reputation of $u_i$ is higher.

### 5.2.2 The Framework

Our framework is composed of three ingredients as an integrated paradigm, namely the prediction model, the penalty function and the link function.

**Prediction Model**

Given a collaborative filtering model $\mathcal{H}$ which can predict the entries of $R$, we assume that the $i$th row, $j$th column of $R$, $r_{ij}$, is a Gaussian random variable centered at model prediction $\mathcal{H}(i,j)$ with variance $\sigma^2$,

$$r_{ij} \sim \mathcal{N}(\mathcal{H}(i,j), \sigma^2).$$

Then the log-likelihood of observing $r_{ij}$ given $\mathcal{H}(i,j)$ is

$$\begin{aligned}\mathcal{L}_{ij} &= \log(P(r_{ij}|\mathcal{H}(i,j))) \\ &= \log(\mathcal{N}(r_{ij}|\mathcal{H}(i,j), \sigma^2)) \\ &= C - \frac{1}{2\sigma^2}(r_{ij} - \mathcal{H}(i,j))^2,\end{aligned}$$

where $C$ is a constant. The quantity

$$s_{ij} = (r_{ij} - \mathcal{H}(i,j))^2$$

can be interpreted as the *unexpectedness*, which is inversely related to the *predictability* of the rating.

If we take the probability mass function $\mathcal{N}(x|\mu, \sigma^2)$ of a Gaussian random variable as the approximation of the probability that an event $E$ happens within a small $\epsilon$-neighborhood of $x$, then $\mathcal{L}_{ij}$ has a natural explanation, i.e., it is the *self-information* of the outcome $E$, which is a measure of information associated with the outcome of a random variable. The larger the $s_{ij}$ is, the more surprising it is for the rating $r_{ij}$ is not expected by $\mathcal{H}$. A well behaved user should have small $s_{ij}$ for the ratings. The underlying reasoning is the same as the assumption made by a collaborative filtering methods, i.e., there should be a few factors that can explain the behavior of a normal user. A well-trained model $\mathcal{H}$ should be able to learn the underpinning factors and make reasonable predictions. On the contrary, if the unexpectedness $s_{ij}$ is large for the ratings given by $u_i$, it is a sign that the ratings are not *accordant* with other ratings in the system and the user is not behaving predictably as the model $\mathcal{H}$ assumes.

If we take a different perspective, we see that $s_{ij}$ is the distance between the model $\mathcal{H}$'s prediction and the observation, which is the $L_2$ measure of training error. In other words, $s_{ij}$ indicates how well the ratings $r_{ij}$ can be trained using current model $\mathcal{H}$. This method of detecting unexpectedness coincides with the Type 1 approach of outlier detection [44].

Since the model $\mathcal{H}$ can predict every entry of $R$, we can readily compute the unexpectedness of all the known ratings $s_{ij}$, for all $(i, j, r) \in \mathcal{Q}$. For user $u_i$, the set $\{s_{ij} | j \in \mathcal{I}_{u_i}\}$ measures the unexpectedness of seeing each of the ratings assigned by the user.

#### Penalty Function

The purpose of the penalty function is summarizing the set of unexpectedness $\{s_{ij}\}$ into one quantity $s_i$ or $s_j$, where the former represents the overall unexpectedness of $u_i$ and the later represents the overall unexpectedness of $i_j$. Then $s_i$ or $s_j$ is used by the link function to get the reputation score for user $u_i$. As an example, the penalty function that computes $s_i$ can be a simple arithmetic mean,

$$s_i = \frac{1}{\|\mathcal{I}_i\|} \sum_{j \in \mathcal{I}_i} s_{ij}. \tag{5.1}$$

#### Link Function

After getting a summarizing unexpectedness $s_i$ for $u_i$ or $s_j$ for $i_j$, we can compute the reputation score for $u_i$. This task is performed by a link function. The link function should relate the reputation of a user inversely to the unexpectedness $s_i$. It is convenient and natural to require that the reputation of a user $c_i$ to lie in between $[0, 1]$, a simple link function that fulfills this requirement is

$$c_i = 1 - \frac{s_i}{s_{max}},$$

where $s_{max}$ is the maximum possible value of $s_i$.

### 5.2.3 Adaptability of the framework

We show the adaptability of the proposed framework by showing how previously proposed reputation estimation methods can be captured using suitable $\mathcal{H}$'s, penalty functions and link functions.

Mizzaro's algorithm [94] is one of the earliest work on reputation estimation. It assumes that each item has an intrinsic quality $q_j$ and measures the *steadiness* of object quality, which is then used to determine the reputation of the user. It can be captured by our framework by choosing

$$\mathcal{H}(i,j) = \sum_{i \in \mathcal{U}_j} c_i r_{ij} / \sum_{i \in \mathcal{U}_j} c_i, \tag{5.2}$$

$$s_j = \sum_{i \in \mathcal{U}_j} c_i,$$

and

$$c_i = \frac{\sum_{j \in \mathcal{I}_i} s_j (1 - \sqrt{\sqrt{s_{ij}}/s_{max}})}{\sum_{j \in \mathcal{I}_j} s_j}.$$

Note that this algorithm is iterative in nature. The employed model predicts $u_i$'s rating on $i_j$ as a weighted average of all the ratings assigned to item $i_j$, where the weight carried by $r_{ij}$ is the reputation score $c_i$ of a user. The algorithm begins by assigning equal reputation score to all users. Then it iteratively updates $c_i$ and $\mathcal{H}(i,j)$ using current parameter set, until a predefined convergence condition is met.

The algorithm proposed by Laureti et al. [60] can also be seen as an instantiation of the framework by choosing the model $\mathcal{H}$ as in Eq. (5.2), the penalty function as

$$s_i = \frac{1}{\|\mathcal{I}_i\|} \sum_{j \in \mathcal{I}_i} s_{ij}, \tag{5.3}$$

and the link function as

$$c_i = (s_i + \epsilon)^{-\beta}, \tag{5.4}$$

Table 5.1: Choice of $\mathcal{H}$, penalty function and link function in Li's algorithms

| | $\mathcal{H}$ | Penalty Function | Link Function |
|---|---|---|---|
| **L1-AVG** | Eq. (5.6) | $\frac{1}{\|\mathcal{I}_i\|}\sum_{j\in\mathcal{I}_i}\sqrt{s_{ij}}$ | $1-\lambda s_i$ |
| **L2-AVG** | Eq. (5.6) | $\frac{1}{\|\mathcal{I}_i\|}\sum_{j\in\mathcal{I}_i} s_{ij}$ | $1-\frac{\lambda}{2}s_i$ |
| **L1-MAX** | Eq. (5.6) | $\max_{j\in\mathcal{I}_i}\sqrt{s_{ij}}$ | $1-\lambda s_i$ |
| **L2-MAX** | Eq. (5.6) | $\max_{j\in\mathcal{I}_i} s_{ij}$ | $1-\frac{\lambda}{2}s_i$ |
| **L1-MIN** | Eq. (5.6) | $\min_{j\in\mathcal{I}_i}\sqrt{s_{ij}}$ | $1-\lambda s_i$ |
| **L2-MIN** | Eq. (5.6) | $\min_{j\in\mathcal{I}_i} s_{ij}$ | $1-\frac{\lambda}{2}s_i$ |

where $\epsilon$ is a small constant to prevent diverging and $\beta$ is the strength of the penalty applied to user. Note that one property of this algorithm is that $c_i$ does not necessarily lie in the range $[0, 1]$. However, since the original purpose of their algorithm is to estimate the intrinsic quality of an item, it does not pose a problem.

De Kerchove et al. [29] proposed an algorithm that is similar to Laureti's algorithm. It is an instance of the proposed framework by choosing the model as in Eq. (5.2), the penalty function as in Eq. (5.3) and the link function

$$c_i = 1 - k \times s_i, \tag{5.5}$$

where $k$ is chosen such that $c_i \geq 0$.

Motivated by the lack of a theoretical convergence guarantee, Li et al. [63] proposed a class of six algorithms for computing the reputation of users in a rating system. They proved their algorithms converge to a unique solution theoretically, which is a desired property of reputation estimation algorithms. However, in practice, the convergence is not a problem for all the mentioned algorithms [89]. Table 5.1 summarizes the choices of $\mathcal{H}$, the penalty function and the link function to recover the six algorithms.

$$\mathcal{H}(i,j) = \frac{1}{\|\mathcal{U}_j\|}\sum_{i\in\mathcal{U}_j} r_{ij}c_i. \tag{5.6}$$

Note that in Eq. (5.6), the prediction model is slightly different from that in Eq. (5.2). The denominator in Eq. (5.6) is chose so as to guarantee the convergence of the algorithms. Also note that the algorithms assume that all the ratings have been transformed into the range $[0, 1]$, which could easily be achieved by mapping $r'_{ij} = (r_{ij} - 1)/(D - 1)$. Through this mapping, the link function is guaranteed to output the reputation $c_i$ in the range of $[0, 1]$. Similarly, these six algorithms are iterative methods with initial $c_i$ set to 1.

The spammer detection algorithm proposed in [90, 92] can also be captured by choosing $\mathcal{H}$ to be PCA and PLSA, respectively.

## 5.3 Reputation Estimation using Low-Rank Matrix Factorization

In Section 5.2.3 we have shown how previous methods can be taken as instances of our framework. It is interesting to note that they share the same prediction model $\mathcal{H}$ (a slightly different weighting scheme is used in Li's algorithm). The prediction model is an item centric model, i.e., it outputs the same prediction score for all users' ratings on the same item. This prediction score is interpreted as the intrinsic item quality, which measures the utility of the item. In the aforementioned methods, the reputation of a user is measured based on how her ratings deviate from the qualities of the items. Such a scheme naturally assumes that there is indeed an intrinsic quality for an item.

However, as we argued in Section 5.1, the utility of an item could vary depending on the taste of the user. In fact, both the taste view and the intrinsic quality view have their area of application. Different prediction models suit different scenarios. In an e-commerce website, the items are mainly the products that are on sale in the website. In this case, the rating that a user gives to an item should depend more on the quality and functionality of the item. Therefore, the assumption that an item has an intrinsic quality is suitable.

On the other hand, in an online stream service website, where music and movies can be purchased and rated, the utility of an item (a song or a movie) could be different depending on the user's taste and preference.

We propose a reputation estimation system that utilizes a personalized prediction model to better capture the case where the taste view is more suitable. The prediction model we use is a low-rank matrix factorization method, which is shown to be a flexible model with good prediction accuracy [114].

### 5.3.1 Low-Rank Matrix Factorization

In a low-rank matrix factorization (MF) method, the rating matrix $R \in \mathbb{R}^{N \times M}$ is assumed to have a low-rank structure, i.e., it has a rank of $K \ll \min\{M, N\}$. Then $R$ can be decomposed into two rank-$K$ matrix $U \in \mathbb{R}^{K \times N}$ and $V \in \mathbb{R}^{K \times M}$ as $R = U^T V$. The column vectors of $U$ and $V$ have a natural interpretation. The $i$th column vector $U_i$ of $U$ is the latent factor that determines the behavior of $u_i$ and the $j$th column vector $V_j$ of $V$ is the latent factor that determines the features of $i_j$. The dot product $U_i^T V_j$ is the model predicted score of $u_i$'s rating on $i_j$.

The Probabilistic Matrix Factorization (PMF) adopts a probabilistic linear model with Gaussian observation noise [114]. Maximizing the posterior probability is equivalent to minimizing the following objective function

$$\mathcal{L} = \frac{1}{2} \sum_{(i,j,r) \in \mathcal{Q}} (r - U_i^T V_j)^2 + \frac{\lambda_U}{2} \|U\|_F^2 + \frac{\lambda_V}{2} \|V\|_F^2,$$

where the first term is the squared loss and the last two terms are regularization imposed to avoid over-fitting problem. To minimize the objective function, first compute the gradients of the objective

function with respect to $U_i$ and $V_j$ as

$$\frac{\partial \mathcal{L}}{\partial U_i} = \sum_{j \in \mathcal{I}_i} (U_i^T V_j - r_{ij}) V_j + \lambda_U U_i,$$
$$\frac{\partial \mathcal{L}}{\partial V_j} = \sum_{i \in \mathcal{U}_j} (U_i^T V_j - r_{ij}) U_i + \lambda_V V_j.$$

Then alternative update on $U_i$ and $V_j$ can be done through

$$U_i \leftarrow U_i - \eta \frac{\partial \mathcal{L}}{\partial U_i} \qquad V_j \leftarrow V_j - \eta \frac{\partial \mathcal{L}}{\partial V_j}.$$

After training, the prediction of user $i$'s rating on item $j$ as predicted by the model is the dot product $U_i^T V_j$. In practice, $r_{ij}$ is usually mapped into the range $[0, 1]$ and the sigmoid function is used to filter $U_i^T V_j$ before it is matched to $r_{ij}$. Apart from the batch-trained algorithm, online algorithms for learning PMF are also proposed [68], in which new users and new items can be accommodated more easily.

### 5.3.2 Penalty Function and Link Function

The penalty function we choose is the arithmetic mean in Eq. (5.1) and the link function we employ is

$$c_i = 1 - s_i.$$

We assume that the collected ratings have been mapped to the range $[0, 1]$ so that $s_i$ and $c_i$ are all in the range $[0, 1]$.

## 5.4 Experiments

In this section, we conduct empirical studies on the reputation estimation algorithms. We compare the algorithms' discrimination capabilities on identifying spammers in a rating system.

### 5.4.1 Datasets

Labeling spam users in a rating system is a tedious and time consuming job. Even for human annotators, usually additional information like text comments is needed to identify a spam user. To the best of our knowledge, there is no publicly available benchmark rating dataset that has ground truth spam user labels. However, there are several rating datasets that are well studied in the collaborative filtering community. These datasets contain ratings collected from real recommender systems. Usually the data in such datasets have gone through a filtering process, i.e., filtering out users who assign too few ratings and items that receive too few ratings. These datasets provide us a chance to evaluate the reputation estimation algorithms. We use MovieLens[2] dataset as the base dataset to evaluate the algorithms.

Basic statistics of MovieLens is shown in Table 5.2. By today's standard, a million-sized dataset is at most a median-sized dataset. We choose to use this dataset because the data are collected from a non-commercial recommender system. It is more likely that the users in this dataset are non-spam users. We take the users already in the dataset as normal users. We add spam users by mimic their rating behavior through the following strategies.

- **Random spamming:** Spam users rate the items in a random way. The rating that a spam user assigns to an item is chosen uniformly at random from $1$ to $5$. This strategy resembles the *random attacks* seen in spammer attacks [17].

- **Semi-random spamming:** Spam users rate half of their ratings randomly as is in the random rating scheme, and the other half rating is copied randomly from normal users' rating on the same item. This is similar to the *average attacks*.

- **Optimistic spamming:** Spam users assign half of their ratings using the highest possible score ($5$ in our case) and copy ran-

[2]http://www.cs.umn.edu/Research/GroupLens

Table 5.2: Statistics of MovieLens Dataset

| # of ratings | # of users | # of items | rating range |
|---|---|---|---|
| 1,000,209 | 6040 | 3706 | 1 to 5 |

domly from normal users on the other half items. This strategy resembles the *Bandwagon attacks*.

- **Pessimistic spamming:** Spam users assign half of their ratings using the lowest possible score (1 in our case) and copy randomly from normal users on the other half items. This is the *nuke* strategy used by spammers [99].

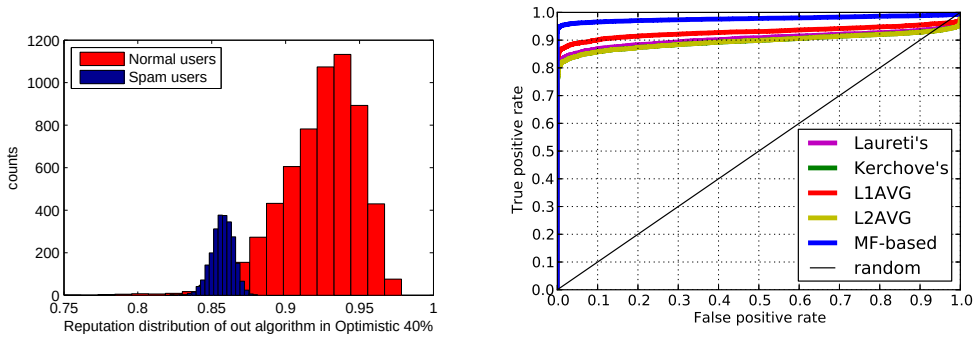
The number of spam users to add may also impact different algorithms' discrimination abilities. We choose to add 4 different levels of spam users to the dataset, 10%, 20%, 30% and 40% (percentage of spammers to normal users), to investigate the effect of the spammer rate. Each spam user randomly selects the average number of ratings received from normal users to rate.

In all four of our spamming strategies, we assume that the spammer would behave normally half of the time. In the real world situation, a spammer may vary this percentage to achieve his/her purpose and better hide himself/herself. Given the working mechanism of a rating based reputation estimation method, it becomes harder to identify a spammer as he increase the percentage of normal behavior. To effectively identify a spammer when the spammer behave normally most of the time, we need to alleviate other types of information to rectify the problem.

### 5.4.2 Evaluation Methods

To evaluate the performance of various methods, we employ the Area under the ROC Curve (AUC) [13] to measure their discrimination abilities. A sample distribution of reputation is shown in

Figure 5.1: Reputation distribution and ROC curve




(a) Reputation distribution

(b) ROC under Optimistic 20%

Figure 5.2(a). As a vertical separation line swipes through the x-axis, the true positive rate (TPR) and false positive rate (FPR) would vary depending on the position. The trade-off of TPR and FPR is shown in the ROC curve in Figure 5.2(b). AUC is the area under the ROC curve, which is a good single number summarization of the discrimination ability using only the reputation scores produced by each algorithm. Note the higher the AUC is, the better.

### 5.4.3 Implementation Details

For Laureti's algorithm, the $\beta$ in Eq. (5.4) is set to $-1$, which yields superior performance over other choices [89]. Parameter $k$ in Eq. (5.5) of De Kerchove's algorithm is set to be $k = [\epsilon + \max_{i \in \mathcal{U}} s_i]^{-1}$ to allow maximum separation. For Li's algorithm, $\lambda$ is set to 1, which yields the best performance. All algorithms are iterated until the change in reputation is less than $10^{-6}$ for all users.

For our proposed method, we use $K = 10$ latent factors in the comparison. In our experience, the MF model converges very fast and we train the model using 50 iterations.

### 5.4.4 Results

Shown in Table 5.3 and Table 5.4 are the Area Under the ROC Curve for different methods under various scenarios. It is evident that our MF-based method outperforms all the compared algorithms under all settings. Especially in the Optimistic case, our method outperforms the runner-up method by about 6% under the 40% spam users case. In real systems, spam users try to blend into the normal user. They might achieve this by mimic other users' rating on the irrelevant items. And for the items they want to promote, high rating scores should be assigned. Optimistic strategy is a good mechanism for generating such spam ratings. This improvement on AUC means more spam users can be identified without deteriorating the false positive rate. For Li's algorithm, we only show the result of L1-AVG and L2-AVG. Other algorithms including L1-MAX L2-MAX L1-MIN L2-MIN are significantly worse than using the average based penalty function under our experiment setting and we omit the results to save space.

Table 5.3: AUC comparison under Random and Semi-random Protocol

| **Type** | **Random** | | | | **Semi-random** | | | |
|---|---|---|---|---|---|---|---|---|
| **Percentage** | 10 | 20 | 30 | 40 | 10 | 20 | 30 | 40 |
| **Laureti's** | .9806 | .9803 | .9797 | .9790 | .9241 | .9240 | .9248 | .9248 |
| **Kerchove's** | .9793 | .9791 | .9785 | .9777 | .9227 | .9231 | .9239 | .9239 |
| **L1-AVG** | .9791 | .9789 | .9780 | .9769 | .9098 | .9111 | .9118 | .9115 |
| **L2-AVG** | .9790 | .9788 | .9782 | .9773 | .9224 | .9228 | .9237 | .9237 |
| **MF-based** | **.9893** | **.9896** | **.9896** | **.9892** | **.9685** | **.9676** | **.9673** | **.9668** |

With the increase of spam users percentage, the discrimination ability (measured by AUC) generally decreases, especially under Optimistic and Pessimistic strategies. It is interesting to point out that our method has the least decrease in AUC compared with other baseline methods. This is probably due to the fact that our method

Table 5.4: AUC comparison under Optimistic and Pessimistic Protocol

| **Type** | **Optimistic** | | | | **Pessimistic** | | | |
|---|---|---|---|---|---|---|---|---|
| **Percentage** | 10 | 20 | 30 | 40 | 10 | 20 | 30 | 40 |
| **Laureti's** | .9464 | .9298 | .9166 | .9047 | .9926 | .9914 | .9902 | .9887 |
| **Kerchove's** | .9428 | .9234 | .9090 | .8960 | .9910 | .9885 | .9858 | .9829 |
| **L1-AVG** | .9578 | .9465 | .9376 | .9295 | .9900 | .9875 | .9847 | .9817 |
| **L2-AVG** | .9425 | .9231 | .9088 | .8959 | .9902 | .9873 | .9841 | .9807 |
| **MF-based** | **.9884** | **.9858** | **.9814** | **.9774** | **.9939** | **.9938** | **.9937** | **.9936** |

utilizes a personalized prediction model while all other methods employ the weighted average. With more spam users, the prediction ability of the weighted average model deteriorates while MF can preserve the meaningful information provided by normal users.

### 5.4.5 Sensitivity of Parameters

We study the effect of regularization parameter $\lambda$ and latent factor number $K$.

#### Regularization

The regularization parameters $\lambda_U$ and $\lambda_V$ affect the performance of our algorithm. We use a single parameter $\lambda$ for both the parameters. We found the impact of $\lambda$ is different under different spamming strategies. In Random and Semi-random strategies, the optimal $\lambda$ is smaller than that in the Optimistic and Pessimistic strategies. As an example, we show the impact of $\lambda$ under Optimistic and Semi-Random strategies in Figure 5.2. The reason for this phenomenon might be that in the Random and Semi-random strategies, a small $\lambda$ is needed to allow the prediction model to fit the normal users' rating better. While in the Optimistic or Pessimistic strategies, a coarse fitting is needed so that the spam users' ratings will not be fitted. We use $\lambda = 10.0$ in Optimistic and Pessimistic scenarios and $\lambda = 1.0$ in Random and Semi-random cases. Our suggestion for choosing $\lambda$ in real-life scenario is that it should be set to a relatively large value.

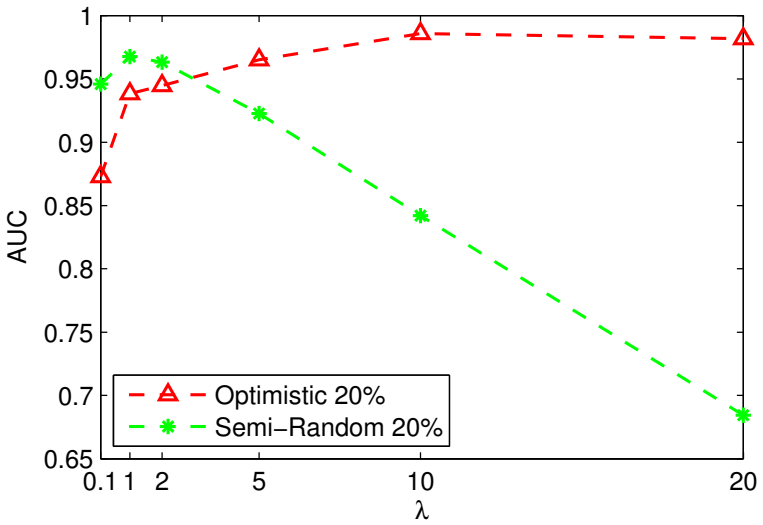



Figure 5.2: Impact of $\lambda$

**Latent Factor Number**

The impact of the latent factor number $K$ is shown in Figure 5.3. As we can see, the effect of $K$ is negligible and we choose $K = 10$ in all experiments.

## 5.5 Summary

Spammers are a serious problem in today's online rating systems. To identify these spam users, a unified framework for reputation estimation is proposed. Many existing prediction models can be readily plugged-in and employed to estimate reputations. We have shown how previously proposed methods can be captured as special cases of our framework. A MF-based reputation estimation method is developed based on the framework. It relaxes the assumption made

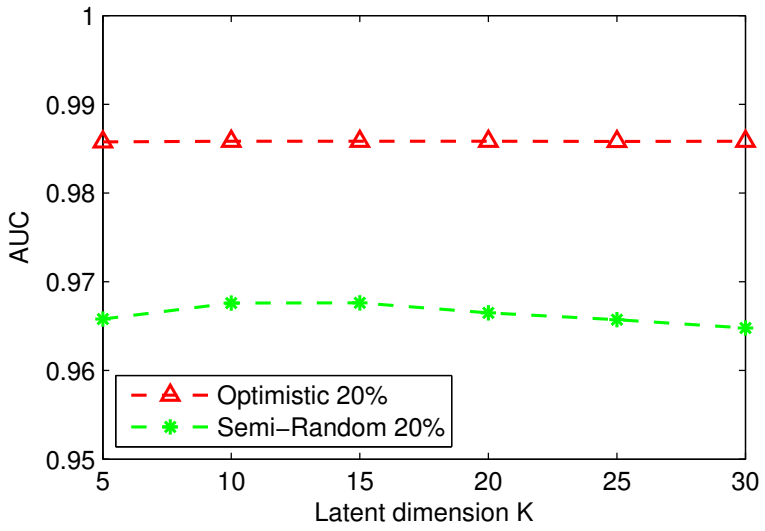



Figure 5.3: Impact of Latent Factor Number $K$

by previous methods, which might not be suitable for all online rating systems. To study the properties of our methods, we conduct a series of carefully designed experiments. Empirically we show the advantageous performance of our method.

□ **End of chapter.**

# Chapter 6

# Combine Ratings with Reviews

## 6.1 Problem and Motivation

Although recommender systems employed in industry seem to perform well in practice, there are some deficiencies with existing approaches. The first problem confronted with most recommender system is their inability to deal with so called *cold-start* problem [118]. When a new user joins a recommender system, there is little data available for the system to learn the preferences of the user accurately. Without an accurate representation of the user, the system cannot make recommendations confidently. Similarly, the systems defer the recommendations for newly included items as well. The cold-start problem leads to poor experience for new users and also when recommending new items. In real-life recommender systems, the cold-start problem is a severe problem. Shown in Figure 6.1 are the statistics of 5 categories in Amazon datasets [88]. Across the 5 listed datasets, over 80% of items have few ratings (less than 10). While at the same time, over 70% of items have review text at length (over 30 words). The ratings alone are inadequate to learn the preferences accurately. Review comments complement the ratings by providing rich knowledge of the items and preferences of the users. Harnessing the information embedded in the review text is the key to successful recommendation in such scenarios.

Another drawback of existing recommender systems is their poor

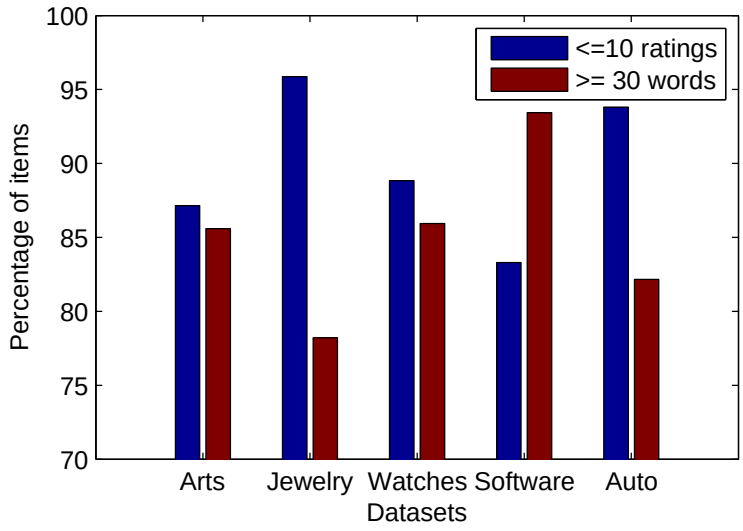



Figure 6.1: Percentage of items having less than 10 ratings and more than 30 words in various Amazon datasets

interpretability, making further understanding of users' preference as well as items' properties impossible. For example in matrix-factorization [114] based methods, we learn two latent feature matrices corresponding to users' latent features and items' latent features. The dot product between a user's and an item's feature vector is used to predict the rating that the user would assign to the item. It is challenging to associate these real valued features with conceivable physical meanings. We know that a user might like an item due to a particular latent feature since they both have a large positive (or negative) value on that feature. But we have no clue of the feature's physical meaning. Does it mean that the user is fond of Sci-Fi and the movie belongs to Sci-Fi genre? Or is it that the user loves the leading actor of the movie? We do not know. In fact, it is possible that each feature corresponds to a combination of the human interpretable features, rendering the feature interpretation problem more difficult.

Both of the above problems can be solved or at least alleviated

by combining *content-based filtering* and *collaborative filtering*. In collaborative filtering, we make predictions on a user's preferences over items based on all users' past ratings. Collaborative filtering rooted in the keen observation that users who shared similar preferences in the past tend to rate similarly in the future. A collaborative filtering model uses only the past rating information and does not take the contents of the items into consideration. On the other hand, content-based filtering approaches the recommendation problem by analyzing the content of the items and matches it with the preference of a user.

In a recommender system, apart from an integer score, users are often allowed to write text reviews about the item to complement the rating. The review text contains a source of rich information *explaining* the reason why the user assigns such a rating to the item. These reviews provide text contents of the items, which can be leveraged to alleviate cold-start problem when the ratings are sparse. This is because the information embedded in the text review is much richer that an integer rating. When we have few ratings, it is nearly impossible to learn an accurate feature of the concerned user/item. However, the text review might allow us to better estimate the features. To solve the interpretation problem, we align latent topic spaces with the rating spaces. Each latent dimension is tagged with a word cloud, explaining the physical meaning of the dimension. For example, when we see that a user and a movie have large positive value on the third feature, which has text label "thriller, sci-fi, nolan", we know that this user likes the science fiction thriller movie directed by Christopher Nolan.

Interpretability and the cold-start problem are not two isolated problems. Learning an interpretable model could help alleviate the cold-start problem [3, 73]. We can leverage prior knowledge of items and suggest completely "cold" items with confidence. For example, if we know that a user assigns high scores for the topic tagged with "fantasy, adventure, peter, jackson", a recommender

system can confidently recommend "The Hobbit: There and Back Again" (a fantasy adventure movie directed by Peter Jackson) to the user even if this movie is not being shown yet.

Due to the advantage of taking the review text into consideration, there are a few efforts [4, 35, 88, 93] explored the combination of content-based filtering and collaborative filtering. In the early work [93], the authors cast the content-based filtering as a classification problem, using which they filled out some of the unobserved user item rating matrix and apply collaborative filtering methods on this denser matrix. The authors of [4] cast the recommendation problem as an ordinal regression problem and applied a combination of kernels to handle the side information. In [35], the authors found that there were a few aspects that affect how users rate items. They harnessed the information embedded in the review text to learn how a user weights these aspects and how an item distributes on these aspects. However, their method required human annotators with expert domain knowledge to pre-define these aspects rather than learning them automatically from the reviews.

In the recent work [88], the authors proposed the Hidden Factors and Hidden Topics (HFT) model, which learnt a Latent Dirichlet Allocation (LDA) [11] model for items using the review text and a matrix factorization model to fit the ratings. To bridge the gap between the stochastic vector obtained from LDA and the real-valued vector in MF model, the authors proposed a transformation to link the two. Their method demonstrated significant improvement over baseline methods that use ratings or reviews alone. However, the transformation function they employ, the exponential function, fixed the relationship between latent vector in MF and the topic distribution. Although a parameter is employed to maintain a more flexible relationship, it is still difficult to ensure that this transformation is correctly scaled.

In [54, 120, 125], the authors also considered the interpretable aspects to make better predictions. However, their approaches dif-

fer from ours in that either they had *explicit* ratings per aspect (i.e., multiple ratings on prescribed aspects per user item pair), or these aspects were inferred from other context than review text. In another line of research called sentiment analysis [47, 71], a positive or negative label rather than an integer score is learnt for a short text. Our work also differs from [95], which recommends personalized reviews. In [127], the authors proposed Collaborative Topic Regression (CTR) to suggest scientific articles to potential readers. Later work [107] extended it to take the social network among users into consideration. As pointed out in [88], the latent dimensions they discovered were not necessarily correlated with ratings.

In this chapter, we develop an approach that exploits the combined power of both ratings and reviews and solve the above problems. Our contributions are three-fold. First, we propose a novel method to combine content-based filtering seamlessly with collaborative filtering, modeling the reviews and ratings simultaneously. Secondly, we derive an efficient collapsed Gibbs sampling method to learn the model. Thirdly, we demonstrate our model's advantage in prediction accuracy compared with previous work, especially under the cold-start setting, on large real-life datasets with millions of users and items. We also show the interpretability of the model using a few examples.

## 6.2 Ratings Meet Reviews

Our model, titled "Ratings Meet Reviews" (RMR) [67], is a probabilistic generative model that combines a topic model seamlessly with a rating model. We describe it as follows.

### 6.2.1 Model and Notations

Suppose there are $N$ users $\mathcal{U} = \{u_1, u_2, \cdots, u_N\}$, $M$ items $\mathcal{V} = \{v_1, v_2, \cdots, v_M\}$, a set of observed indices $Q = \{i, j\}$, where $\{u_i, v_j\} \in$

$\mathcal{U} \times \mathcal{V}$ defines the observed ratings $\mathcal{X} = \{x_{i,j}\}$, each of which is optionally associated with a review $r_{i,j} = \{w|w \subset V\}$ of length $L_{i,j}$, where $V$ is the set of vocabulary used in the review text. Alternatively, let $\mathcal{U}_j$ denote the indices of users who have rated item $v_j$. Let $K$ denote the number of topics.

RMR operates on items[1], which has an intrinsic distribution $\theta$ on topics. This distribution describes the proportion that the item belongs to each topic.

We present the generative process below:

1. For each user $u \in \mathcal{U}$:
   (a) For each latent topic dimension $k \in [1, K]$:
      i. Draw $\mu_{u,k} \sim$ Gaussian$(\mu_0, \sigma_0^2)$
2. For each latent topic dimension $k \in [1, K]$:
   (a) Draw $\psi_k \sim$ Dirichlet$(\beta)$
3. For each item $v \in \mathcal{V}$:
   (a) Draw topic mixture proportion $\theta_v \sim$ Dirichlet$(\alpha)$
   (b) For each description word $w_{v,n}$:
      i. Draw topic assignment $z_{v,n} \sim$ Multinomial$(\theta_v)$
      ii. Draw word $w_{v,n} \sim$ Multinomial$(\psi_{z_{v,n}})$
   (c) For each observed rating assigned by $u$ to $v$:
      i. Draw topic assignment $f_{v,u} \sim$ Multinomial$(\theta_v)$
      ii. Draw the rating $x_{v,u} \sim$ Gaussian$(\mu_{u,f_{v,u}}, \sigma^2)$.

From the generative process, we can identify that the text reviews are generated similarly as the LDA model. We use a mixture of Gaussian rather than matrix factorization based methods [88, 127]

[1]RMR is symmetrical in that the topic distribution $\theta$ can also be user specific. We found, however, that item specific $\theta$ performs better in practice.

to model the ratings. These user-topic specific Gaussian distributions have clear interpretations. They describe how a user values the aspects denoted by each latent topic. The item is modeled as a distribution of topics, which together with the user-topic specific Gaussian distributions determine how a user would rate the item. The ratings and review text are connected by the same item topic distribution $\theta$. The more a user talks about certain aspects concerning an item, the higher the distribution will be on these topics, which in turn affects the rating that the user would assign to the item.

We choose to model ratings using mixture of Gaussians for two reasons. First, we can avoid the difficult choice of the transformation function employed in [88]. As discussed above, the transformation function is restrictive and the scaling parameter is non-trivial to select. Secondly, we can retain the interpretability of the topics with no compromise. The interpretability of the latent dimensions is an important factor to solve the cold start problem. Take book recommendation for example, when a user showed strong interest in dimension with high probability on words "da vinci code dan brown". We can confidently recommend Dan Brown's new book "Inferno". We are able to associate the latent dimensions with the *prior* knowledge (for example, Meta data) that is available *without* ratings or reviews.

In our model, we collect all the text reviews submitted for an item into one document and take this document as the text review for the item. Note that in the original data, there is a correspondence between the rating value and the text review. We lose such information in RMR. By dropping this correspondence information, we can develop the more compact model RMR. The previous methods HFT and CTR also choose to model in such a way to reduce the model complexity.

Given the generative process, the probability of observing the review text and the ratings given the model parameters $\Theta = \{\theta, \psi, \mu\}$

is

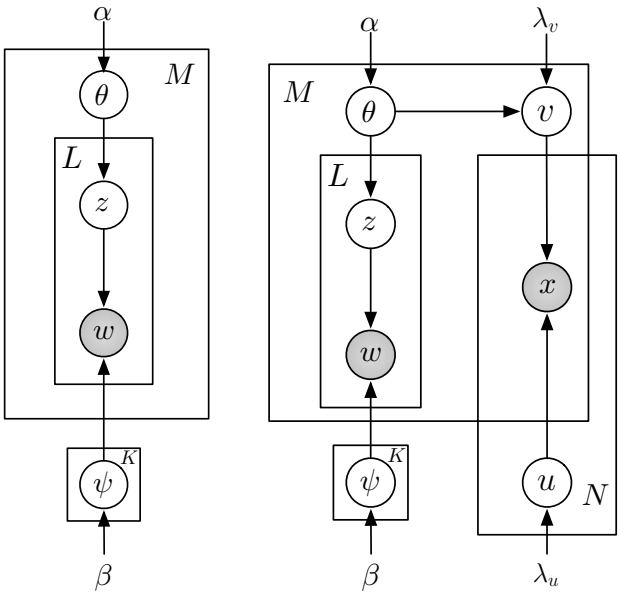
$$P(\mathbf{w}, \mathbf{x}|\Theta; \alpha, \beta, \mu_0, \sigma_0^2, \sigma^2) = \prod_{j=1}^{M} P(\theta_j|\alpha) \prod_{i \in \mathcal{U}_j} \left( \sum_{f=1}^{K} P(f|\theta_j) P(x_{i,j}|\mu_{i,f}, \sigma^2) \right) \left( \prod_{l=1}^{L_{i,j}} \sum_{z=1}^{K} P(z|\theta_j) P(w_l|\psi_z) \right) \left( \prod_{i=1}^{N} \prod_{k=1}^{K} P(\mu_{i,k}|\mu_0, \sigma_0^2) \right) \left( \prod_{k=1}^{K} P(\psi_k|\beta) \right). \quad (6.1)$$

If we take the log of Eq. (6.1), we get the log-likelihood of model parameters. However, because of the summation inside the log, direct optimization is not feasible. We subsequently develop an efficient collapsed Gibbs sampling method to learn the model parameters in Section 6.2.3. We now take a deeper look at RMR and compare it with HFT and CTR.

### 6.2.2 Comparison with HFT and CTR

Shown in Figure 6.2 and Figure 6.3 are the graphical models of RMR and several related work. As is clear from the figure, the left parts of CTR, HFT and RMR resemble LDA, which was originally proposed by David Blei et al. to learn the latent topics in a corpus of documents (items in our setting) in an unsupervised manner. The LDA algorithm assumes there are $K$ latent topics in the corpus, which are $K$ multinomial distributions over the vocabulary. Each document in the corpus is a mixture of these topics. A document specific topic distribution over the $K$ latent topics, $\theta_j$ with Dirichlet prior $\alpha$, governs how much weight each topic takes in document $j$. This $\theta_j$ is a length-$K$ stochastic vector with non-negative entries that sums up to 1.

Both HFT and CTR adopt the matrix factorization method to model the ratings. Arrange users' ratings on items in a partially observed matrix $X \in \mathbb{R}^{N \times M}$. The matrix factorization model assumes



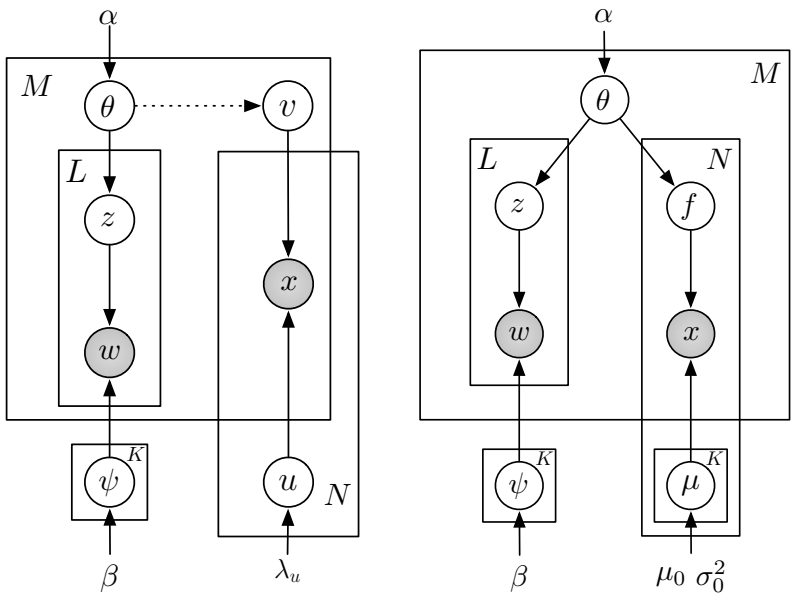

(a) LDA (b) CTR

Figure 6.2: Graphical Models of LDA and CTR

that $X$ has a low rank structure and thus can be decomposed in to the product of two matrices $U^T V$, where both $U$ and $V$ are of rank $K \ll \min(M, N)$. The columns of $U$ and $V$ can be interpreted as the latent features of users and items respectively. The dot product between a user's feature vector and an item's feature vector approximates the rating that the user would assign to the item. To regularize the value that the latent features can assume, often zero-mean isotropic Gaussian priors are placed on both the user and item latent features. The objective function of a matrix factorization model can be formulated as follows,

$$\mathcal{L} = \sum_{i,j \in Q} (U_i^T V_j - X_{i,j})^2 + \lambda_U \|U\|_F + \lambda_V \|V\|_F, \tag{6.2}$$

in which the first term is the difference between observed and prediction and the rest are regularization terms.




(a) HFT (b) RMR

Figure 6.3: Graphical Models of HFT and RMR

Clearly, there is a discrepancy between the item topic distribution $\theta_j$ in LDA and the item feature vector $V_j$ in MF model. The former is a distribution, which is all positive and sums up to 1, while the later can assume any real value. Both HFT and CTR try to align the item features with the item topic distribution and hence the rich text review can be exploited to better model the item features. The main difference between HFT and CTR model lies in the way they align the topic distribution $\theta_j$ and the item feature $V_j$. In CTR, the item feature $V_j$ is assumed to be a Gaussian random variable with mean $\theta_j$ and precision $c$. In other words, the $\theta_j$ is taken as the *default* value of $V_j$, but the later can adapt to match the ratings. If an item receives a lot of ratings, it is possible that $V_j$ differs from $\theta_j$ significantly. The interpretation of the latent topics in such case might be distorted. CTR model was proposed to recommend scien-

tific articles to potential readers, which is a one-class collaborative filtering task [48, 102]. It adopt the strategy to set different $c$ for observed ratings ($X_{i,j} = 1$) and unobserved ratings ($X_{i,j} = 0$). In our setting, we try to match only the observed ratings with the given rating scales (for example, $1$ to $5$). On the other hand, HFT adopts a fixed transformation function to map one-to-one between $\theta_j$ and $V_j$. In Figure 6.3(a), we use a dashed line to represent this relationship. The HFT model is effectively a matrix factorization model with the item feature regularization replaced by the corpus likelihood. This fixed transformation function is difficult to select and restrictive in modeling capability.

RMR adopt a mixture of Gaussian to model the ratings. The mixture proportion is assumed to have the same distribution as the topic distribution. Thus when there are few ratings for an item, the text review can still allow us to learn the topic distribution $\theta$ accurately. We avoid the difficult choice of the transformation function in HFT and retain the interpretability of the latent topics.

### 6.2.3 Collapsed Gibbs Sampler

To develop a Gibbs sampler for RMR, we need to specify the conditional probability of the hidden variables $z$ and $f$, which are the hidden topics associated with the observed words $w$ and observed ratings $x$, $P(\mathbf{z}, \mathbf{f}|\mathbf{w}, \mathbf{x})$. This conditional probability does not have a closed form and is difficult to sample directly. The collapsed Gibbs sampler runs a Markov chain that uses the full conditional in order to simulate it. In our case, we need to specify the following two conditional probabilities

$$P(z_i = j|\mathbf{z}_{\neg \mathbf{i}}, \mathbf{w}, \mathbf{f}, \mathbf{x}), \tag{6.3}$$

$$P(f_i = j|\mathbf{z}, \mathbf{w}, \mathbf{f}_{\neg \mathbf{i}}, \mathbf{x}). \tag{6.4}$$

We will briefly derive the expression for the probability in Eq. (6.4).

Using Bayes' theorem and the conditional independence, we obtain

$$\begin{aligned} & P(f_i = j|\mathbf{z}, \mathbf{w}, \mathbf{f}_{\neg \mathbf{i}}, \mathbf{x}) \\ \propto\ & P(x_i|f_i = j, \mathbf{f}_{\neg \mathbf{i}}, \mathbf{x}_{\neg \mathbf{i}}) P(f_i = j|\mathbf{f}_{\neg \mathbf{i}}, \mathbf{z}) \end{aligned} \tag{6.5}$$

Now we will derive the expression for the two terms in Eq. (6.5). Let the index $i$ denote the rating assigned by user $u$ to item $v$.

$$\begin{aligned} & P(x_i|f_i = j, \mathbf{f}_{\neg \mathbf{i}}, \mathbf{x}_{\neg \mathbf{i}}) \\ =\ & \int_{\mu_{u,j}} P(x_i, |\mu_{u,j}, f_i = j) P(\mu_{u,j}|\mathbf{f}_{\neg \mathbf{i}}, \mathbf{x}_{\neg \mathbf{i}}) \mathrm{d}\mu_{u,j} \end{aligned} \tag{6.6}$$

The second term in Eq. (6.6) is a Gaussian distribution, because

$$P(\mu_{u,j}|\mathbf{f}_{\neg \mathbf{i}}, \mathbf{x}_{\neg \mathbf{i}}) \propto P(\mathbf{x}_{\neg \mathbf{i}}|\mu_{u,j}, \mathbf{f}_{\neg \mathbf{i}}) P(\mu_{u,j}). \tag{6.7}$$

Since $P(\mu_{u,j})$ is Gaussian $\mathcal{N}(\mu_0, \sigma_0^2)$ and conjugate to $P(\mathbf{x}_{\neg \mathbf{i}}|\mu_{u,j}, \mathbf{f}_{\neg \mathbf{i}})$, the posterior distribution $P(\mu_{u,j}|\mathbf{f}_{\neg \mathbf{i}}, \mathbf{x}_{\neg \mathbf{i}})$ will be Gaussian $\mathcal{N}(\mu_i, \sigma_i^2)$, where

$$\sigma_i^2 = \frac{1}{\sigma_0^2} + \frac{|x^j_{u,(\cdot)}|}{\sigma^2}, \tag{6.8}$$

$$\mu_i = (\sigma_i^2)^{-1} (\frac{\mu_0}{\sigma_0^2} + \frac{\sum_m x^j_{u,m}}{\sigma^2}). \tag{6.9}$$

The predictive posterior in Eq. (6.6) is Gaussian $\mathcal{N}(\mu_i, \sigma_i^2 + \sigma_0^2)$ [96]. Similarly, the expression for the second term in Eq. (6.5) is

$$P(f_i = j|\mathbf{f}_{\neg \mathbf{i}}, \mathbf{z}) \propto \int_{\theta_v} P(f_i = j|\theta_v) P(\theta_v|\mathbf{f}_{\neg \mathbf{i}}, \mathbf{z}) \mathrm{d}\theta_v, \tag{6.10}$$

of which the second term is

$$P(\theta_v|\mathbf{f}_{\neg \mathbf{i}}, \mathbf{z}) \propto P(\mathbf{f}_{\neg \mathbf{i}}|\theta_v) P(\mathbf{z}|\theta_v) P(\theta_v). \tag{6.11}$$

Again, since $P(\theta_v)$ is Dirichlet($\alpha$) and conjugate to $P(\mathbf{f}_{\neg \mathbf{i}}|\theta_v)$ and $P(\mathbf{z}|\theta_v)$, the posterior is also a Dirichlet distribution and the posterior predictive of Eq. (6.10) is

$$P(f_i = j|\mathbf{z}, \mathbf{f}_{\neg \mathbf{i}}) = \frac{n^v_{f,\neg i,j} + n^v_{z,j} + \alpha}{n^v_{f,\neg i,(\cdot)} + n^v_{z,(\cdot)} + K\alpha}. \tag{6.12}$$

| Symbol | Description |
|---|---|
| $\lvert x^j_{u,(\cdot)}\rvert$ | # of ratings assigned by $u$ with topic $j$ |
| $x^j_{u,m}$ | value of rating assigned by $u$ to $m$ with topic $j$ |
| $n^v_{z,j}$ | # of $z$ of item $v$ assigned to topic $j$ |
| $n^v_{z,(\cdot)}$ | total # of review words $v$ received |
| $n^v_{f,j}$ | # of $f$ of item $v$ assigned to topic $j$ |
| $n^v_{f,(\cdot)}$ | total # of ratings item $v$ received |
| $n^{w_i}_j$ | # of word $w_i$ assigned to topic $j$ |
| $n^{(\cdot)}_j$ | total # of words assigned to topic $j$ |

Table 6.1: Notations

Combine the result in Eq. (6.6) and Eq. (6.10), we get the expression for the full conditional for the first probability in Eq. (6.4)

$$\begin{aligned} &P(f_i = j|\mathbf{z}, \mathbf{w}, \mathbf{f}_{\neg \mathbf{i}}, \mathbf{x}) \\ &\propto \mathcal{N}(x_i|\mu_i, \sigma_i^2 + \sigma_0^2) \frac{n^v_{f,\neg i,j} + n^v_{z,j} + \alpha}{n^v_{f,\neg i,(\cdot)} + n^v_{z,(\cdot)} + K\alpha}, \end{aligned} \tag{6.13}$$

and by employing a similar procedure, we get the expression for the first probability in Eq. (6.4)

$$\begin{aligned} &P(z_i = j|\mathbf{z}_{\neg \mathbf{i}}, \mathbf{w}, \mathbf{f}, \mathbf{x}) \\ &\propto \frac{n^{w_i}_{\neg i,j} + \beta}{n^{(\cdot)}_{\neg i,j} + |V|\beta} \frac{n^v_{z,\neg i,j} + n^v_{f,j} + \alpha}{n^v_{z,\neg i,(\cdot)} + n^v_{f,(\cdot)} + K\alpha}. \end{aligned} \tag{6.14}$$

We summarize the notations used in the derivation process in Table 6.1. Note that we omit the $\neg i$ subscription in some of the notations to save space. With this notation, it means that when counting the respective values, we exclude current word $w_i$ or current rating $x_i$.

#### Readout Parameters

Once the sampling process is finished, we can readily readout the model parameters

$$\theta_{v,j} = \frac{n^v_{z,j} + n^v_{f,j} + \alpha}{n^v_{z,(\cdot)} + n^v_{z,(\cdot)} + K\alpha}, \quad \psi_{j,w_i} = \frac{n^{w_i}_j + \beta}{n^{(\cdot)}_j + |V|\beta}, \tag{6.15}$$

and

$$\mu_{u,j} = \left(\frac{1}{\sigma_0^2} + \frac{|x^j_{u,(\cdot)}|}{\sigma^2}\right)^{-1} \left(\frac{\mu_0}{\sigma_0^2} + \frac{\sum_m x^j_{u,m}}{\sigma^2}\right). \tag{6.16}$$

The notations denote the same meaning as is in Table 6.1, except that the counters now count all effective samples.

#### Time and Space Complexity of the Sampler

Our collapsed Gibbs sampler mainly use the following counters to keep track of current states: counter $n^v_{z,j}$ of size $M \times K$, counter $n^v_{f,j}$ of size $M \times K$, counter $n^w_{\neg i,j}$ of size $V \times K$, counter $\sum_m x^j_{n,m}$ of size $N \times K$ and counter $|x^j_{n,(\cdot)}|$ of size $N \times K$. Other than the above listed counters, there are summation counters that are one order smaller than the above counters and thus negligible. The total space complexity is $O((M + N + V) \times K)$.

To sample $z$ or $f$ conditioned on everything else, we only need to calculate the conditional probabilities in Eq. (6.4). This operation requires $O(K)$ operations. Given a fixed $K$, our sampler scales *linearly* with the length of the observed review text and the number received ratings. Usually the number of latent topics $K$ is small, making our sampler scales up well. Note that training a RMR model is faster than training a HFT model [88]. This is because the later one requires an additional step to learn the feature vectors for all the users.

### 6.2.4 Prediction

In order to make a prediction that user $u$ assigns to item $v$, we compute the expected value

$$x_{u,v} = \sum_k \theta_{v,k}\mu_{v,k}. \tag{6.17}$$

In practice, we compute the empirical global mean $g$, user bias $b_u$ and item bias $b_v$ for all users and all items from the training set. We feed the $x'_{u,v} = x_{u,v} - g - b_u - b_v$ to the sampler and when making predictions, add $g$, $b_u$, and $b_v$ back.

## 6.3 Experiments

We conduct an empirical study of RMR and various baseline models to show the following facts:

1. Our model leads to significant improvement on prediction accuracy across various categories of items over several strong baseline models.
2. Our model learns latent topic dimensions that are clearly interpretable.
3. Our model performs better in datasets that are extremely sparse, which resembles the cold-start settings.

### 6.3.1 Dataset

We use the Amazon Review dataset collected by [88]. This dataset is a collection of 27 datasets corresponding to various types of items that are available on Amazon[2]. This is the largest rating dataset with text reviews publicly available, to the best of our knowledge. We

[2]The statistics of category Baby we calculated differs from the description provided on the webpage and we exclude it from consideration.

| Dataset | #users | #items | #review | #words | words/review | reviews/item |
|---|---|---|---|---|---|---|
| Arts | 24,071 | 4,211 | 27,980 | 2,006,874 | 71.73 | 6.64 |
| Jewelry | 40,594 | 18,794 | 58,621 | 3,100,948 | 52.90 | 3.12 |
| Industrial Scientific | 29,590 | 22,622 | 13,7042 | 6,920,151 | 50.50 | 6.06 |
| Watches | 62,041 | 10,318 | 68,356 | 5,436,671 | 79.53 | 6.62 |
| Cell Phones and Accessories | 68,041 | 7,438 | 78,930 | 7,567,961 | 95.88 | 10.61 |
| Musical Instruments | 67,007 | 14,182 | 85,405 | 7,442,294 | 87.14 | 6.02 |
| Software | 68,464 | 11,234 | 95,084 | 11,012,882 | 115.82 | 8.46 |
| Gourmet Foods | 112,544 | 23,476 | 154,635 | 10,542,984 | 68.18 | 6.59 |
| Office Products | 110,472 | 14,224 | 138,084 | 11,206,338 | 81.16 | 9.71 |
| Automotive | 133,256 | 47,577 | 188,728 | 13,249,641 | 70.21 | 3.97 |
| Patio | 166,832 | 19,531 | 206,250 | 17,290,881 | 83.83 | 10.56 |
| Pet Supplies | 160,496 | 17,523 | 217,170 | 18,684,153 | 86.03 | 12.39 |
| Beauty | 167,725 | 29,004 | 252,056 | 17,889,577 | 70.97 | 8.69 |
| Shoes | 73,590 | 48,410 | 389,877 | 23,604,059 | 60.54 | 8.05 |
| Kindle Store | 116,191 | 4,372 | 160,793 | 21,533,201 | 133.92 | 36.78 |
| Clothing and Accessories | 128,794 | 66,370 | 581,933 | 34,267,151 | 58.89 | 8.77 |
| Health | 311,636 | 39,539 | 428,781 | 33,277,423 | 77.61 | 10.84 |
| Toys and Games | 290,713 | 53,600 | 435,996 | 35,034,001 | 80.35 | 8.13 |
| Tools and Home Improvement | 283,514 | 51,004 | 409,499 | 34,591,409 | 84.47 | 8.03 |
| Sports and Outdoors | 329,232 | 68,293 | 510,991 | 38,898,738 | 76.12 | 7.48 |
| Video Games | 228,570 | 21,025 | 463,669 | 55,532,148 | 119.77 | 22.05 |
| Home and Kitchen | 644,509 | 79,006 | 991,794 | 81,923,017 | 82.60 | 12.55 |
| Amazon Instant Video | 312,930 | 22,204 | 717,651 | 88,958,349 | 123.96 | 32.32 |
| Electronics | 811,034 | 82,067 | 1,241,778 | 124,064,510 | 99.91 | 15.13 |
| Music | 1,134,684 | 556,814 | 6,396,350 | 774,791,468 | 121.13 | 11.49 |
| Movies and TV | 1,224,267 | 212,836 | 7,850,072 | 997,261,969 | 127.04 | 36.88 |
| Books | 2,588,991 | 929,264 | 12,886,488 | 1,613,603,531 | 125.22 | 13.87 |
| All categories | 6,643,669 | 2,441,053 | 34,686,880 | 4,053,795,667 | 116.87 | 14.21 |

Table 6.2: Statistics of the datasets

show the statistics of the datasets in Table 6.2. Refer to Table 6.2; there are two facts that are evident immediately. First, the datasets are extremely sparse. The sparseness would clearly deteriorate the performance of most existing recommender systems that only model the ratings. Secondly, a review contains 116.87 words on average across all categories. As will be apparent in the results shown later, these review texts are key to model the ratings accurately.

### 6.3.2 Baseline Methods

We compare our model with four baseline models MF, LDAMF, CTR and HFT.

- **MF** This is the standard matrix factorization model as is described in [114]. We ignore the review texts completely and model the ratings only. This is typically a very strong baseline model in collaborative filtering [59, 108].
- **LDAMF** This baseline model is proposed in [88]. This baseline model tries to harness the information in the review text by fitting an LDA model on the review text and then treat the learnt topic distribution on items (or users) as the latent factors in matrix factorization models. By holding the latent factors for items (or users) fixed, the latent factors for users (or items) are learnt by gradient descent methods.
- **CTR** This is the state-of-the-art method that recommends scientific articles to potential interested readers [127]. The CTR model solves the one-class collaborative filtering problem by using different precision parameter $c$. In our setting, we use it to match the observed integer ratings using the same precision $c$. We employ LDA-C [11] to pre-train the model. Note that CTR utilizes both ratings and reviews information.
- **HFT** This is the state-of-the-art method that combines reviews with ratings [88]. HFT models the ratings using a matrix fac-

torization model with an exponential transformation function to link the stochastic topic distribution in modeling the review text and the latent vector in modeling the ratings. The topic distribution can be modeled on either users or items. On most datasets, the item specific topic distribution produces more accurate predictions. We report the results whichever are more accurate.

### 6.3.3 Evaluation

We use Mean Squared Error (MSE) to evaluate various models. For each of the dataset, we randomly select 80% as training set up to 2 million reviews. The remaining reviews are split evenly into validation set and testing set. The initial latent variables $z$ and $f$ are uniformly randomly assigned. We run 2500 iterations with a thinning of 50 iterations to get samples and MSE readout. We report the MSE of the testing set that has the lowest MSE on the validation set. The training of the baseline methods MF, LDAMF, CTR and HFT follow the same routine described in [88]. We use $K = 5$ for all models. We set hyperparameters[3] $\alpha = 0.1$, $\beta = 0.02$, $\mu_0 = 0$, $\sigma_0^2 = 1$ and we use the empirical variance of $x$ as $\sigma^2$. In practice, the time required to train the RMR model is about half of the time that is spent on training the HFT model on the same machine.

### 6.3.4 Rating Prediction

Shown in Table 6.3 are the MSE results. The best MSE of each dataset is in bold. We listed the performance of various models on the datasets and the average improvement. The standard deviations of MSE results are shown in parenthesis. Out of the 27 datasets, RMR performs the best on 19 datasets among all considered methods.

[3]We searched through the parameters linearly and reported hyperparameters which performed the best.

Compared with matrix factorization (MF column in Table 6.3), RMR performs better on 26 out of the 27 datasets with an average improvement on MSE of nearly 8%. Matrix factorization method usually performs well in practice [59, 108] and is a strong baseline method. However, as shown in our case, in datasets that are extremely sparse, MF is unable to learn an accurate representation of users/items and thus under-performs other methods which take the review text into consideration. However, in the datasets such as Music, Movies and TV and Books, which are relatively denser compared with other datasets; the MF method still performs very well.

The baseline method LDAMF, which was proposed as a baseline method in [88], is probably the simplest model that combines review text and ratings. This baseline method takes the item topic distribution produced by LDA as the feature vectors for the items and then learns the user feature vectors by fitting the observed ratings with item features fixed. The feature vectors of items are learnt using *only* the reviews, which might be sub-optimal to fit the rating data. The expressiveness is thus restricted and we think this restriction caused the nearly 8% improvement produced by RMR.

Compared with CTR, which take the full advantage of the combined information of both the reviews and ratings, our proposed model still leads to an average improvement of 3.28% and performs better on 25 out of the 27 datasets. Similar to LDAMF, CTR takes the item topic distribution produced by LDA as the initial item features. However unlike LDAMF, during the training period, CTR alters both the user features and item features to fit the ratings. The regularization parameter $\lambda_V$ controls how much the item features can deviate from the item topic distribution vectors. It performs better due to the more flexible modeling capability. However, the CTR does not perform as well as RMR in the extremely sparse datasets such as Arts and Jewelry. We observe that during the experiment, CTR can learn a model that fits the data with a small training error. But the generalization of the learnt model to the unobserved rating

is not as good. Note that we report the performance of CTR on the test set by setting $\lambda_V$ and $\lambda_U$ to the value that gives best performance on validation set. So the issue of under-regularization is minimized. The performance of CTR on the relatively dense datasets is very competitive.

Compared with HFT, another recommendation method that takes review text into consideration, RMR is still able to improve the performance by 1.22% on average and performs better or equally well in 21 out of 27 datasets. As discussed in previous sections, we think the fixed one-to-one mapping between the item topic distribution and item feature vector impose restrictions of the expressiveness of HFT and allow RMR to out-perform it. Due to large size of the datasets, the improvements reported are significant at 1% level.

We consider the improvements of RMR over CTR (3.28%) and HFT (1.22%) significant because both of these two baselines are full-fledged models that take both the ratings and reviews into consideration. Also, these improvements are verified on 27 real-life datasets. In a real system where recommendation plays a central role, e.g. Amazon, Netflix, these improvements could lead to better revenue and profit.

### 6.3.5 Cold-start Setting

An interesting phenomenon we found in the results is that the improvement of RMR over the traditional collaborative filtering methods (matrix factorization) is more significant for datasets that are sparse. For classes such as Arts, Industrial Scientific, RMR show substantial improvement. In such cases, the number of ratings is too scarce to model the items and users adequately. The text in the review associated with the ratings comes as rescue, which allows our model to learn a more accurate topic distribution. Whereas for classes such as Music, Movies and Books, which are the largest 3 datasets with larger reviews per user and reviews per item, the tra-

| | a | b | c | d | e | Improvement of RMR versus | | |
|---|---|---|---|---|---|---|---|---|
| Dataset | MF | LDAMF | CTR | HFT | RMR | min(a,b) | c | d |
| Arts | 1.565 (0.04) | 1.575 (0.04) | 1.471 (0.04) | 1.390 (0.04) | **1.371 (0.04)** | 14.15% | 7.29% | 1.39% |
| Jewelry | 1.257 (0.03) | 1.279 (0.03) | 1.206 (0.03) | 1.177 (0.02) | **1.160 (0.02)** | 8.36% | 3.97% | 1.47% |
| Industrial Scientific | 0.461 (0.02) | 0.462 (0.02) | 0.382 (0.02) | **0.359 (0.02)** | 0.362 (0.02) | 27.35% | 5.52% | -0.83% |
| Watches | 1.535 (0.03) | 1.518 (0.03) | 1.491 (0.03) | 1.488 (0.03) | **1.458 (0.02)** | 4.12% | 2.26% | 2.06% |
| Cell Phones and Accessories | 2.230 (0.04) | 2.308 (0.04) | 2.177 (0.04) | 2.135 (0.03) | **2.085 (0.03)** | 6.95% | 4.41% | 2.40% |
| Musical Instruments | 1.506 (0.02) | 1.520 (0.02) | 1.422 (0.02) | 1.395 (0.02) | **1.374 (0.02)** | 9.61% | 3.49% | 1.53% |
| Software | 2.409 (0.02) | 2.214 (0.02) | 2.254 (0.02) | 2.219 (0.02) | **2.173 (0.02)** | 1.89% | 3.73% | 2.12% |
| Gourmet Foods | 1.515 (0.01) | 1.491 (0.01) | 1.482 (0.01) | **1.457 (0.01)** | 1.465 (0.01) | 1.77% | 1.16% | -0.55% |
| Office Products | 1.814 (0.01) | 1.796 (0.01) | 1.733 (0.01) | 1.669 (0.01) | **1.638 (0.01)** | 9.65% | 5.80% | 1.89% |
| Automotive | 1.570 (0.01) | 1.585 (0.01) | 1.492 (0.01) | 1.432 (0.01) | **1.403 (0.01)** | 11.90% | 6.34% | 2.07% |
| Patio | 1.771 (0.01) | 1.793 (0.01) | 1.720 (0.01) | 1.698 (0.01) | **1.669 (0.01)** | 6.11% | 3.06% | 1.74% |
| Pet Supplies | 1.700 (0.01) | 1.700 (0.01) | 1.613 (0.01) | 1.583 (0.01) | **1.562 (0.01)** | 8.83% | 3.27% | 1.34% |
| Beauty | 1.399 (0.01) | 1.414 (0.01) | 1.361 (0.01) | 1.358 (0.01) | **1.334 (0.01)** | 4.87% | 2.02% | 1.80% |
| Shoes | 0.305 (0.00) | 0.335 (0.00) | 0.271 (0.00) | **0.247 (0.00)** | 0.251 (0.00) | 21.51% | 7.97% | -1.59% |
| Kindle Store | 1.553 (0.01) | 1.561 (0.01) | 1.457 (0.01) | 1.437 (0.01) | **1.412 (0.01)** | 9.99% | 3.19% | 1.77% |
| Clothing and Accessories | 0.393 (0.00) | 0.406 (0.00) | 0.355 (0.00) | 0.349 (0.00) | **0.336 (0.00)** | 16.96% | 5.65% | 3.87% |
| Health | 1.615 (0.01) | 1.608 (0.01) | 1.552 (0.01) | 1.538 (0.01) | **1.512 (0.01)** | 6.35% | 2.65% | 1.72% |
| Toys and Games | 1.467 (0.01) | 1.395 (0.01) | 1.389 (0.01) | **1.370 (0.01)** | 1.372 (0.01) | 1.68% | 1.24% | -0.15% |
| Tools and Home Improvement | 1.600 (0.01) | 1.610 (0.01) | 1.513 (0.01) | 1.510 (0.01) | **1.491 (0.01)** | 7.31% | 1.48% | 1.27% |
| Sports and Outdoors | 1.219 (0.01) | 1.223 (0.01) | 1.150 (0.01) | 1.138 (0.01) | **1.129 (0.01)** | 7.97% | 1.86% | 0.80% |
| Video Games | 1.610 (0.01) | 1.608 (0.01) | 1.572 (0.01) | 1.528 (0.01) | **1.510 (0.01)** | 6.49% | 4.11% | 1.19% |
| Home and Kitchen | 1.628 (0.05) | 1.610 (0.05) | 1.577 (0.05) | 1.531 (0.04) | **1.501 (0.04)** | 7.26% | 5.06% | 2.00% |
| Amazon Instant Video | 1.330 (0.01) | 1.328 (0.01) | 1.291 (0.01) | **1.260 (0.01)** | 1.270 (0.01) | 4.57% | 1.65% | -0.79% |
| Electronics | 1.828 (0.00) | 1.823 (0.00) | 1.764 (0.00) | **1.722 (0.00)** | **1.722 (0.00)** | 5.87% | 2.44% | 0.00% |
| Music | **0.956 (0.00)** | 0.958 (0.00) | 0.959 (0.00) | 0.980 (0.00) | 0.959 (0.00) | -0.31% | 0.00% | 2.19% |
| Movies and TV | 1.119 (0.00) | 1.117 (0.00) | **1.114 (0.00)** | 1.119 (0.00) | 1.120 (0.00) | -0.27% | -0.54% | -0.09% |
| Books | 1.107 (0.00) | 1.109 (0.00) | **1.106 (0.00)** | 1.138 (0.00) | 1.113 (0.00) | -0.54% | -0.63% | 2.25% |
| Average on all datasets | | | | | | 7.79% | 3.28% | 1.22% |

Table 6.3: MSE results of various models

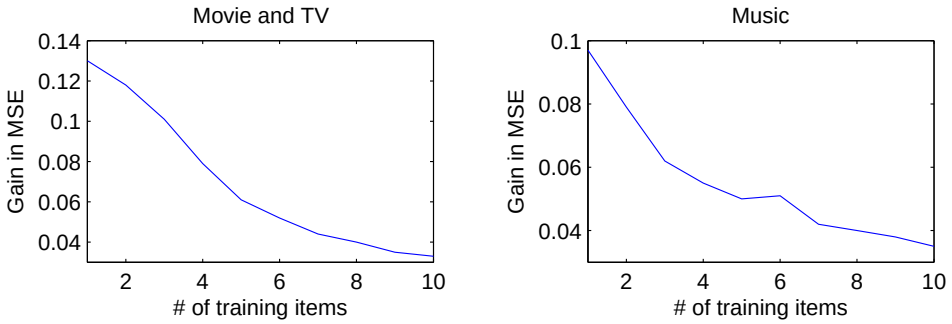



Figure 6.4: Gain in MSE for user with limited training data

ditional methods tend to produce accurate predictions. We further verify this finding by comparing the performance of RMR with MF on users with limited training data. Shown in Figure 6.4 is the gain of RMR compared with MF for users with limited training items. We show the result on two datasets due to space limit and the phenomenon repeats across all the datasets. As we can see, our model gains the most when the user has few training items. The performance gain starts to decrease with the number of training items available for each user. This further demonstrates that RMR is valuable for the cold-start settings.

### 6.3.6 Interpretability of Topics

Apart from being more accurate at prediction, another advantage of RMR is that it learns interpretable latent topics. We show two examples of the top words in each topic learnt in RMR in Table 6.4 and Table 6.5. Table 6.4 shows the top words for topics learnt with *software* dataset. Note that Roxio is software for burning DVDs and Quicken is personal financial software. Leopard and Tiger are the code name of Mac OS X and Parallels is a popular virtual machine on OS X. The fourth topic is about the company Microsoft and its products and the last topic is related to Linux. Table 6.5 shows the top words for topics learnt with Movie and TV dataset. The first

| roxio | quicken | leopard | office | suse |
|---|---|---|---|---|
| contacted | son | os | excel | accounts |
| perfect | pick | parallels | 2007 | 2004 |
| burning | given | apple | student | nav |
| dvds | spanish | turbo | activation | federal |
| care | starting | tiger | microsoft | symantec |

Table 6.4: Top words for topics in Software

| workout | season | batman | disney | godzilla |
|---|---|---|---|---|
| yoga | match | effects | christmas | hitchcock |
| workouts | episodes | alien | animation | kidman |
| videos | seasons | harry | kids | murder |
| exercises | vs | matrix | shrek | densel |
| cardio | episode | edition | animated | nicole |

Table 6.5: Top words for topics in Movie and TV

topic is dedicated to workout related videos. The second topic contains commonly used words to describe TV series. Batman, Matrix trilogy, Alien and Harry Potter are either science fiction, adventure or fantasy movies. Godzilla is a disaster thriller and Hitchcock is a famous director of psychological thrillers. Nicole Kidman is the leading actress of the classic thriller "Eyes Wide Shut".

Clearly these interpretable topics would help us understand items and users better. For items, the top topic words can be employed as extended tags attached to the item and may improve the prediction accuracy in a tag-aware recommender system [42]. We may also gain better understanding of items by analyzing the topic distribution similarities. For users, once obtaining the topic preferences, we can recommend "cold" items, which have few or no ratings, to the users with confidence. For example, if we know that a user tends to rate high for topic three and five in Table 6.5, we can confidently recommend the movie "Interstellar" (a Sci-Fi Thriller movie) even if this movie is not being shown yet. Our prior knowledge of items therefore can help alleviate the cold-start problem.

## 6.4 Summary

In this chapter, we propose a model that combines content-based filtering with collaborative filtering seamlessly. By exploiting the information in both ratings and reviews, we are able to improve the prediction accuracy significantly across various classes of datasets over existing strong baseline methods, especially under the cold-start settings where the data are extremely sparse. We develop an efficient collapsed Gibbs sampler for learning the model parameters. Our model also learns topics that are interpretable, enabling us to exploit prior knowledge to alleviate the cold start problem.

---

□ **End of chapter.**

# Chapter 7

# Conclusion

In this chapter, we summarize the main contributions made in this thesis and discuss several interesting future directions.

## 7.1 Summary

Recommender systems have become an indispensable tool for both the online service providers such as e-commerce website, music and video streaming providers and users of such services. On one hand, online service providers rely on recommender systems to attract users to stick with the service and enlarge the user base and expand business. On the other hand, users also benefit from recommender systems. However, there are problems confronted with recommender systems that deteriorate the user experiences. This thesis established models that improve the recommender systems from four perspectives.

In particular, in Chapter 3, we point out the problems with batch training algorithm for collaborative filtering. Batch training algorithms require the presents of all data before training and all data are needed during each training iteration. The sheer data size managed by a real life recommender system renders batch training algorithms impractical. Also, in practice, new users and new items join the system constantly and to include these new users and items,

batch training algorithms have to re-train the system. To solve this problem, we propose online learning algorithms that make update incrementally based on each rating point. The training speed is significantly faster and recommendation accuracy is not compromised. In addition, online learning algorithms can handle the new users and new items effortlessly.

In Chapter 4, we examine the implicit assumption made by most existing recommender systems, that the rating distribution of collected data and the unobserved data are the same. Using data evidence collected from Yahoo!Music, we show the assumption is unlikely to be true in real systems. Equipped with missing data theory, we prove that most existing recommender systems produce biased recommendations without the assumption. We take response pattern into consideration and develop Response Aware PMF. By incorporating a missing data model, Response Aware drops the incorrect assumption and makes unbiased recommendations.

In Chapter 5, we inspect the spam user problem in online rating systems. Spam users rate items with malicious purpose and exert negative effect on a recommender system by contaminating the rating data. We purpose to use reputation estimation system to keep track of users' reputations and identify spam users. We develop a unified framework in the setup of online rating systems that subsumes many previous methods as special cases. Leveraging the powerful matrix factorization based method, we instantiate the framework and present a matrix factorization based reputation estimation method. We demonstrate the outstanding discrimination ability of our method under various spamming strategies.

In Chapter 6, we combine content-based filtering and collaborative filtering and present the Ratings Meet Reviews model. Review comments are mostly discarded in an online rating system due to the difficulty in incorporating them into a collaborative filtering model. However, reviews contain rich information on how a user likes an item and provide reasons judging the ratings assigned by

the user. This information are especially valuable in the cold-start settings since reviews in general contain much richer information than an integer rating. Also, using techniques from topic modeling, we can tag the black-box collaborative filtering algorithms with interpretable words. This helps the recommender system providing reasons on why items are being recommended and helps users deciding whether to take further actions.

In summary, we solved four important problems that are faced with recommender systems and improved the recommender systems in various ways.

## 7.2 Future Work

Although a substantial number of promising achievements on improving recommender systems have been made in this thesis, there are still numerous interesting open issues that worth study in the future.

First, we focus on the empirical study of the online learning algorithms DA-PMF, SGD-PMF, DA-RMF and SGD-RMF. Explore the convergence rate from a theoretical perspective for these algorithms are an interesting direction. A theoretical convergence rate allows us to apply online learning algorithms more confidently. Secondly, the impact of user response patterns in systems other than music streaming services might be different. Study how users decide whether to rate an item or not in online rating system involving the recommendation of movies, books could be a different avenue. There could different response patterns for different applications. Thirdly, the reputation estimation framework we develop is ignorant of the relationship between users. In online social rating systems such as Douban and Epinions, users have strong or weak connections with other users. Are normal users friend with spam users? How the reputation of a user affects his/her friends' reputation? Study user reputation under such networked setting is a promising research di-

rection. Fourthly, we study the performance of RMR using review text and rating data. In fact, RMR can take any form of text descriptive data as input and combine them with ratings. In e-commerce websites, we may have detailed description data for items and users might also provide descriptive profile data. Study how such data affects the performance can be an interesting avenue.

There are other directions that can potentially improve recommender systems. How to develop distributed learning algorithms for recommender system is a practical problem that is faced by large recommender systems. Although online learning algorithm can help alleviate the scale problem, distributed learning algorithms could be a more direct solution. Active learning for recommender system is the study of identifying items for users to rate so that the system can learn the most. Active learning can help guide the build of user problem and alleviate the cold-start problem.

---

□ **End of chapter.**

# Bibliography


[1] Jacob Abernethy, Kevin Canini, John Langford, and Alex Simma. Online collaborative filtering. Technical report, University of California at Berkeley, 2007.

[2] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, 17(6):734–749, 2005.

[3] Deepak Agarwal and Bee-Chung Chen. flda: matrix factorization through latent dirichlet allocation. In *WSDM*, pages 91–100, 2010.

[4] Justin Basilico and Thomas Hofmann. Unifying collaborative and content-based filtering. In *ICML*, 2004.

[5] J. Bennett and S. Lanning. The Netflix Prize. In *Proceedings of the KDD Cup Workshop 2007*, pages 3–6. ACM, New York, 2007.

[6] Adam Berenzweig, Beth Logan, Daniel PW Ellis, and Brian Whitman. A large-scale evaluation of acoustic and subjective music-similarity measures. *Computer Music Journal*, 28(2):63–76, 2004.

[7] Thierry Bertin-Mahieux, Daniel P. W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *ISMIR*, pages 591–596, 2011.

[8] Daniel Billsus and Michael J. Pazzani. A hybrid user model for news story classification. In *Proceedings of the Seventh International Conference on User Modeling*, UM '99, pages 99–108, Secaucus, NJ, USA, 1999. Springer-Verlag New York, Inc.

[9] Daniel Billsus and Michael J. Pazzani. User modeling for adaptive news access. *User Modeling and User-Adapted Interaction*, 10(2-3):147–180, February 2000.

[10] Christopher M Bishop et al. *Pattern recognition and machine learning*, volume 1. springer New York, 2006.

[11] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[12] Léon Bottou and Yann LeCun. Large scale online learning. In *NIPS*, 2003.

[13] Andrew P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, 1997.

[14] John S. Breese, David Heckerman, and Carl Myers Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *UAI*, pages 43–52, 1998.

[15] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the seventh international conference on World Wide Web 7*, WWW7, pages 107–117, Amsterdam, The Netherlands, The Netherlands, 1998. Elsevier Science Publishers B. V.

[16] Jay T Buckingham, John D Mehr, Paul S Rehfuss, and Robert L Rounthwaite. Network domain reputation-based spam filtering, February 3 2009. US Patent 7,487,217.

[17] Robin D. Burke, Bamshad Mobasher, Chad Williams, and Runa Bhaumik. Classification features for attack detection in collaborative recommender systems. In *KDD*, pages 542–547, 2006.

[18] P. Cano, E. Batle, T. Kalker, and J. Haitsma. A review of algorithms for audio fingerprinting. In *Multimedia Signal Processing, 2002 IEEE Workshop on*, pages 169–173, Dec 2002.

[19] Pedro Cano, Oscar Celma, Markus Koppenberger, and Javier M. Buld. Topology of music recommendation networks. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 16(1):–, 2006.

[20] Pedro Cano, Markus Koppenberger, and Nicolas Wack. Content-based music audio recommendation. In *Proceedings of the 13th Annual ACM International Conference on Multimedia*, MULTIMEDIA '05, pages 211–212, New York, NY, USA, 2005. ACM.

[21] Bin Cao, Dou Shen, Jian-Tao Sun, Xuanhui Wang, Qiang Yang, and Zheng Chen. Detect and track latent factors with online nonnegative matrix factorization. In *IJCAI*, pages 2689–2694, 2007.

[22] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *ICML*, pages 129–136, 2007.

[23] Ricardo Carreira, Jaime M. Crato, Daniel Gonçalves, and Joaquim A. Jorge. Evaluating adaptive user profiles for news classification. In *Proceedings of the 9th International Conference on Intelligent User Interfaces*, IUI '04, pages 206–212, New York, NY, USA, 2004. ACM.

[24] Chen Cheng, Haiqin Yang, Irwin King, and Michael R. Lyu. Fused matrix factorization with geographical and social in-

fluence in location-based social networks. In *AAAI*, Toronto, Canada, 2012.

[25] Paul-Alexandru Chirita, Jörg Diederich, and Wolfgang Nejdl. Mailrank: using ranking for spam detection. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 373–380. ACM, 2005.

[26] William W. Cohen, Robert E. Schapire, and Yoram Singer. Learning to order things. In *Proceedings of the 1997 conference on Advances in neural information processing systems 10*, NIPS '97, pages 451–457, Cambridge, MA, USA, 1998. MIT Press.

[27] Ram Dantu and Prakash Kolan. Detecting spam in voip networks. *Proc. SRUTI*, 5, 2005.

[28] Abhinandan Das, Mayur Datar, Ashutosh Garg, and Shyam-Sundar Rajaram. Google news personalization: scalable online collaborative filtering. In *WWW*, pages 271–280, 2007.

[29] Cristobald de Kerchove and Paul Van Dooren. Iterative filtering in reputation systems. *SIAM J. Matrix Analysis Applications*, 31(4):1812–1834, 2010.

[30] Mukund Deshpande and George Karypis. Item-based top-n recommendation algorithms. *ACM Trans. Inf. Syst.*, 22:143–177, January 2004.

[31] Mark Dredze, Koby Crammer, and Fernando Pereira. Confidence-weighted linear classification. In *ICML*, pages 264–271, 2008.

[32] Delbert Dueck, Brendan J. Frey, Delbert Dueck, and Brendan J. Frey. Probabilistic sparse matrix factorization. Technical report, University of Toronto, 2004.

[33] Holly Esquivel, Aditya Akella, and Tatsuya Mori. On the effectiveness of ip reputation for spam filtering. In *Communication Systems and Networks (COMSNETS), 2010 Second International Conference on*, pages 1–10. IEEE, 2010.

[34] Yoav Freund, Raj Iyer, Robert E. Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. *J. Mach. Learn. Res.*, 4:933–969, December 2003.

[35] Gayatree Ganu, Noemie Elhadad, and Amlie Marian. Beyond the stars: Improving rating predictions using review text content. In *WebDB*, 2009.

[36] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[37] Jennifer Golbeck and James A Hendler. Reputation network analysis for email filtering. In *CEAS*, 2004.

[38] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4:133–151, 2001. 10.1023/A:1011419012209.

[39] Fabien Gouyon and Simon Dixon. A review of automatic rhythm description systems. *Computer music journal*, 29(1):34–54, 2005.

[40] R. Guha, Ravi Kumar, Prabhakar Raghavan, and Andrew Tomkins. Propagation of trust and distrust. In *Proceedings of the 13th international conference on World Wide Web*, WWW '04, pages 403–412, New York, NY, USA, 2004. ACM.

[41] Ihsan Gunes, Cihan Kaleli, Alper Bilge, and Huseyin Polat. Shilling attacks against recommender systems: a comprehensive survey. *Artificial Intelligence Review*, pages 1–33, 2012.

[42] Ido Guy, Naama Zwerdling, Inbal Ronen, David Carmel, and Erel Uziel. Social media recommendation based on people and tags. In *SIGIR*, pages 194–201, 2010.

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

[44] Victoria J. Hodge and Jim Austin. A survey of outlier detection methodologies. *Artif. Intell. Rev.*, 22(2):85–126, 2004.

[45] Thomas Hofmann. Unsupervised learning by probabilistic latent semantic analysis. *Mach. Learn.*, 42:177–196, January 2001.

[46] Thomas Hofmann. Latent semantic models for collaborative filtering. *ACM Trans. Inf. Syst.*, 22:89–115, January 2004.

[47] Xia Hu, Jiliang Tang, Huiji Gao, and Huan Liu. Unsupervised sentiment analysis with emotional signals. In *WWW*, pages 607–618, 2013.

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

[49] David R. Hunter. Mm algorithms for generalized bradley-terry models. *Annals of Statistics*, 2004.

[50] Ah hwee Tan, Christine Teo, and Heng Mui Keng. Learning user profiles for personalized information dissemination. In *In proceedings, International Joint Conference on Neural Networks (IJCNN'98*, pages 183–188. IEEE, 1998.

[51] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20:422–446, October 2002.

[52] Rong Jin, Luo Si, and ChengXiang Zhai. Preference-based graphic models for collaborative filtering. In *UAI*, pages 329–336, 2003.

[53] Rong Jin, Luo Si, ChengXiang Zhai, and Jamie Callan. Collaborative filtering with decoupled models for preferences and ratings. In *Proceedings of the twelfth international conference on Information and knowledge management*, CIKM '03, pages 309–316, New York, NY, USA, 2003. ACM.

[54] Yohan Jo and Alice H. Oh. Aspect and sentiment unification model for online review analysis. In *WSDM*, pages 815–824, 2011.

[55] Michal Kompan and Mria Bielikov. Content-based news recommendation. In Francesco Buccafurri and Giovanni Semeraro, editors, *E-Commerce and Web Technologies*, volume 61 of *Lecture Notes in Business Information Processing*, pages 61–72. Springer Berlin Heidelberg, 2010.

[56] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *KDD*, pages 426–434, New York, NY, USA, 2008. ACM.

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

[58] Yehuda Koren. Collaborative filtering with temporal dynamics. *Commun. ACM*, 53(4):89–97, 2010.

[59] Yehuda Koren and Joe Sill. Ordrec: an ordinal model for predicting personalized item rating distributions. In *RecSys*, pages 117–124, 2011.

[60] Paulo Laureti, L. Moret, Yi-Cheng Zhang, and Y.-K. Yu. Information filtering via iterative refinement. *CoRR*, abs/physics/0608166, 2006.

[61] Neil D. Lawrence and Raquel Urtasun. Non-linear matrix factorization with gaussian processes. In *ICML*, page 76, 2009.

[62] Barry Leiba, Joel Ossher, Vadakkedathu Thomas Rajan, Richard Segal, and Mark N Wegman. Method for recognizing spam email, January 6 2009. US Patent 7,475,118.

[63] Rong-Hua Li, Jeffrey Xu Yu, Xin Huang, and Hong Cheng. Robust reputation-based ranking on bipartite rating networks. In *SDM*, pages 612–623, 2012.

[64] Yanen Li, Jia Hu, ChengXiang Zhai, and Ye Chen. Improving one-class collaborative filtering by incorporating rich user information. In *CIKM*, pages 959–968, 2010.

[65] Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7:76–80, January 2003.

[66] Guang Ling, Irwin King, and Michael R. Lyu. A unified framework for reputation estimation in online rating systems. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, 2013.

[67] Guang Ling, Michael R. Lyu, and Irwin King. Ratings meet reviews, a combined approach to recommend. In *Eighth ACM Conference on Recommender Systems, RecSys '14, Foster City, Silicon Valley, CA, USA - October 06 - 10, 2014*, pages 105–112, 2014.

[68] Guang Ling, Haiqin Yang, Irwin King, and Michael R. Lyu. Online learning for collaborative filtering. In *IJCNN*, pages 1–8, 2012.

[69] Guang Ling, Haiqin Yang, Michael R. Lyu, and Irwin King. Response aware model-based collaborative filtering. In *Pro-*

*ceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, August 14-18, 2012*, pages 501–510, 2012.

[70] Roderick J. A. Little and Donald B. Rubin. *Statistical Analysis with Missing Data, Second Edition*. Wiley-Interscience, 2 edition, September 2002.

[71] Bing Liu and Lei Zhang. A survey of opinion mining and sentiment analysis. In *Mining Text Data*, pages 415–463. Springer, 2012.

[72] Jiahui Liu, Peter Dolan, and Elin Rønby Pedersen. Personalized news recommendation based on click behavior. In *Proceedings of the 15th International Conference on Intelligent User Interfaces*, IUI '10, pages 31–40, New York, NY, USA, 2010. ACM.

[73] Nathan Nan Liu, Xiangrui Meng, Chao Liu, and Qiang Yang. Wisdom of the better few: cold start recommendation via representative based rating elicitation. In *RecSys*, pages 37–44, 2011.

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

[75] Nathan Nan Liu, Min Zhao, Evan Wei Xiang, and Qiang Yang. Online evolutionary collaborative filtering. In *RecSys*, pages 95–102, 2010.

[76] Nathan Nan Liu, Min Zhao, and Qiang Yang. Probabilistic latent preference analysis for collaborative filtering. In *CIKM*, pages 759–766, 2009.

[77] Tie-Yan Liu. Learning to rank for information retrieval. *Found. Trends Inf. Retr.*, 3:225–331, March 2009.

[78] Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. Content-based recommender systems: State of the art and trends. In Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors, *Recommender Systems Handbook*, pages 73–105. Springer US, 2011.

[79] Hao Ma, Irwin King, and Michael R. Lyu. Effective missing data prediction for collaborative filtering. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, pages 39–46, New York, NY, USA, 2007. ACM.

[80] Hao Ma, Irwin King, and Michael R. Lyu. Learning to recommend with social trust ensemble. In *SIGIR*, pages 203–210, 2009.

[81] Hao Ma, Irwin King, and Michael R. Lyu. Mining web graphs for recommendations. *IEEE Trans. Knowl. Data Eng.*, 24(6):1051–1064, 2012.

[82] Hao Ma, Dengyong Zhou, Chao Liu, Michael R. Lyu, and Irwin King. Recommender systems with social regularization. In *WSDM*, pages 287–296, 2011.

[83] Pattie Maes et al. Agents that reduce work and information overload. *Communications of the ACM*, 37(7):30–40, 1994.

[84] Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online learning for matrix factorization and sparse coding. *Journal of Machine Learning Research*, 11:19–60, 2010.

[85] John I. Marden. *Analyzing and modeling rank data*. Chapman & Hall, New York, NY, USA, 1995.

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

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

[88] Julian J. McAuley and Jure Leskovec. Hidden factors and hidden topics: understanding rating dimensions with review text. In *RecSys*, pages 165–172, 2013.

[89] Matus Medo and Joseph R. Wakeling. The effect of discrete vs. continuous-valued ratings on reputation and ranking systems. *CoRR*, abs/1001.3745, 2010.

[90] Bhaskar Mehta, Thomas Hofmann, and Peter Fankhauser. Lies and propaganda: detecting spam users in collaborative filtering. In *Proceedings of the 12th international conference on Intelligent user interfaces*, IUI '07, pages 14–21, New York, NY, USA, 2007. ACM.

[91] Bhaskar Mehta, Thomas Hofmann, and Wolfgang Nejdl. Robust collaborative filtering. In *Proceedings of the 2007 ACM conference on Recommender systems*, RecSys '07, pages 49–56, New York, NY, USA, 2007. ACM.

[92] Bhaskar Mehta and Wolfgang Nejdl. Unsupervised strategies for shilling detection and robust collaborative filtering. *User Model. User-Adapt. Interact.*, 19(1-2):65–97, 2009.

[93] Prem Melville, Raymond J. Mooney, and Ramadass Nagarajan. Content-boosted collaborative filtering for improved recommendations. In *AAAI/IAAI*, pages 187–192, 2002.

[94] Stefano Mizzaro. Quality control in scholarly publishing: A new proposal. *JASIST*, 54(11):989–1005, 2003.

[95] Samaneh Moghaddam and Martin Ester. Aspect-based opinion mining from product reviews. In *SIGIR*, page 1184, 2012.

[96] Kevin P. Murphy. Conjugate bayesian analysis of the gaussian distribution. Technical report, University of British Columbia, 2007.

[97] Yurii Nesterov. Primal-dual subgradient methods for convex problems. *Mathematical Programming*, 120(1):221–259, 2009.

[98] Kamal Nigam, John Lafferty, and Andrew McCallum. Using maximum entropy for text classification. In *IJCAI-99 workshop on machine learning for information filtering*, volume 1, pages 61–67, 1999.

[99] Michael P. O'Mahony, Neil J. Hurley, Nicholas Kushmerick, and Guenole C. M. Silvestre. Collaborative recommendation: A robustness analysis. *ACM Trans. Internet Techn.*, 4(4):344–377, 2004.

[100] Jan Overgoor, Ellery Wulczyn, and Christopher Potts. Trust propagation with mixed-effects models. In *ICWSM*, 2012.

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

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

[103] Michael J. Pazzani and Daniel Billsus. Content-based recommendation systems. In *The Adaptive Web*, pages 325–341, 2007.

[104] David M. Pennock, Eric Horvitz, Steve Lawrence, and C. Lee Giles. Collaborative filtering by personality diagnosis: A hybrid memory and model-based approach. In *UAI*, pages 473–480, 2000.

[105] Owen Phelan, Kevin McCarthy, Mike Bennett, and Barry Smyth. Terms of a feather: Content-based news recommendation and discovery using twitter. In Paul Clough, Colum Foley, Cathal Gurrin, GarethJ.F. Jones, Wessel Kraaij, Hyowon Lee, and Vanessa Mudoch, editors, *Advances in Information Retrieval*, volume 6611 of *Lecture Notes in Computer Science*, pages 448–459. Springer Berlin Heidelberg, 2011.

[106] Ian Porteous, David Newman, Alexander Ihler, Arthur Asuncion, Padhraic Smyth, and Max Welling. Fast collapsed gibbs sampling for latent dirichlet allocation. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 569–577. ACM, 2008.

[107] Sanjay Purushotham and Yan Liu. Collaborative topic regression with social matrix factorization for recommendation systems. In *ICML*, 2012.

[108] Steffen Rendle. Factorization machines with libfm. *ACM TIST*, 3(3):57, 2012.

[109] Steffen Rendle. Learning recommender systems with adaptive regularization. In *WSDM*, pages 133–142, 2012.

[110] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. *CoRR*, abs/1205.2618, 2012.

[111] Steffen Rendle and Lars Schmidt-Thieme. Pairwise interaction tensor factorization for personalized tag recommendation. In *WSDM*, pages 81–90, 2010.

[112] Jasson D. M. Rennie and Nathan Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *Proceedings of the 22nd international conference on Machine learning*, ICML '05, pages 713–719, New York, NY, USA, 2005. ACM.

[113] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *CSCW*, pages 175–186, 1994.

[114] Ruslan Salakhutdinov and Andriy Mnih. Probabilistic matrix factorization. In *NIPS*, 2007.

[115] Ruslan Salakhutdinov and Andriy Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *ICML*, pages 880–887, 2008.

[116] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey E. Hinton. Restricted boltzmann machines for collaborative filtering. In *ICML*, pages 791–798, 2007.

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

[118] Andrew I. Schein, Alexandrin Popescul, Lyle H. Ungar, and David M. Pennock. Methods and metrics for cold-start recommendations. In *SIGIR*, pages 253–260, 2002.

[119] A. Shapiro and Y. Wardi. Convergence analysis of gradient descent stochastic algorithms. *Journal of Optimization Theory and Applications*, 91:439–454, 1996.

[120] Amit Sharma and Dan Cosley. Do social explanations work?: studying and modeling the effects of social explanations in recommender systems. In *WWW*, pages 1133–1144, 2013.

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

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

[123] Ja-Hwung Su, Hsin-Ho Yeh, P.S. Yu, and V.S. Tseng. Music recommendation using content and context information mining. *Intelligent Systems, IEEE*, 25(1):16–26, Jan 2010.

[124] Michael E. Tipping and Christopher M. Bishop. Probabilistic principal component analysis. *Journal Of The Royal Statistical Society Series B*, 61(3):611–622, 1999.

[125] Ivan Titov and Ryan T. McDonald. A joint model of text and aspect ratings for sentiment summarization. In *ACL*, pages 308–316, 2008.

[126] Alex Hai Wang. Don't follow me: Spam detection in twitter. In *Security and Cryptography (SECRYPT), Proceedings of the 2010 International Conference on*, pages 1–10. IEEE, 2010.

[127] Chong Wang and David M. Blei. Collaborative topic modeling for recommending scientific articles. In *KDD*, pages 448–456, 2011.

[128] Lin Xiao. Dual averaging methods for regularized stochastic learning and online optimization. *J. Mach. Learn. Res.*, 9999:2543–2596, December 2010.

[129] Zenglin Xu, Rong Jin, Haiqin Yang, Irwin King, and Michael R. Lyu. Simple and efficient multiple kernel learning by group lasso. In *ICML*, pages 1175–1182, Haifa, Israel, 2010.

[130] Gui-Rong Xue, Chenxi Lin, Qiang Yang, WenSi Xi, Hua-Jun Zeng, Yong Yu, and Zheng Chen. Scalable collaborative filtering using cluster-based smoothing. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '05, pages 114–121, New York, NY, USA, 2005. ACM.

[131] H. Yang, G. Ling, Y. Su, M. Lyu, and I. King. Boosting response aware model-based collaborative filtering. *Knowledge and Data Engineering, IEEE Transactions on*, PP(99):1–1, 2015.

[132] Haiqin Yang, Irwin King, and Michael R. Lyu. Online learning for multi-task feature selection. In *CIKM*, pages 1693–1696, 2010.

[133] Haiqin Yang, Irwin King, and Michael R. Lyu. *Sparse Learning Under Regularization Framework*. LAP Lambert Academic Publishing, first edition, April 2011.

[134] Haiqin Yang, Zenglin Xu, Irwin King, and Michael R. Lyu. Online learning for group lasso. In *ICML*, pages 1191–1198, 2010.

[135] Haiqin Yang, Zenglin Xu, Jieping Ye, Irwin King, and Michael R. Lyu. Efficient sparse generalized multiple kernel learning. *IEEE Transactions on Neural Networks*, 22(3):433–446, 2011.

[136] Liang Zhang, Deepak Agarwal, and Bee-Chung Chen. Generalizing matrix factorization through flexible regression priors. In *RecSys*, pages 13–20, 2011.

[137] Yu Zhang, Bin Cao, and Dit-Yan Yeung. Multi-domain collaborative filtering. In *UAI*, pages 725–732, 2010.