Performance Evaluation of Latent Factor Models for Rating Prediction

by

Lan Zheng
MSc, Beijing University of Posts and Telecommunications, 2012
Bachelor of Management, Beijing University of Posts and Telecommunications, 2009

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science

in the Department of Computer Science

Performance Evaluation of Latent Factor Models for Rating Prediction

by

Lan Zheng
MSc, Beijing University of Posts and Telecommunications, 2012
Bachelor of Management, Beijing University of Posts and Telecommunications, 2009

Supervisory Committee

Dr. Kui Wu, Co-Supervisor
(Department of Computer Science)

Dr. Alex Thomo, Co-Supervisor
(Department of Computer Science)

Supervisory Committee

Dr. Kui Wu, Co-Supervisor
(Department of Computer Science)

Dr. Alex Thomo, Co-Supervisor
(Department of Computer Science)

ABSTRACT

Since the Netflix Prize competition, latent factor models (LFMs) have become the comparison "staples" for many of the recent recommender methods. Meanwhile, it is still unclear to understand the impact of data preprocessing and updating algorithms on LFMs. The performance improvement of LFMs over baseline approaches, however, hovers at only low percentage numbers. Therefore, it is time for a better understanding of their real power beyond the overall root mean square error (RMSE), which as it happens, lies at a very compressed range, without providing too much chance for deeper insight.

We introduce an experiment based handbook of LFMs and reveal data preprocessing and updating algorithms' power. We perform a detailed experimental study regarding the performance of classical staple LFMs on a classical dataset, Movielens 1M , that sheds light on a much more pronounced excellence of LFMs for particular categories of users and items, for RMSE and other measures. In particular, LFMs exhibit surprising and excellent advantages when handling several difficult user and item categories. By comparing the distributions of test ratings and predicted ratings, we show that the performance of LFMs is influenced by rating distribution. We then propose a method to estimate the performance of LFMs for a given rating dataset. Also, we provide a very simple, open-source library that implements staple LFMs achieving a similar performance as some very recent (2013) developments in LFMs, and at the same time being more transparent than some other libraries in wide use.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

A lot of people have helped me along the road to completing my master's degree. Dr. Kui Wu, Dr. Alex Thomo and Dr. Venkatesh Srinivasan are great advisors, who have provided lots of amazing feedback to my research. I can hardly finish my thesis without your mentor and support.

Thank you to my colleagues at Dr. Kui's Lab. I have greatly benefited from our office-conversations and lab-discussions. Those suggestions have helped me a lot in overcoming obstacles during my research.

I would not be successfully finishing and defending my thesis without the support of my family. My parents are always there and encourage me to believe in myself. They taught me to face every challenge with a big smile. And my brother has helped me get through the transition in my first oversea study.

Chapter 1

Introduction

1.1 The Recommendation Age

With the rapid development in big data, enterprises are facing a tremendous challenge when they try to improve their services in the increasingly competitive world. According to International Data Corporation (IDC), 1.8 zettabytes (or 1.8 trillion gigabytes) of data was created in 2011 and the number reached to 2.8 zettabytes in 2012. To be specific, Figure 1.1 from Domo infographic shows that data generated every minute can be: more than 204 million email messages, over 2 million Google search queries, or 684,000 bits of content shared on Facebook, etc. [1] Further more, IDC now forecasts that data generated within one year will climb to 40 zettabytes by 2020. [2]

As the amount of available data grows, recommender system (RS) is one of the crucial techniques to benefit people in the big data era. We are leaving the age of information and entering the age of recommendation [2]. On one hand, RS helps entrepreneurs to better target products to the right user group and therefore increases the customer conversion rate. On the other hand, consumers find it easier to make the right choice among numerous products with the help of RS. As a result, RS has been widely implemented in diverse areas in the Internet, such as e-commerce [25], music [37], movie [67, 50, 6], etc.

[1] Source: `http://www.domo.com/blog/2012/06/how-much-data-is-created-every-minute/`

[2] Source: `http://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm`

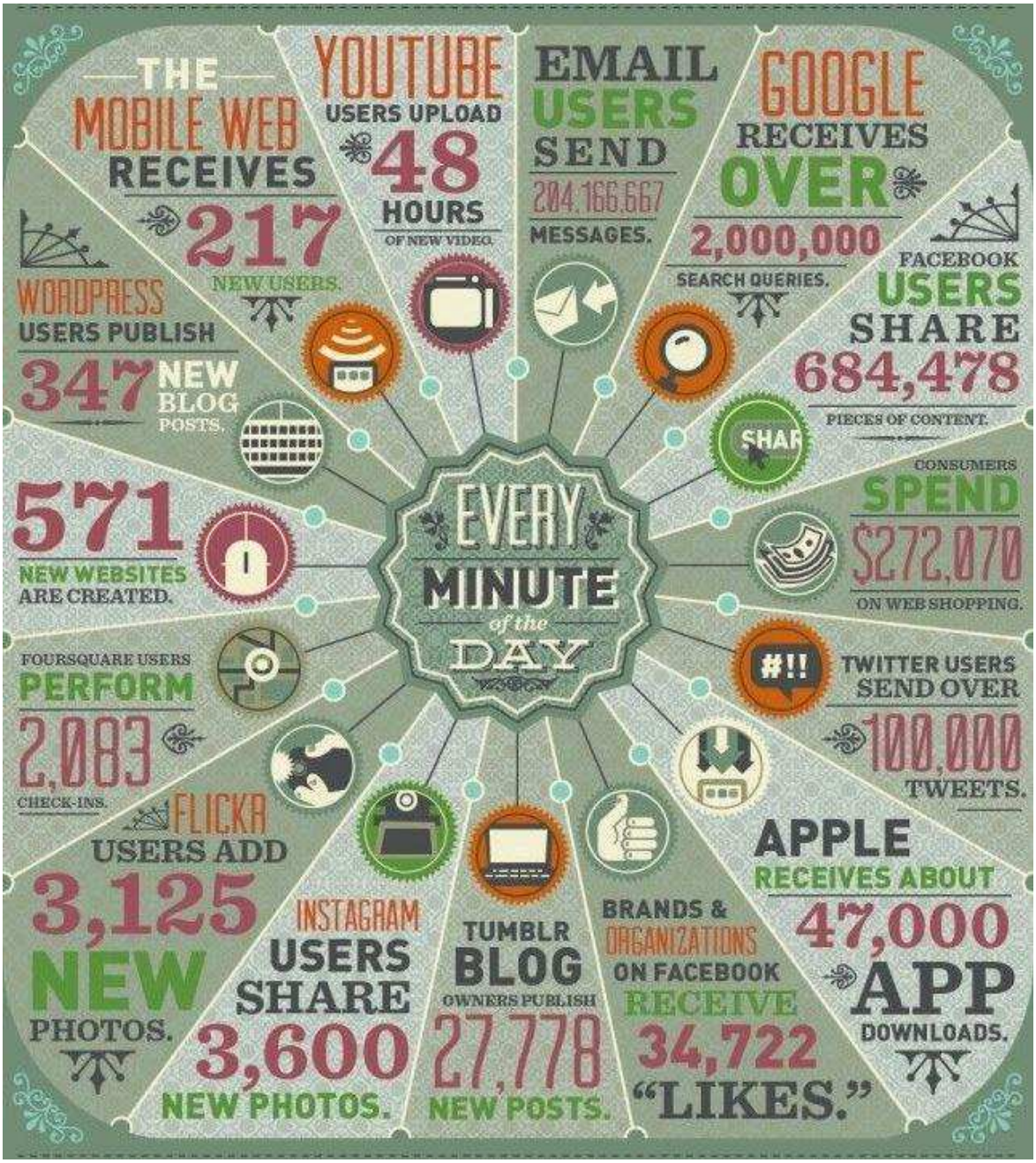


Figure 1.1: Data created every minute

1.2 Typical Recommender System

Since RS plays an important role in the information age, it has become a basic service of a modern web service provider. Recommendation engine can be seen among most of the popular social media and e-commerce websites, including Amazon, LinkedIn,

Hulu, Netflix, Facebook, Twitter, Google, and etc. While the goals of different RS engines and the algorithms they use may vary, there is no doubt that both service providers and users have benefited a lot from the use of RS.

1.2.1 Amazon's recommendation

Amazon was the earliest pioneer in RS, and personalized recommendation has been an inseparable part of its operations. By automatically adjusting the display of products according to customer interests, the click-through rates and conversion rates, measurement of web_based advertising effects, vastly exceed traditional untargeted approaches, such as banner advertisements and top-seller lists [39].

The best known RS algorithm at Amazon is item-to-item collaborative filtering, which makes recommendation based on customers' shopping history. General recommendations provided can be "Frequently Bought Together" and "Customers Who Viewed This Item Also Viewed", as shown in Figure 1.2. Also, Figure 1.3 shows that personalized recommendations are also displayed in "Recommended for you" section for logged-in customers, when the RS engine is in full action.

Figure 1.2: General recommendation at Amazon

Figure 1.3: Personalized recommendation at Amazon

1.2.2 Netflix Recommendations

Netflix, an on-demand Internet streaming media provider, enhanced the impact of RS by announcing the Netflix Prize in 2006. In that competition, one million dollars were offered to whoever improved the accuracy of their existing system Cinematch by 10%. Thanks to the improved RS, statistics show that 75% of what people watch at Netflix is from the recommendations generated from the adapted RS algorithms. [3]

Netflix is still actively looking for better ways to improve their recommendation performance by suggesting interesting movies to their members. According to the Netflix Tech Blog [4], Netflix is trying to include Deep Learning into their recommendation engine. In the distributed neural networks, by using cloud computing from Amazon Web Services and GPUs from Nvidia, Netflix is able to train four million samples and obtain more personalized recommendations within only 47 minutes, which is a big improvement to original 20 hours.

1.2.3 Recommendations at LinkedIn

LinkedIn has the world's largest professional network. The LinkedIn recommendations platform computes more than 100 billion recommendations every week in 2011,

[3] http://techblog.netflix.com/2012/04/netflix-recommendations-beyond-5-stars.html
[4] `http://techblog.netflix.com/search?updated-min=2014-01-01T00:00:00-08:00&updated-max=2015`

as suggested in the talk given by Abhishek Gupta. According to LinkedIn's study, 50% of total job applications and job views from members were a direct result of recommendations, while that number was only 6% in the past year and half. Moreover, contribution of RS is observed across all LinedIn products and is growing by the day. [5]

At LinkedIn's platform, recommendations are based on the terabytes of data flowing through the systems, which are generated from member's profile, their connections and their activities on this website. This huge semi-structured data can thus be used to build the following recommendation products:

1. Job Recommendation: Suggest top jobs for the job hunter based on the member's profile.

2. Talent Recommendation: Suggest best fit candidates for the recruiters according to job descriptions or requirements.

3. News Recommendation: Suggest top news within the industry that the member might be interested in or be related with.

4. Companies Recommendation: Recommend companies for the members to follow.

5. Connections Recommendations: Recommend new connections within or outside of LinkedIn for members to connect with.

6. Similar Profiles: Find most similar candidates for recruiters based on a given set of candidate profiles.

1.2.4 Hulu's Recommendation System

Hulu offers ad-supported on-demand streaming videos, including TV shows, movies, webisodes and other new media. Without the help of RS, Hulu users would find it impossible to discover interesting videos given tens of thousands of hours of premium video content. Therefore, it is important for Hulu's RS to help users find new videos that match their historic interests.

[5] http://www.hadoopworld.com/session/linkedin-products-you-may-like/

Figure 1.4 shows that Hulu's recommendations are more useful than simple approaches, such as recommending most popular or highest rated videos. The click-through rate(CTR) of Hulu's RS is twice higher than that of common techniques. [6]

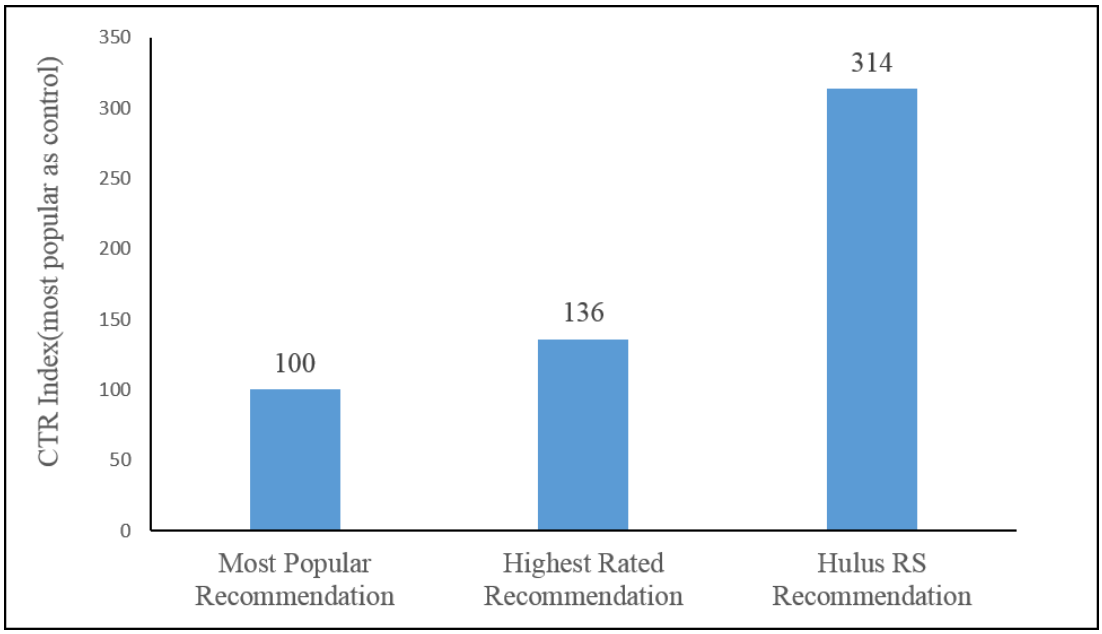


Figure 1.4: Performance of Hulus Recommendation Hub

1.3 Importance of Latent Factor Models

Until now, Latent Factor Models (LFMs) have gained an immense popularity for the implementation of RS, especially in the task of rating prediction. As demonstrated in the Netflix Prize competition, LFMs are superior to classic neighbourhood methods for the purpose of movie recommendations, allowing the incorporation of additional side information, such as implicit feedback, temporal effects and confidence levels [30].

Hundreds of citations can be found to the central articles of Bell, Koren, and Volinsky (BKV) describing their LFM methodology for movie recommendation [27, 30, 29]. The basic BKV methods have a great appeal of simplicity that has also contributed to their popularity. The metric of choice for comparing recommenders is the root mean squared error (RMSE), and improving on it by devising new adaptations and additions has become a "sport" in the quest for better recommender systems.

[6] Source: `http://tech.hulu.com/blog/2011/09/19/recommendation-system/`

1.4 Thesis Contributions

Notwithstanding their popularity, the fact is that LFMs improve RMSE against naive baselines by successfully handling difficult categories. For instance, the LFMs described in [29] show an improvement of about 0.05 against baseline approaches on the classical MovieLens 1M dataset (with ratings in a 1-5 scale). This is somewhat discouraging and naturally raises the question if the popularity of LFMs is well-founded. Here evidence towards a positive answer to this question is provided in this thesis and the excellence of LFMs over baseline approaches is exhibited by providing a detailed study on the Movielens 1M dataset. This study shows the advantage of LFMs over baseline approaches, especially for difficult to handle categories of users and items. Further more, the insight on the interplay of biases and latent factors for each of the considered metrics and categories is explained. Finally, experiments show that the performance of LFMs might be influenced by the underlying rating prediction.

More specifically the contributions of this thesis are as follows.

1. We provide an experiment based handbook of LFMs and explain the general process of working with LFMs, including data preprocessing, updating algorithms.

2. We describe four different combinations of latent factors with the mean, user and item biases, and provide a simple library implementation that is more transparent than other available software toolkits.

3. We show that the performance of the LFMs over the MovieLens 1M data set (without considering user and item categories) exhibits a similar behavior as that of the baseline. Therefore, for the general case, the benefit from latent factors is not particularly strong.

4. We define user and item categories, some of which are difficult to handle by recommender systems (RMSE is high for them), and show that it is for some of these categories that the latent factors really excel and produce impressive results.

5. We compare the groundtruth rating (i.e., real-world data) distribution as well as the predicted rating distribution for each category and show that LFMs have better performance on a Gaussian-like distributed rating dataset. We

then propose a method to estimate the performance of LFMs for a given rating dataset.

Chapter 2

Background and Related Work

2.1 Recommender System

Generally speaking, any software tools and techniques that provide suggestions about relevant items for users can be considered as recommender systems [54, 35, 42, 10]. The term "item" represents the object that is recommended to users by a recommender system. An item could be a movie, a computer or even a restaurant. And relevant items mean that those items are useful for users.

On the other hand, RS can be regarded as a special search engine. RS can automatically provide search results, i.e. recommendations, without any search words. Thus it can be used to ease the ever growing pressure of information overload. With the help of RS, users can make decisions based on the provided recommendations. Meanwhile, merchants can adjust their products or service based on their customers' preference.

Recommendations produced by RS can be non-personalized or personalized. Non-personalized recommendations are much easier to generate and are usually based on the whole population. Typical examples include New York Times Best Seller List, featured sale on the front pages of major e-commerce websites, etc. Even though non-personalized recommendations are informative, they may not be tailored for individual users' need. As such, personalized recommendations become more important and are actively studied in current RS research. This thesis will be mainly focused on personalized RS.

Generally speaking, an RS takes the input of users' past preferences and outputs ranked list of items as personalized recommendations. Users' preferences can be

explicitly expressed, for example, ratings of hotels, which can be easily captured. There are also implicit user preferences that can be inferred from users' actions. For instance, users' dislikes can be inferred from the movies that they choose not to watch or give low ratings.

2.1.1 Recommender System Strategies

There are three main strategies in RS, content-based filtering(CBF), collaborative filtering(CF) and hybrid recommender systems. CBF utilizes characteristics of items to make recommendations. On the contrary, CF usually builds models on observed rating data and rarely uses user or item information. Hybrid approaches try to obtain a better solution by combining CBF and CF.

CBF originates from information retrieval(IR) field, and therefore many IR techniques have be applied in this context [48, 46]. CBF is mainly used in the context where detail information of items' is provided, such as books, movies, news, web pages, etc. Items are recommended based on a comparison between items' characteristics and users' profiles. The recommendation is based purely on the match between the descriptive information of user and item. For example, CBF would recommend romantic movies to users who always give high ratings to romantic movies.

According to CF, items are recommended to a user based on the users' similar preferences. Memory-based algorithms and model-based algorithms are two general classes of CF algorithms. Memory-based algorithms make recommendations based on users' or items' similarity, which can be computed from the entire user rating data. This method is easy to implement, effective and has been used in many commercial systems[39, 58, 62]. Model-based approaches try to find patterns in the training data via machine learning algorithms. Models are built to detect users' preferences and items' characteristics from the observed ratings and then they are used to match users and items. These types of methods are popular ever since the Netflix Prize and have a lot of applications, such as video suggestion in YouTube [4].

Hybrid approaches are taken by combining CF and CBF to avoid the limitations of the single method, such as cold start problems and sparsity problems, which are related with insufficient or incomplete information of users or items. These approaches can be implemented in four ways [1]: predicting using CF and CBF separately and then combining the two individual predictions [61, 64, 50], incorporating some CBF features into a CF method (or including some CF features in a CBF method) [63]

and building a unifying model that incorporates both CF and CBF [51].

2.1.2 Recommender System Tools

There have been several projects initiated to implement RS algorithms to make recommendations. The main targets as well as the existing functionalities of these software packages vary greatly. Thorsten Angermannn [3] has performed a systematic comparison among several RS projects and reported on experiments involving several recommender system frameworks. In this thesis, we will focus on following open-source projects.

1. easyrec [1] aims to provide a ready-to-use personalized solution integrated in real world applications. It is written in Java under the terms of the GNU General Public License version 3 or later. The most important feature of easyrec is that its access is provided through web services for an easy and quick integration.

   By using easyrec, the service provider can make recommendations without any additional programming. However, the algorithms already implemented in easyrec are quite simple. In order to apply advanced algorithms, users have to develop their own recommendation algorithms as plugins.

2. GraphLab [2] and GraphChi [41, 32] are both a graph-based open source distributed computation framework. They include a collection of implementations of machine learning and data mining algorithms, with one section dedicated to CF algorithms.

   Both GraphLab and GraphChi are capable of handling large scale data sets and models. The major difference between the two projects is that GraphLab is distributed and targeted for a computer cluster. While GraphChi can be implemented on just a single machine. Besides, GraphChi includes both C++ implementation and Java version, which provides more flexibility.

   The main advantage of GraphLab and GraphChi is the ability to efficiently dealing with big data. By virtue of graph computation and distributed computing, GraphLab is able to scale to graphs with billions of vertices and edges with a relatively fast speed. While designed for small-scale systems, GraphChi

[1] http://easyrec.org/about
[2] http://graphlab.org/

is still available for large graph computation on just a single machine with the goal of solving problems that previously demanded an entire cluster.

Because of its great commercial potential in dealing with large data sets, GraphLab provides an ipython cloud interface, GraphLab Notebook [3]. More over, GraphLab has implemented a rich family of LFMs, including PMF [55], ALS [67], SGD [30], bias-SGD [27], SVD++ [27], time-SVD++ [28], CLIMF [60], NMF(non-negative matrix factorization) [59], SVD(restarted lanczos) [22], RBM(restricted Bolzman machines) [23], FM(factorization machines) [52]. Thus users can use GraphLab Notebook to develop powerful recommendation tools with simple instructions. .

3. LensKit [4], developed by GroupsLens, is a Java based open source toolkit for building, researching, and studying RS [17]. Its main goal is to provide a platform for reproducible RS research.

   LensKit is licensed under the GNU Lesser General Public License version 2 or later. It has implemented a variety of algorithms, including FunkSVD [47, 18] for LFMs. The software not only supports rating prediction but also Top-N recommendation. A wide range of evaluation metrics, including coverage, entropy, half-life utility [9], nDCG [26], MAE, global RMSE and per user RMSE, have also been provided. Cross-validation modular with multiple options is another distinct feature in LensKit . This function can not only split rating data by percentage, but also support Given-N method [9]. The ratings can be withheld randomly or by timestamp.

4. Mahout [5], also known as the Apache Mahout^{TM} machine learning library, offers batch based collaborative filtering under Apache software license version 2.0. Mahout is scalable to reasonably large data sets since it uses map/reduce paradigm implemented on Apache Hadoop. In addition, Mahout can be accessed via web services and HTTP.

   Mahout provides both non-distributed and distributed Hadoop-based recommenders. The non-distributed recommenders were originated from the "Taste" project. This feature only includes memory-based, item-based and slope one

[3] https://beta.graphlab.com/
[4] http://lenskit.grouplens.org/
[5] http://mahout.apache.org/

recommender instead of LFMs. On the other hand, the distributed version supports not only item based CF, but also LFMs with ALS updating rule [67]. It is worth noting that the performance of LFM in Mahout is restricted because of the enormous overhead of scheduling and checkpointing at each single iteration.

5. MyMediaLite [6] is currently the most complete open source RS frameworks, in terms of recommendation tasks and implemented methods [20]. It targets both academic and industrial users and keeps records of state of the art RS algorithms. MyMediaLite is implemented in $C\#$, but can be easily called from other languages, Ruby and Python, for example.

   MyMediaLite provides not only the implementation of different LFMs, it also supports hyper-parameter selection. This approach is based on grid search and Nelder-Mead[50]. And it is extremely useful for parameter tuning and reproducing experiments.

6. PREA [7] [36] is a Java based open source framework under BSD. It includes several up-to-date algorithms as well as the classical algorithms. PREA is rich in the evaluation metrics, varying from RMSE and MAE to normalized mean absolute error (NMAE), asymmetric measures, HLU, NDCG, Kendall's Tau [43] and Spearman [43].

   PREA also has the potential to deal with large scale data sets. The goal is achieved by implementing several fast algorithms. Take LLORMA [34] for example, it is designed for scalable matrix approximation.

7. SVD feature [8] [11] is a competition oriented software under Apache 2.0 license. And it has won great reputations for winning KDD cup for two consecutive years. SVD feature aims to solve the feature-based matrix factorization and allows for incorporating side information such as neighborhood relationship and temporal dynamics.

8. libFM [9], also known as Factorization Machine Library, is a generic approach that supports various factorization models. Similar to SVD feature, libFM also supports models with a variety of side information. What's more, the updating

[6] http://www.mymedialite.net/
[7] http://prea.gatech.edu/
[8] `http://svdfeature.apexlab.org/wiki/Main_Page`
[9] http://www.libfm.org/

rules supported in libFM include SGD, ALS as well as Bayesian inference using Markov Chain Monte Carlo(MCMC).

The comparison of the above eight open source RS projects is shown in Table 2.1. Most of them, except for easyrec, LensKit and MyMediaLite, are scalable. This is achieved either by distributed implementations or fast algorithms. All the projects except easyrec support LFMs.

The trends shown in the development of the projects can be summarized as below.

1. Research and education oriented: LensKit and MyMediaLite are primarily intended to support research and education, not for large-scale data sets.

2. Scalability: GraphLab, GraphChi and Apache Mahout are mainly developed to support large data sets through efficient computing approaches, graph computation or distributed computing. Other projects have developed or implemented fast algorithms for scalability.

3. Compatibility and commercial use: easyrec is originally developed as a web application. MyMediaLite and Apache Mahout also allows for web access. GraphLab allows access from Python applications by using GraphLab Notebook.

Table 2.1: Comparison of Recommender System Software

Project	Latest Version Release Date	License	Language	Scalable	LFMs
easyrec	0.99 $2013-11-21$	GPL (3.0 or later)	Java	–	–
GraphLab	2.2 $2013-7-2$	Apache License (2.0)	C++	Yes	Yes
GraphChi	0.2 $2013-1-21$	Apache License (2.0)	C++ Java	Yes	Yes
LensKit	2.0.5 $2013-11-11$	LGPL (2.0 or later)	Java	–	Yes
Mahout	0.8 $2013-7-24$	Apache License (2.0)	Java	Yes	Yes
MyMediaLite	3.10 $2013-9-23$	GPL	*C#*	–	Yes
PREA	1.2 $2013-6-13$	BSD	Java	Yes	Yes
SVD feature	1.2.2 $2013-1-9$	Apache 2.0	C++	Yes	Yes
libFM	1.40 $2014-09-14$	GPL v3 license	C++	Yes	Yes

2.2 Latent Factor Models for Recommender System

2.2.1 Notations

Following notation are used in this thesis:

(i) $\mathbf{R} = \{r_{ui}\}_{n_u \times n_i}$: the user-item matrix, where each element r_{ui} represents the observed rating of user u towards item i. n_u indicates the number of users, and n_i designates the number of items.

(ii) $\hat{r}_{ui}$: the predicted rating for user u and item i.

(iii) $\mathbf{I}$: the index set of the known ratings.

(iv) N: the total number of observed ratings.

(v) m: the mean rating of all observed ratings.

(vi) F: the number of latent features for users and items.

(vii) b_u: bias for user u. It is a real number for each user that captures how lenient or critical user u is.

(viii) b_i: bias for item i. A real number for each item that shows how liked item i is. For example, b_i is expected to be an negative value if item i is a terrible movie.

(ix) $\mathbf{P}$: the user latent feature matrix, $\mathbf{P} = [\mathbf{p}_u]$, where $\mathbf{p}_u \in \mathbb{R}^F, u = 1...n_u$, is the u^{th} column of $\mathbf{P}$.

(x) $\mathbf{p}_u$: vector of latent factor values for user u. It is used to characterize user u by F real values.

(xi) $\mathbf{Q}$: the item latent feature matrix, $\mathbf{Q} = [\mathbf{q}_i]$, where $\mathbf{q}_i \in \mathbb{R}^F, i = 1...n_i$, denotes the i^{th} column of $\mathbf{Q}$.

(xii) $\mathbf{q}_i$: vector of latent factor values for item i. It is used to characterize item i by F real values.

(xiii) e_{ui}: the absolute error between the predicted rating $\hat{r}_{ui}$ and the real rating r_{ui}.

(xiv) λ, λ_1, λ_2, λ_3, λ_4: regularization parameters.

(xv) γ: the learning rate.

2.2.2 Models

LFMs assume that a recommendation is made between highly correspondent item and user pair. And the similarity between users and items is measured through some factors hidden in the rating data. Therefore, LFMs attempt to characterize users and items by vectors of factors inferred from item rating patterns.

LFMs are generally built on a matrix called rating matrix, or user-item matrix. In the rating matrix, rows are referred to users and columns are items. Each entry in the matrix represent the rating of the user in the row giving to item at the column. An example of a rating matrix is shown in Figure 2.1.

	i1	i2	i3	i4
u1	1	2	?	5
u2		3	?	
u3	2	?	4	5

Figure 2.1: Rating matrix example

The following combinations are considered for computing the predicted rating from LFMs.

(1). $\mathbf{p}_u \cdot \mathbf{q}_i$

(2). $m + \mathbf{p}_u \cdot \mathbf{q}_i$

(3). $m + b_u + b_i + \mathbf{p}_u \cdot \mathbf{q}_i$

(1) is the *pure factor* approach, and (2) and (3) are the *mixed* approaches, with (3) being the standard latent factor model advocated by Koren and Bell [29].

The left part of model (3) is called baseline method, i.e. $m+b_u+b_i$. m is the overall average rating, capturing the general preference of users towards items. For example, the recommendation task is to make predictions for the ratings of books. Suppose the average rating over all books, m, is 3.6. Furthermore, Pride and Prejudice is better than an average book, so it tends to be rated 0.7 stars above m, and $b_i = 0.7$. On the other hand, Tony is a critical user and he likes to rate 0.4 stars lower than the

average, $b_u = -0.4$. Therefore, the rating prediction from baseline method for user Tony on Pride and Prejudice would be $3.6 + 0.7 - 0.4 = 3.9$.

The right part of model (3) measures the user-item interaction with inner products of latent factors. As a result, each item is associated with a vector q_i, and each user is associated with a vector p_u. These latent factors try to explain ratings by characterizing both items and users on factors inferred from user feedback. For example, when the recommended items are books, factors might measure level of romance, number of pages, or author's popularity. If Tony loves reading big romantic books by popular writers, so the p_u would have relatively larger values on these three factors. If a book also has large values on these three factors, this book will be recommended to Tony because the inner product of p_u and q_i would be large.

2.2.3 Learning Algorithms

All b_u, b_i, $\mathbf{p}_u$, and $\mathbf{q}_i$ are typically learned by learning algorithms, stochastic gradient descent(SGD) or alternating least squares(ALS).

SGD

SGD procedure strives to minimize the squared error $e_{ui}^2 = (r_{ui} - \hat{r}_{ui})^2$ for each existing r_{ui} entry. Take the standard latent factor model for example. The system minimizes the regularized squared error as shown in the following equation:

$f(\mathbf{P}, \mathbf{Q}) = \sum_{(u,i)\in\mathbf{I}} (r_{ui} - \mathbf{p}_u^T \mathbf{q}_i)^2 + \lambda(\|\mathbf{p}_u\|^2 + \|\mathbf{q}_i\|^2)$

where $\hat{r}_{ui} = \mathbf{p}_u^T \mathbf{q}_i$.

The SGD rules for updating b_u, b_i, $\mathbf{p}_u$, and $\mathbf{q}_i$ at each known r_{ui} are

$$\begin{aligned} b_u &\leftarrow b_u + \gamma(e_{ui} - \lambda_1 b_u) \\ b_i &\leftarrow b_i + \gamma(e_{ui} - \lambda_2 b_i) \\ \mathbf{p}_u &\leftarrow \mathbf{p}_u + \gamma(e_{ui}\mathbf{q}_i - \lambda_3 \mathbf{p}_u) \\ \mathbf{q}_i &\leftarrow \mathbf{q}_i + \gamma(e_{ui}\mathbf{p}_u - \lambda_4 \mathbf{q}_i) \end{aligned}$$

In the implementation, b_u and b_i can be initialized as 0, and small random numbers can be assigned to the latent factors of p_u and q_i. The learning rate and regularization constants can be tuned by cross-validation. Usually, γ starts from 0.001 and regularization constants start from 0.01.

ALS

The objective function related with ALS [67] is

$f(\mathbf{P}, \mathbf{Q}) = \sum_{(u,i)\in\mathbf{I}}(r_{ui} - \mathbf{p}_u^T\mathbf{q}_i)^2 + \lambda(\sum_u n_{r_u}\|\mathbf{p}_u\|^2 + \sum_i n_{r_i}\|\mathbf{q}_i\|^2)$

where n_{r_u} and n_{r_i} denotes the number of ratings of user u and item i respectively. n_{r_u} and n_{r_i} are introduced to address the overfitting problem brought by the increase of number of features or number of iterations.

Since both $\mathbf{p}_u$ and $\mathbf{q}_i$ are unknown, the objective equation is not convex. However, this problem can be solved by rotating between fixing $\mathbf{q}_i$ and $\mathbf{p}_u$. This is the main reason it is called the alternating least square (ALS) approach. The general updating rules of ALS are as follows:

Step 1 . Initialize item latent features $\mathbf{Q}$ first by assigning the average ratings of each item as the first row, and small random numbers for the rest entries.

Step 2 . Fix $\mathbf{Q}$, solve for $\mathbf{P}$ by minimizing the above objective function. $\mathbf{p}_u = \mathbf{A}_u^{-1}\mathbf{V}_u$, where $\mathbf{A}_u = \mathbf{Q}_{\mathbf{I}_u^P}\mathbf{Q}_{\mathbf{I}_u^P}^T + \lambda n_{r_u}\mathbf{E}$, $\mathbf{V}_u = \mathbf{Q}_{\mathbf{I}_u^P}\mathbf{R}^T(u, \mathbf{I}_u^P)$. E is the $F \times F$ identity matrix. $\mathbf{I}_u^P$ is the set of indices of items that user u rated, and n_{r_u} is the cardinality of $\mathbf{I}_u^P$. Similarly $\mathbf{I}_i^Q$ denotes the set of indices of users who rated item i, and n_{r_i} is the cardinality of $\mathbf{I}_i^Q$.$\mathbf{Q}_{\mathbf{I}_u^P}$ is the sub-matrix of $\mathbf{Q}$ where columns $j \in \mathbf{I}_u^P$ are selected, and $\mathbf{R}^T(u, \mathbf{I}_u^P)$ denotes the row vector where columns $j \in \mathbf{I}_u^P$ of the $u-$th row of $\mathbf{R}$ are selected.

Step 3 . Fix $\mathbf{P}$, solve for $\mathbf{Q}$ by minimizing the above objective function similarly. $\mathbf{q}_i = \mathbf{A}_i^{-1}\mathbf{V}_i$, where $\mathbf{A}_i = \mathbf{P}_{\mathbf{I}_i^Q}\mathbf{P}_{\mathbf{I}_i^Q}^T + \lambda n_{r_i}\mathbf{E}$, $\mathbf{V}_i = \mathbf{P}_{\mathbf{I}_i^Q}\mathbf{R}^T(\mathbf{I}_i^Q, i)$. $\mathbf{P}_{\mathbf{I}_i^Q}$ is the sub-matrix of $\mathbf{P}$ where columns $j \in \mathbf{I}_i^Q$ are selected, and $\mathbf{R}^T(\mathbf{I}_i^Q, i)$ denotes the column vector where rows $j \in \mathbf{I}_i^Q$ of the $i-$th column of $\mathbf{R}$ are selected.

Step 4 . Repeat the above two steps until a stopping criterion is satisfied, for example, the specified iteration number has been reached.

2.3 Evaluation Metrics

Challenges and problems still exist in evaluating CF algorithms ever since RS accuracy has been evaluated in 1994 [53]. Recommendation accuracy metrics for rating prediction tasks can be broadly classified into two classes, prediction accuracy and classification accuracy.

2.3.1 Prediction Accuracy

Prediction accuracy is used to evaluate how close the predicted ratings obtained are to the observed ratings. The most popular metric of this kind is the mean absolute error(MAE) and root mean squared error(RMSE).

MAE can be used to measure the average absolute deviation between the predicted ratings and the real ratings.

$MAE = \frac{\sum_{(u,i)\in \mathbf{I}} |r_{ui} - \hat{r}_{ui}|}{N}$

RMSE is another prediction accuracy metric that is similar to MAE. Their main difference lies into that RMSE focuses more on large errors. RMSE has been used in the Netflix Prize competition [6], and is still very popular in recent years.

$RMSE = \sqrt{\frac{\sum_{(u,i)\in \mathbf{I}} (r_{ui} - \hat{r}_{ui})^2}{N}}$

2.3.2 Classification Accuracy

Classification accuracy measures shows how well the system makes decisions about good items and bad items. Typical metrics of this type include precision, recall, F-measure and accuracy. Generally, they are appropriate for finding good items task. Thus, in order to apply them in rating prediction task, good or bad items should be defined first. Since data used in this thesis has a rating scale of 1 - 5, the following four performance metrics are evaluate with 3.5 as the threshold to classify good and bad items.

Precision and recall, originally popular for evaluating information retrieval systems, have been used for the evaluation of RS since 1998 [8, 5, 56, 57].

Classification accuracy metrics are computed based on a 2×2 table, as shown in Table 2.2. The target ratings are separated into two classes, either good or bad. Thus, rating scale that is not binary should be transformed into binary scale. In this study, if the original rating is between 1 and 5, every rating over 3.5 is regarded as

good and the rest are transformed as bad. Also, it is assumed that users will consider all items that are retrieved.

Table 2.2: Illustration of Classification Accuracy

		True Rating	
		Good	Bad
Predicted Rating	Good	True Positive	False Positive
	Bad	False Negative	True Negative

$$precision = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

$$recall = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

$$F - measure = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$accuracy = \frac{\text{True Negative} + \text{True Positive}}{\text{Total Number of Users}}$$

2.4 Related Work

Current study on RS usually jumps to the experiment part directly from description of the algorithm, which makes it hard to understand the gap between raw data and the algorithm [54]. The influence of data preprocessing has not been fully explored yet. Our goal is to understand the effect of data preprocessing on the performance of LFMs and provide a clear explanation with experiment results.

In addition, little work has been done to explain why different RS strategies are good or bad in particular circumstances. Ekstrand and Riedl [16] confirmed that different algorithms do fail on different users and items and identified some users features in predicting relative algorithm's performance. However, they have not found out the reason behind this phenomenon. Alice Zheng [10] wrote a post about choosing an appropriate model based on the data and the evaluation of the outcome. But these suggestions are made from the point of applicability instead of performance. In

[10]http://blog.graphlab.com/choosing-a-recommender-model

our work, we showed that LFMs excel on some difficult cases and the reason behind that is that the performance of LFMs is influenced by the rating distribution.

Chapter 3

Experiment Design

3.1 Data sets

Movielens 1M data set(ML 1M) [1] and Movielens 100K data set(ML 100K) [2] were used and evaluated in this experiment. In particular, our experiments were mainly based on Movielens 1M data, while Movielens 100K data was only used to illustrate the effect of data preprocessing. Both data sets are popular benchmarks in the field of CF and can easily been obtained from the Internet. They have been widely used in many CF experiments [19, 17, 33, 65] along with Netflix dataset[6]. Moreover, these two datasets pertain to the same domain, the recommendation of movies. Therefore, they are especially useful for our purpose.

The MovieLens datasets contain real rating data for movies on the MovieLens web site[3]. They are collected and provided by GroupLens Research. Movielens 1M data set consists of $1,000,209$ ratings from $6,040$ users on $3,706$ movies. The sparsity of it is 95.53% and the minimum number of ratings given by users is 20. In this thesis, sparsity is computed as the fraction of zero elements over the total number of elements in the rating matrix. Therefore, it is a quite dense dataset compared with Netflix dataset and Epinions dataset[44, 45]. It is still been actively used as a standard dataset for evaluating collaborative filtering methods.

There are $100,000$ ratings with 943 users and $1,682$ movies in Movielens 100K data set. All the data covered from September 19th, 1997 through April 22nd, 1998. It is also a dense data set with each user having at least 20 ratings. Movielens 100K

[1] http://www.grouplens.org/node/73
[2] http://files.grouplens.org/datasets/movielens/ml-100k/
[3] http://movielens.umn.edu

data is relatively small data set compared to other standard benchmark data, so it is preferred by experiments requiring heavy computations [66]. General statistics of the two data sets are shown in Table 3.1 .

Table 3.1: General statistics Movielens data sets

Datasets	Users	Items	Ratings	Sparsity
ML 1M	6,040	3,706	1,000,209	95.53%
ML 100K	943	1,682	100,000	93.70%

As seen in Figure 3.1, the distributions of the number of ratings that each user give obey power law for both MovieLens data sets. Also, most items receive a relative small number of ratings, as shown in Figure 3.2. Figure 3.3 shows that users tend to rate movies that they like and thus give higher ratings. Most ratings are greater than average rating in both MovieLens dataset. In Movielens 1M data set, 84% rating are greater than or equal to 3.

In addition, this study only focuses on evaluating pure CF algorithms where only the ratings are considered. Therefore, extra information is not referred in the models discussed despite that both datasets contain additional information. For example, MovieLens contatins genre information about the movies. Besides, the study is based on data in the movie domain. Since the performance of CF algorithm is strongly dependent on the characteristics of the dataset, the results obtained in this study might be different from experiment results applied in other domains.

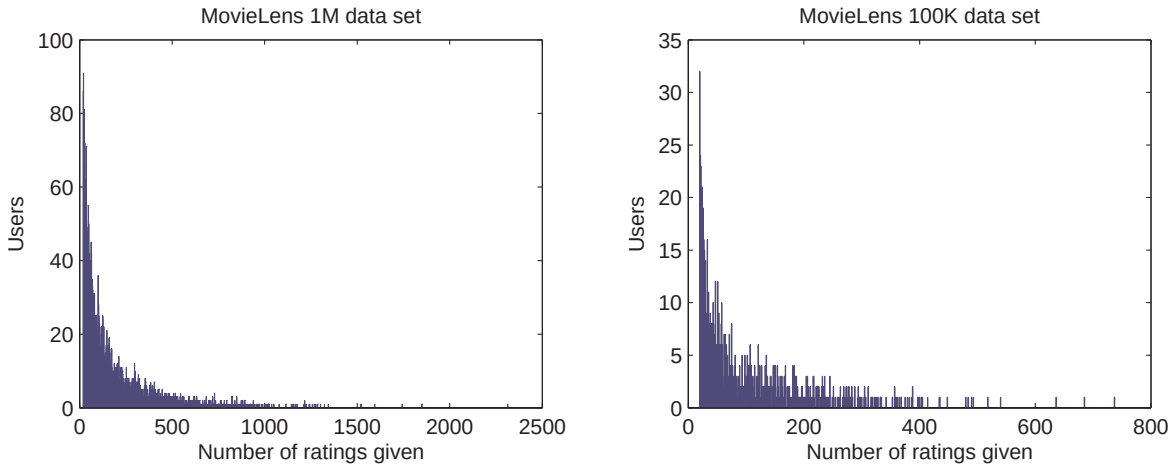


Figure 3.1: Ratings per user for ML 1M and ML 100K

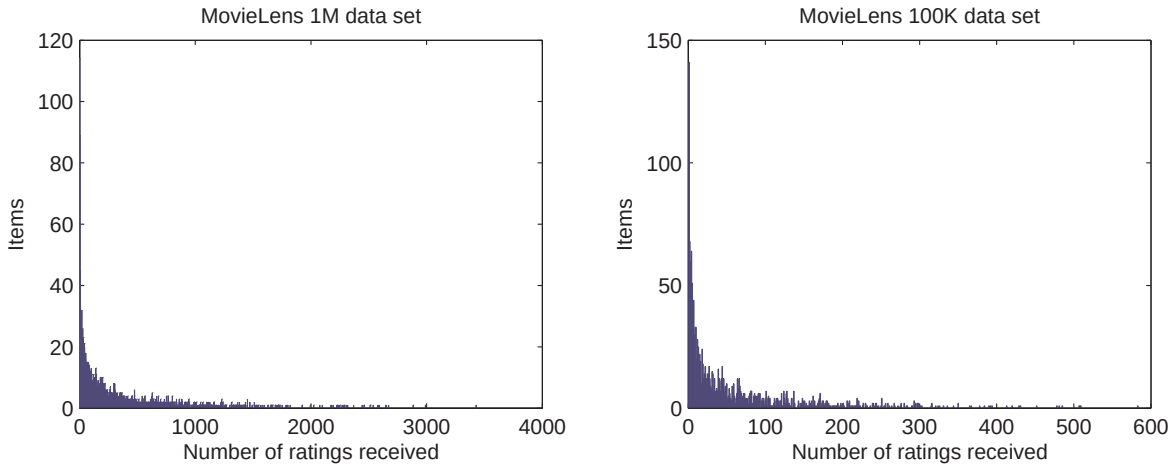


Figure 3.2: Ratings per item for ML 1M and ML 100K

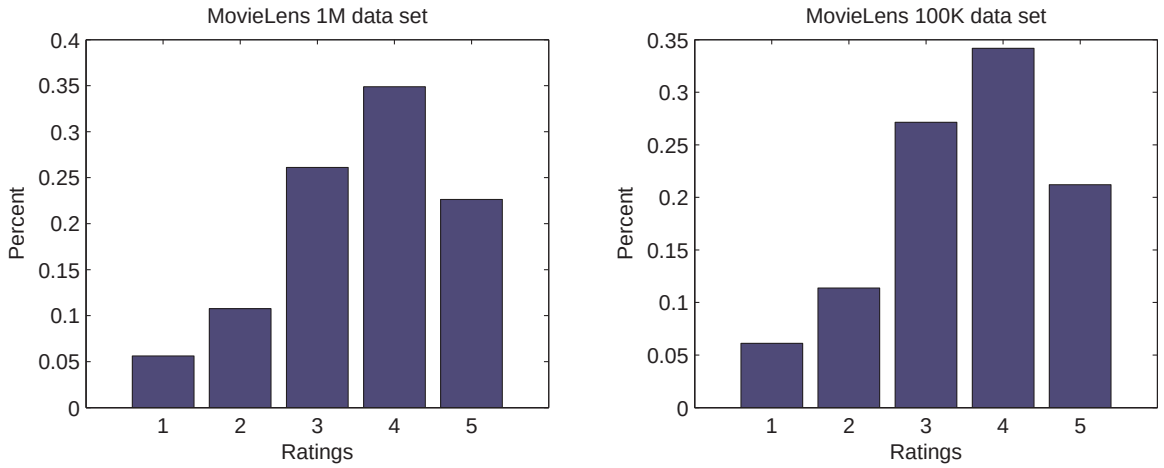


Figure 3.3: Percentage of each rating for ML 1M and ML 100K

3.2 Methodology

The following models are used in our experiments. Model (1) is referred as the *baseline* approach, and model (2,3,4) are known as LFMs, as discussed in Chapter 2.

(1). $m + b_u + b_i$

(2). $\mathbf{p}_u \cdot \mathbf{q}_i$

(3). $m + \mathbf{p}_u \cdot \mathbf{q}_i$

(4). $m + b_u + b_i + \mathbf{p}_u \cdot \mathbf{q}_i$

The experiments are mainly conducted on the MovieLens 1M dataset, while Movielens 100K data was only used to illustrate the effect of data preprocessing. All the experiments are carried in a manner of 10-fold cross validation. Therefore, the performance is computed as the average of the 10-fold test. For experiments on MovieLens 1M dataset, no significant change can be observed when the dimension for latent features exceeds 25. Thus, the dimension for the latent space is set at $F = 25$. On the other hand, the learning rate and regularization constants is fine tuned by cross-validation. Specifically, $\gamma = 0.005$, $\lambda_1 = \lambda_2 = 0.02$, $\lambda_3 = \lambda_4 = 0.03$. Besides that, we provide a simple, open-source, implementation of the above LFM methods, where we strive for conciseness and transparency.

3.3 Is Data Preprocessing Necessary?

Data preprocessing has always been an important step for a data mining task. The torrent of data flooding our world stimulates the development of RS. At the same time, these datasets may be polluted by noisy, missing, and inconsistent data [21]. Chances are low-quality data will lead to low-quality recommendations. Thus, it is necessary to explore the impact of data preprocessing on the performance of LFMs.

Three data preprocessing methods are explored here, to process the dataset before inputting them into LFMs. Winsorization transforms the dataset by limiting extreme values in the data to reduce the effect of possibly spurious outliers [15]. Trimming also addresses the problem of possible outliers by excluding some of the extreme values [38]. Imputation aims at reducing the sparseness of the dataset by replacing missing data with substituted values [40].

The experiments are conducted on the MovieLens $100K$ dataset. 10-fold cross validation has been carried and therefore the performance is computed as the average of the 10-fold test. In order to evaluate the impact of different data preprocessing approaches, model 4 is used here and the parameters for the following experiments are fixed. Therefore the dimension of user and item latent features is kept at 25, i.e. $F = 25$. $\gamma = 0.005$, $\lambda_1 = \lambda_2 = 0.02$, $\lambda_3 = \lambda_4 = 0.03$. .

It is worth noting that data preprocessing is only applied to the training set. To be specific, seven data preprocessing approaches have been discussed in this experiment. Hence, eight different training datasets are prepared here.

1. **Original**: Original dataset of MoviLens $100K$ data.

2. **Imputation with user's average rating (Impute-uave)**: Impute the missing entries, which do not appear in both the training set and test set, with the average rating of the corresponding user. For example, if user 1 did not give any rating for item 3, either in the training set or the test set. Then r_{13} is imputed with the average rating of user 1.

3. **Imputation with item's average rating (Impute-iave)**: Impute the missing entries for each item with the average rating computed from the corresponding items.

4. **Imputation with both user's and item's average rating (Impute-uiave)**: Impute the missing entries with the average rating of the corresponding users and items.

5. **Trimming based on user's rating distribution (Trim-u)**: For each user, the ratings fall below the 5th percentile or above the 95th percentile of that user's rating list are discarded.

6. **Trimming based on item's rating distribution (Trim-i)**: For each item, the ratings fall below the 5th percentile or above the 95th percentile of that item's rating list are removed.

7. **Winsorization based on user's rating distribution (Winsorize-u)**: For each user, the ratings fall below the 5th percentile of that user's rating list are set to the 5th percentile. And ratings above the 95th percentile are set to the 95th percentile.

8. **Winsorization based on item's rating distribution (Winsorize-i)**: Similarly to **winsorize-u**, for each item, the ratings fall below the 5th percentile of that item's rating list are set to the 5th percentile. And ratings above the 95th percentile are set to the 95th percentile.

Figure 3.4 shows the performance of LFMs on both the original dataset and 7 preprocessed dataset. Compared to the original dataset and the other preprocessed data, RMSEs change very slowly with the increase of N, the iteration number, for the imputed data. Also, the RMSEs of the imputed data keep decreasing even when the iteration number is large, while at the same time other RMSEs have begun to increase. However, this cannot change the fact that imputed datasets maintain a relatively

larger RMSE, which indicates a worse performance. To sum up, it is expected that it is much more difficult to achieve similar performance for imputed datasets.

The results from original dataset have demonstrated larger precision, recall and accuracy than the preprocessed data set. Particularly, the behaviour of LFMs on trimmed dataset and winsorized dataset is quite similar to the original data. In terms of the ability of adjusting to the change of iteration number N, it is hard to tell the difference between the original dataset and trimmed or winsorized dataset.

On the other hand, LFMs on the imputed datasets share the same pattern for their performance. Compared to that on the original dataset and other preprocessed datasets, LFMs do not work quite well on three imputed datasets. Also, for the imputed datasets, changes regarding model's performance are consistent with the increase of iteration number N. No significant change can be observed from RMSE. But we can see a dramatic change on precision, recall and accuracy when N increases from 10 to 20.

Therefore,the three imputation measures lead to worse performance of LFMs, with much higher RMSEs and lower precision, recall and accuracy. That is probably because denser rating matrix obtained from imputation brings noise to the dataset and thus makes the performance even worse. On the other hand, the performance of **Impute-uiave** is better than that of **Impute-uave**, which means that we can obtain more information for users and items in **Impute-uiave** dataset.

Trimming and winsorization also failed in this experiment. As shown in Figure 3.4, the original data has the best performance. Therefore, extreme values are also important in the rating prediction task and they cannot be treated as outliers. This partly proves previous finding that high/low ratings were most important to the recommender[12, 7]. Hence it is even more difficult to detect and deal with outlier in the data set.

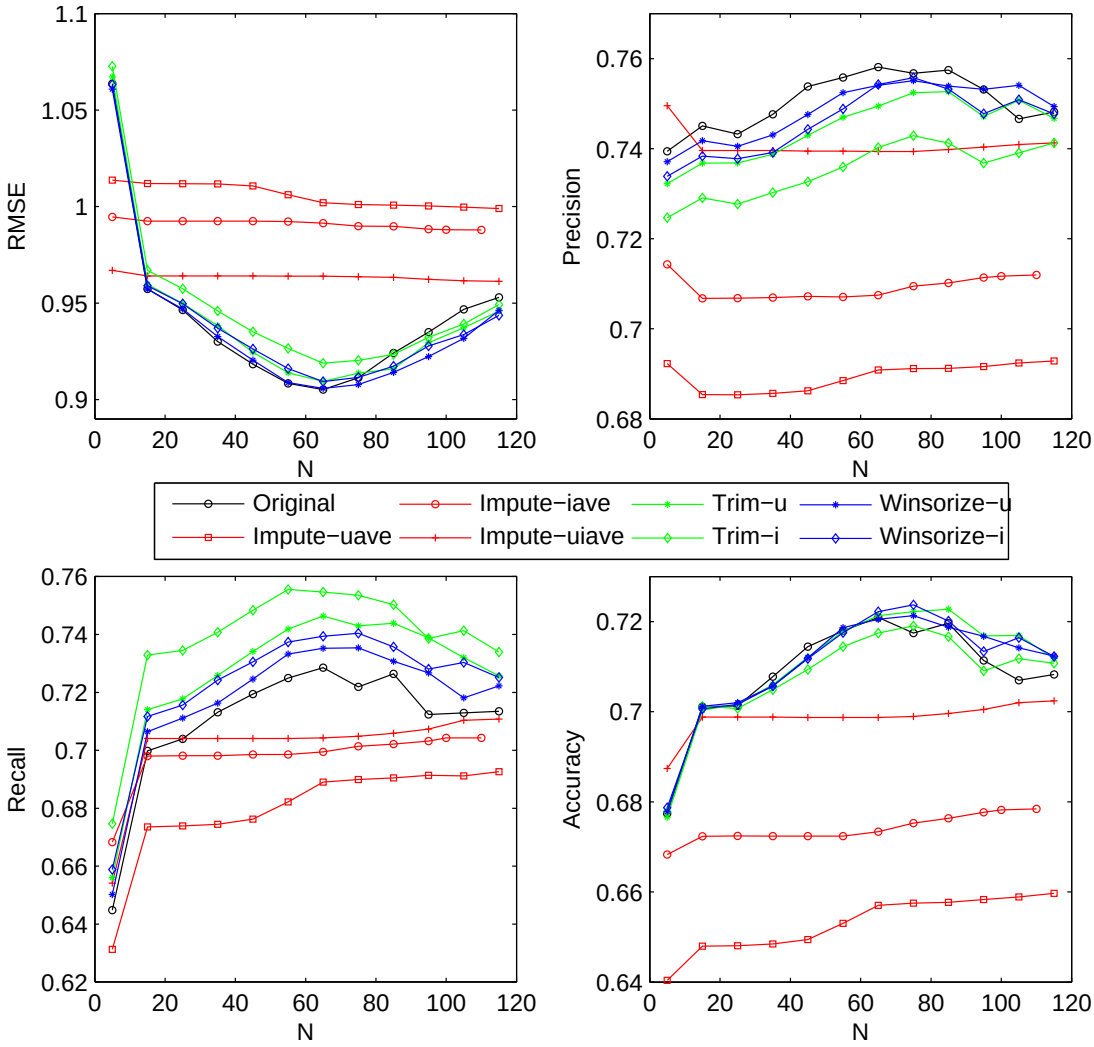


Figure 3.4: The impact of data preprocessing

3.4 Learning Algorithms: SGD or ALS?

Both SGD and ALS can be used to solve the objective function for LFMs. As a matter of convenience, SGD and ALS in this section means implementation of LFMs with the corresponding updating rule. Generally speaking, SGD is faster than ALS and less likely to overfit. Also, SGD can be easily implemented. But SGD is hard to tune with more free parameters. On the contrary, as shown in Chapter 2, ALS has less parameters. The updates of $\mathbf{p}_u$ and $\mathbf{q}_i$ can be parallelized while parallelizing ALS, which is very useful for dealing with large scale datasets [67]. In addition, for situations where SGD is not practical, ALS is a better choice [49, 24].

In order to explore the difference between SGD and ALS in detail, the effect of parameters will be discussed in this section, i.e. one parameter among the three parameters, λ, F and N, was kept fixed while the other two parameters were changed.

First, the evolution of the accuracy of SGD and ALS was studied when the number of iterations N and regularization parameter λ was changed. The results for SGD are shown in Figure 3.5, while those for ALS in Figure 3.6. In this experiment, the learning rate $\gamma = 0.005$ and the number of hidden featuress was fixed at 8. The latent factor model used here was method (3).

As observed in Figure 3.5, SGD requires a large N and a small λ to achieve a good performance. All the performance improves as the increase of the number of iterations. To be specific, RMSE is decreasing and precision, recall and accuracy are increasing with the increase of iteration number under different λ settings. Since the performance keeps improving, the expected number of iterations should be greater than 25. As for the change of λ, method (3) with SGD has a better performance at $\lambda = 0.03$.

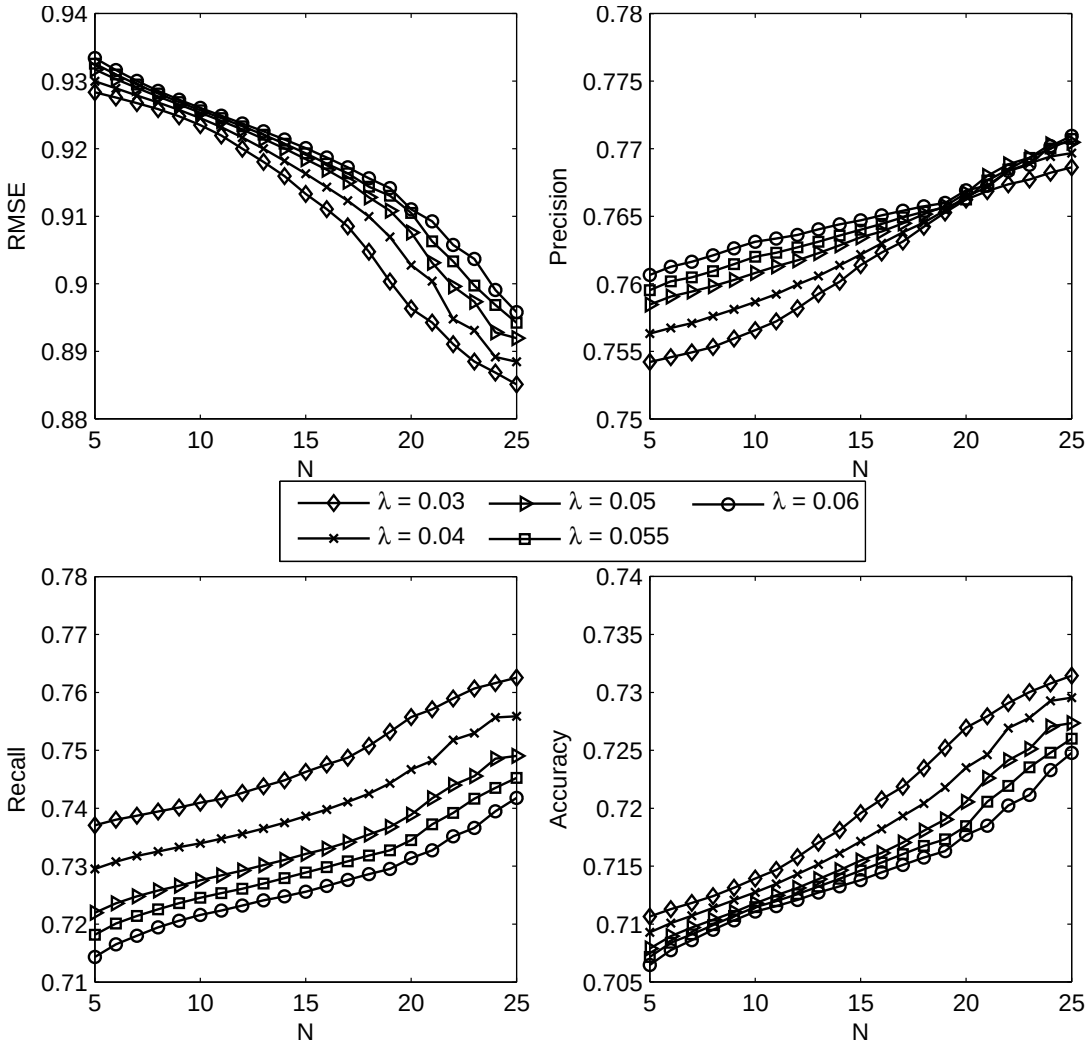


Figure 3.5: Comparison of different λ values for SGD with $F = 8$.

ALS generally has a good performance even when the iteration number is less than 25, as demonstrated in Figure 3.6. Similarly, as the number of iterations increases, RMSE decreases with different λ for ALS. But the change diminishes very quickly when N exceeds 10. RMSE changes dramatically from 0.87 to 0.855 when N increases from 5 to 10. However, the change of RMSE reduces to less than 0.005 while N is over 15.

For ALS, recall and accuracy increase while precision decreases with the change of N. If the focus of the recommender system is on RMSE and precision, a small iteration number and a larger λ, i.e. $\lambda = 0.06$ is more favourable. However, if the overall performance also includes recall and accuracy, the parameters are difficult to decide. The reason lies into the fact that recall and accuracy behave better at a

smaller λ value, with $\lambda = 0.03$ in this experiment.

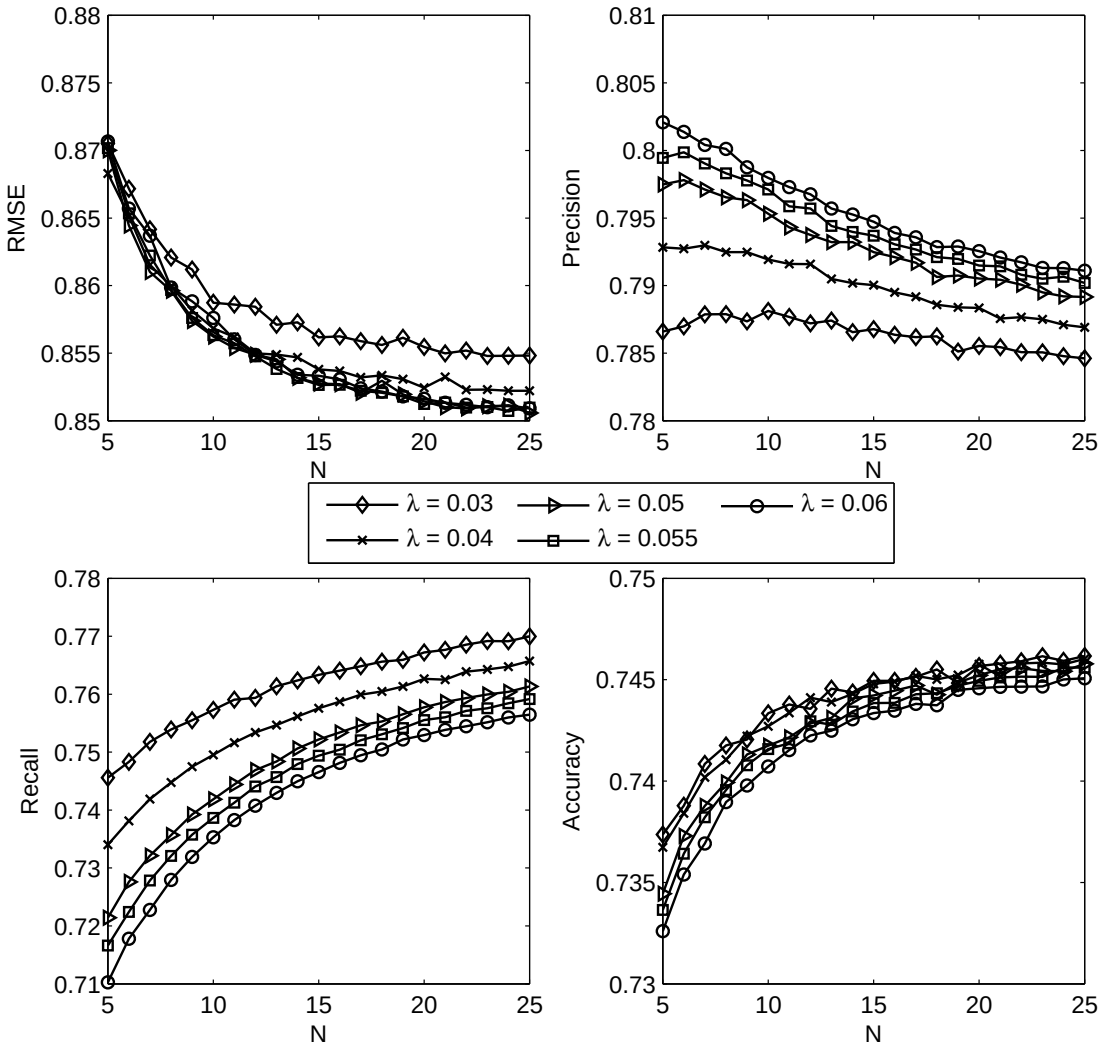


Figure 3.6: Comparison of different λ values for ALS with $F = 8$.

Under the same setting, ALS performs better than SGD, as shown in Figure 3.5 and Figure 3.6. A relatively large number of iterations is expected while applying SGD method. And a smaller value of λ is preferred for SGD. On the other hand, the selection of parameters for ALS is a little tricky because of its complex behaviour with N and λ. Trade-off should be made based on the focus of evaluation metrics. A larger value of λ and N can lead to a lower RMSE, but can not ensure other performance, precision for example.

In fact, ALS and SGD can achieve similar performance in spite of their different behaviour with the change of N. As shown in Figure 3.7, both methods converge at

larger number of N. The change of RMSE, recall and accuracy of SGD is consistent with that of ALS. They are getting better as N increases. The differences in terms of RMSE, precision and accuracy among SGD and ALS decrease with the increase of N. Thus, these facts show that there is not any fundamental difference between SGD and ALS when it comes to the actuate results achieved by the method.

However, it should be noted that SGD and ALS have different responses to the changes of parameters. Precision of ALS keeps decreasing as N increases. The performance of SGD is getting better at the same time, but still cannot be as good as that of ALS. On the other hand, SGD beats ALS at recall measurement. As mentioned before, the performance of SGD keeps improving with the increase of N. While ALS converges earlier than SGD and its precision decreases for large number of iterations. In this case, small N, maybe 25, is enough for ALS. More iterations are expected for SGD.

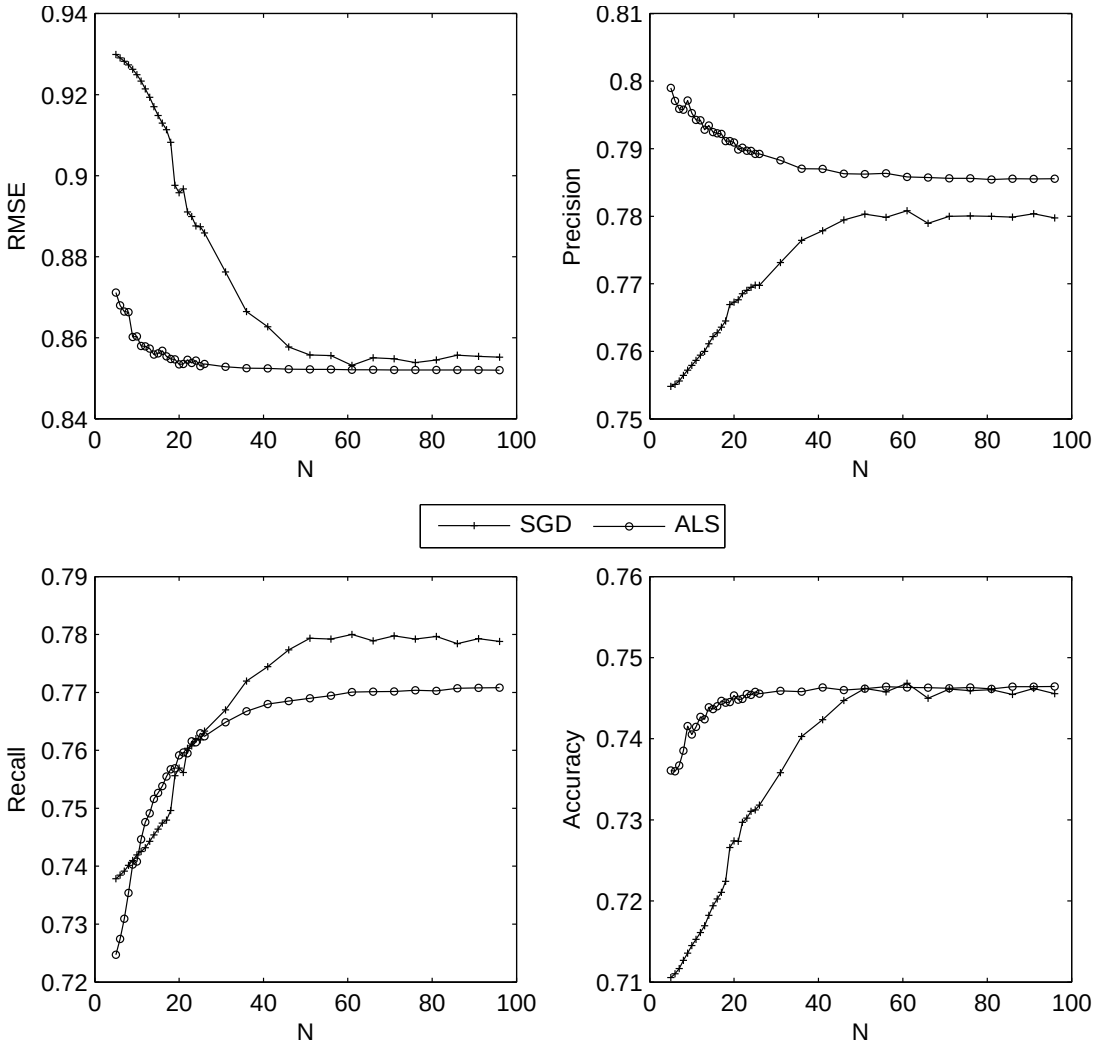


Figure 3.7: Comparison of ALS and SGD with $F = 8$.

The effect of latent factors is explored with iteration number being fixed at 25. The performance of SGD and ALS is discussed with fixed λ value and varying number of hidden features(F ranges from 2 to 20).

Figure 3.8 shows that more features give more accurate results for SGD method. With different λ, RMSE decreases and precision, recall and accuracy increase as F increases. However, the improvement starts to diminish when F is larger than 10. Therefore, the value of F should be around 15. A larger F might bring a better performance, but it is not worthy at the cost of time and other computing resources.

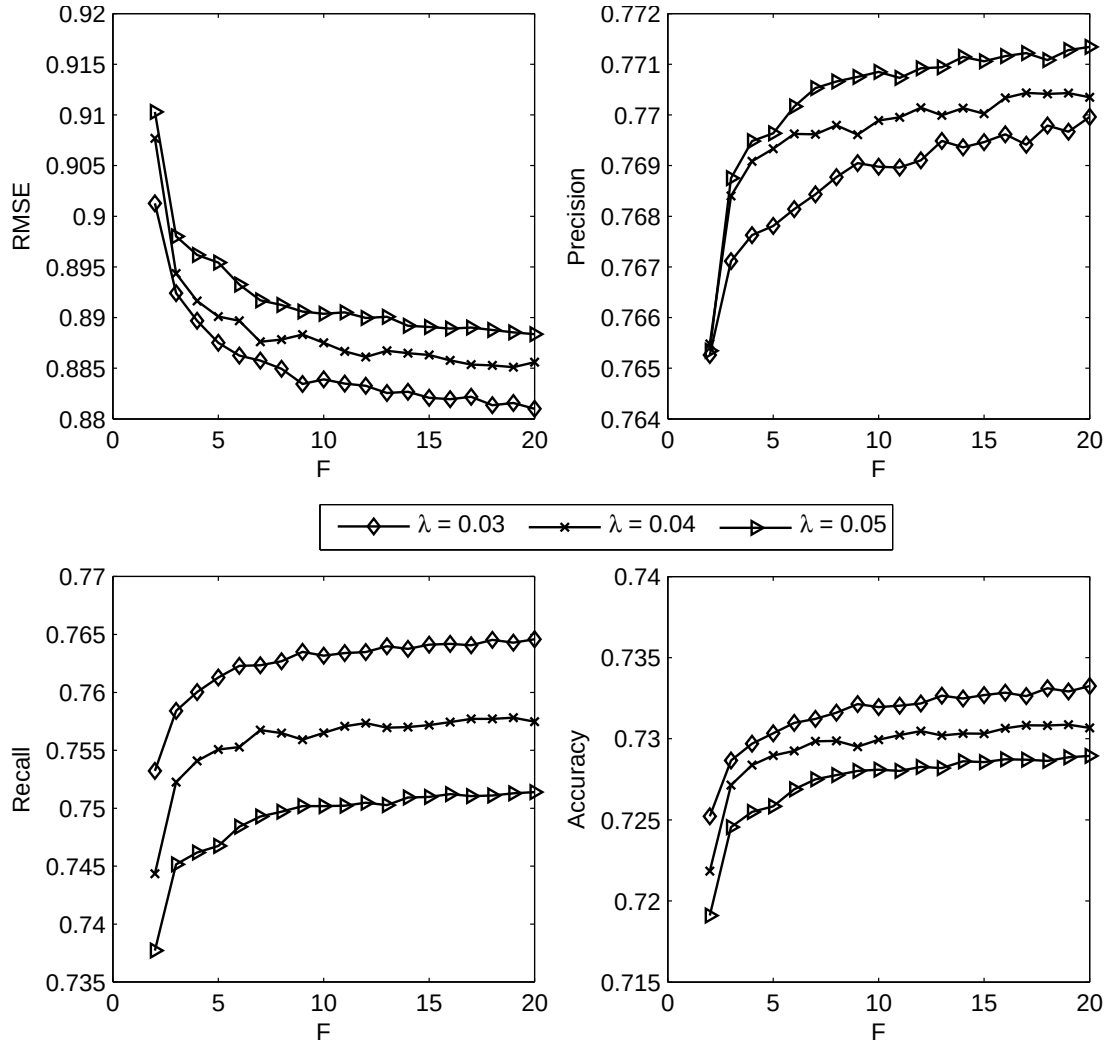


Figure 3.8: Comparison of different λ and F values for SGD with $N = 25$.

ALS has complex behaviour with regards to the change of F in Figure 3.9. The performance is improving when F changes within the range between 1 and 10. And the performance starts to get bad when F is higher than 15. This observation is much more obvious for $\lambda = 0.03$.

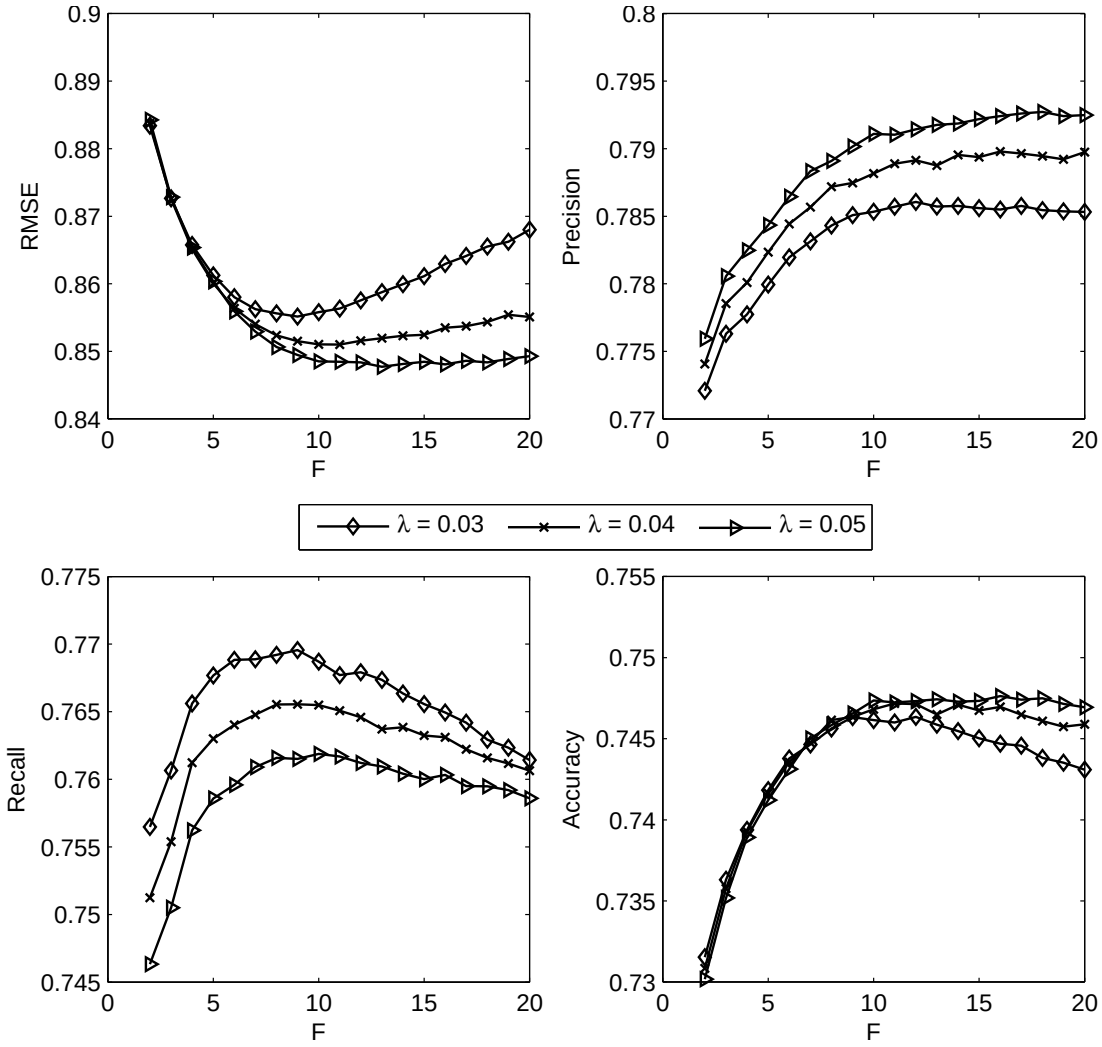


Figure 3.9: Comparison of different λ and F values for ALS with $N = 25$.

Similar to the effect of N, SGD and ALS share some characteristics with the the change of λ and F. For both SGD and ALS, the performance improves very fast under 6 hidden features. SGD generally converges at higher number of F while ALS starts to over fit when F is large. The over-fitting is obvious with a smaller value of λ.

Finally, the prediction errors made from different updating rules are analysed. Per prediction error is the error related to each rating, while per user error is the error for each user. Figure 3.10 shows that there is a strong positive correlation between the error produced by SGD and ALS with the linear correlation coefficients over 0.9. As a result, there is no need to swing between updating with SGD and ALS for a more accurate results. The two methods not only have similar overall performance,

but also behave similarly at individual level.

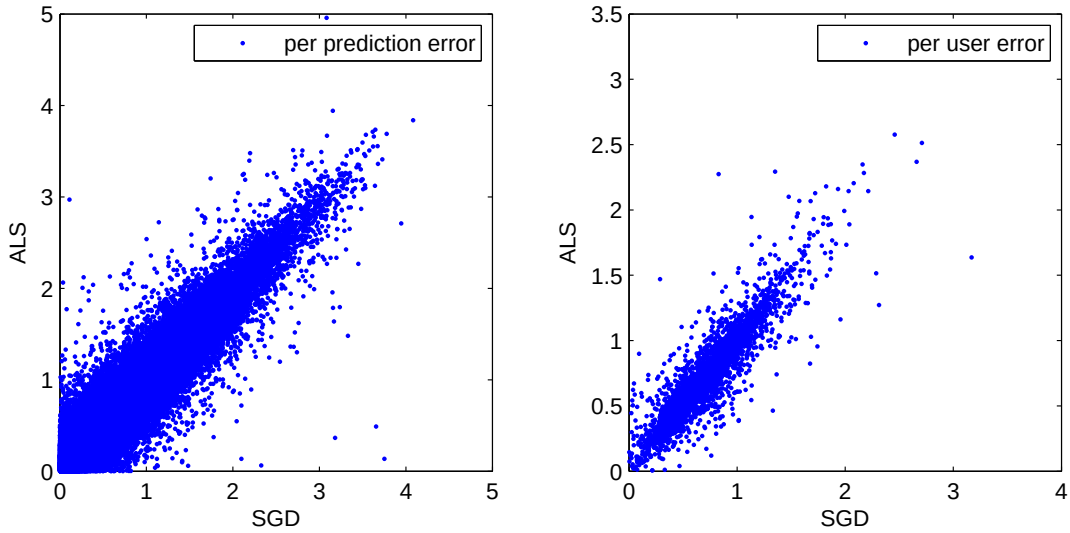


Figure 3.10: The error between SGD and ALS

In conclusion, updating rules, SGD and ALS, do not have an impact on the performance of latent factor models. Generally, the implementation of SGD is straight forward and it is not hard to achieve the best performance. While ALS has potential for advanced use, dealing with large scale data set and parallel computing, for example.

Chapter 4

Performance of LFMs

4.1 Overall Performance of LFMs

First, the overall performance of LFMs on MovieLens 1M dataset is studied. The results are shown in Figure 4.1, 4.2, 4.3, 4.4 and 4.5. Models in these figures refer to model (1), (2), (3) and (4) in Chapter 3. Results below indicate that LFMs have advantages over other methods in terms of prediction accuracy. RMSE of LFMs is found to be less than baseline methods. At the same time, precision, recall, F measure and accuracy of LFMs maintain a higher degree.

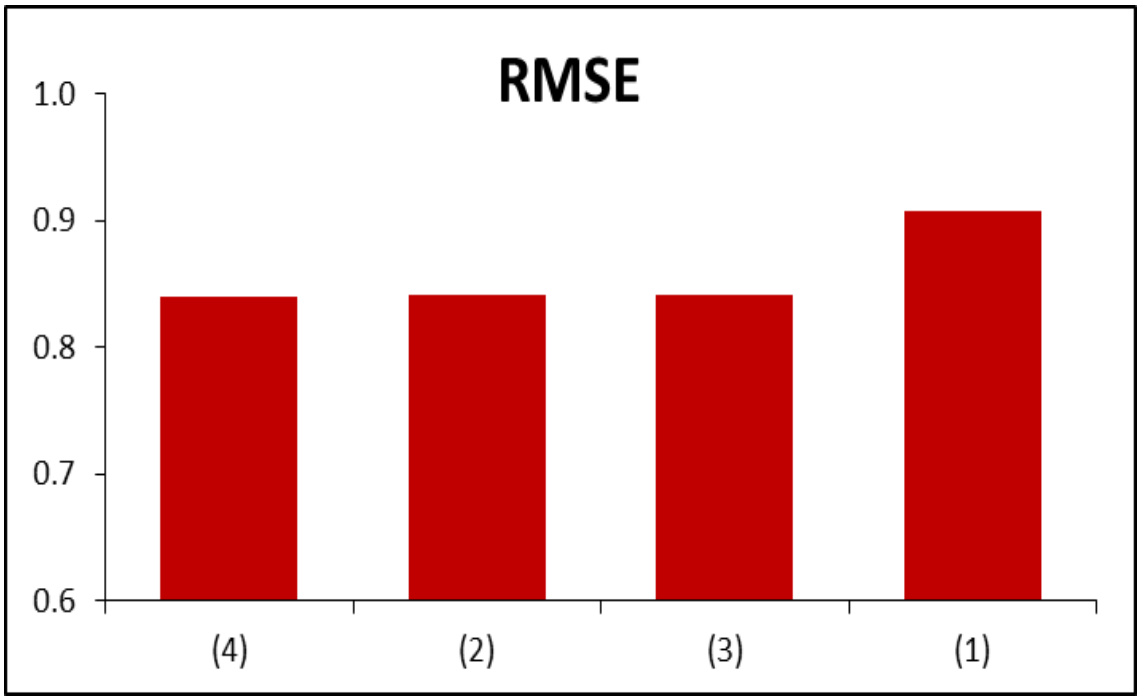


Figure 4.1: Overall RMSE for MovieLens1M dataset

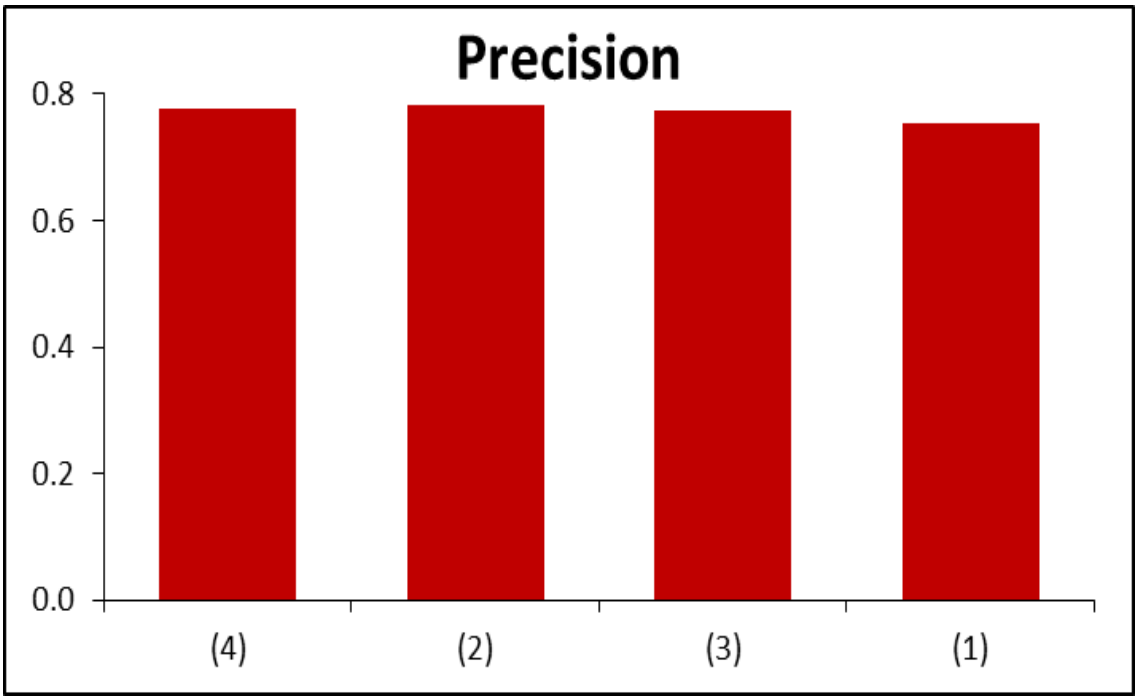


Figure 4.2: Overall precision for MovieLens1M dataset

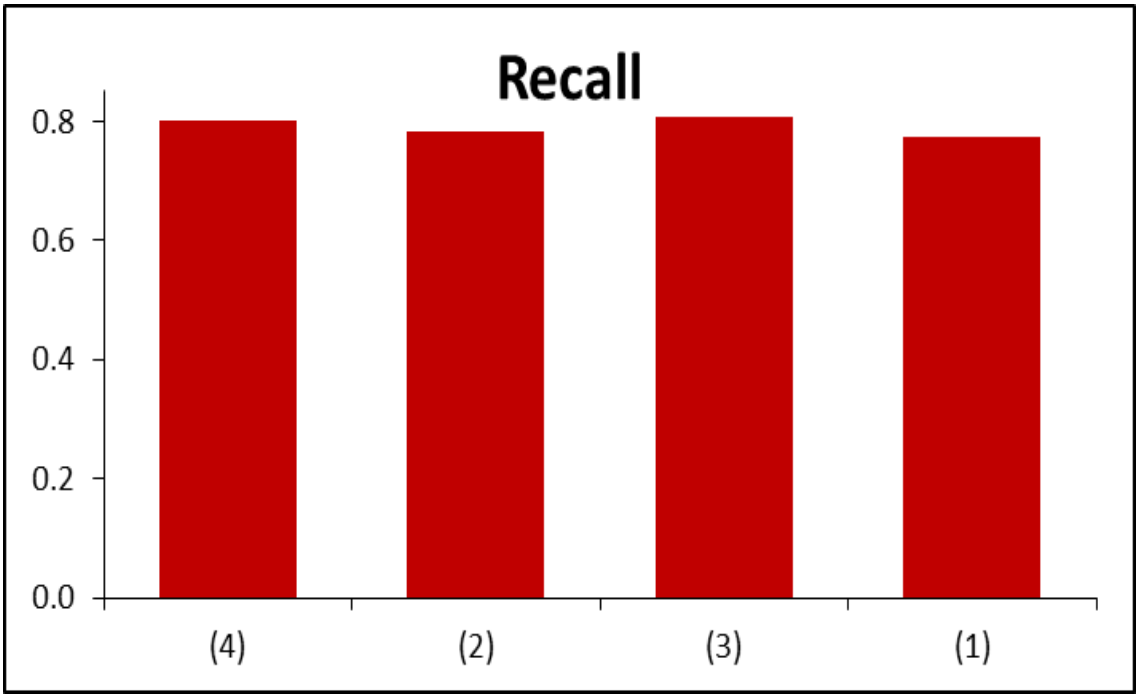


Figure 4.3: Overall recall for MovieLens1M dataset

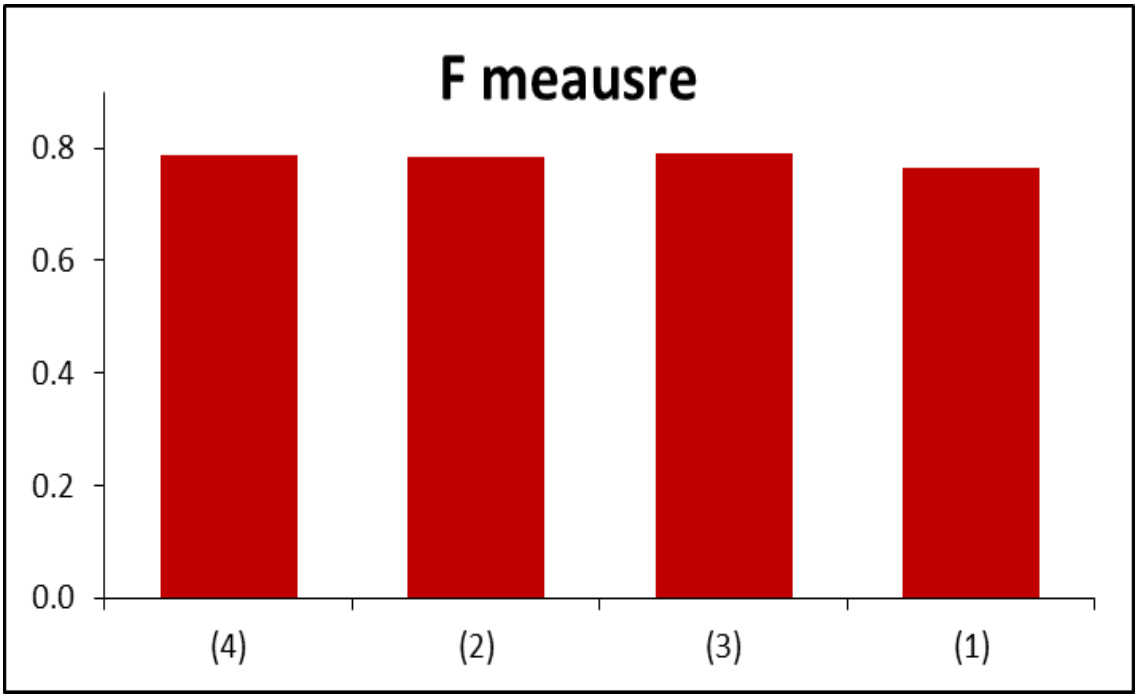


Figure 4.4: Overall F meansure for MovieLens1M dataset

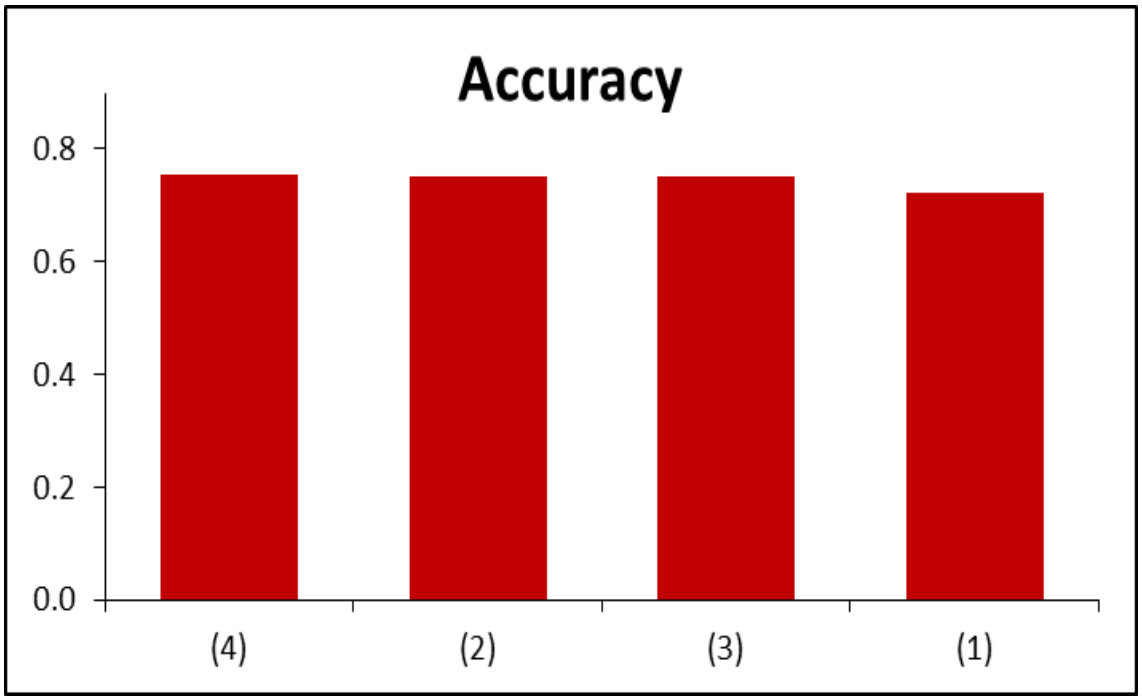


Figure 4.5: Overall accuracy for MovieLens1M dataset

Good prediction is used to evaluate the algorithm's performance for each prediction. As suggested by [16], "good prediction" is defined as getting a prediction right within some threshold, such as 0.5 stars. Therefore, the absolute error between a good prediction and the real rating is less than 0.5 stars.

According to Table 4.1, the performance of LFMs and baseline method is quite similar at individual prediction level. Each entry in the table represents the percentage of common good predictions that the corresponding row and column share. For example, entry at the first row and second column shows that method (1) and method (2) share 79.67% good predictions, which proves that these two methods have a lot in common. Further more, LFMs, with about 90% common good predictions, are also similar to each other in terms of getting predictions right. These results demonstrate that the ability of LFMs is constrained.

Table 4.1: Good predictions in common

	(1)	(2)	(3)	(4)
(1)	-	79.67%	79.80%	82.07%
(2)	73.21%	-	88.34%	91.61%
(3)	73.19%	88.17%	-	90.01%
(4)	74.56%	90.58%	89.17%	-

LFMs beats baseline in the overall performance, but they do not ensure an improvement for each individual prediction. More specifically, Figure 4.6 shows the scatter plots of the absolute error of the baseline against the LFMs over the whole test set. Absolute error is equal to the absolute difference between the true rating and the predicted one. We can observe that the points on the plots clearly demonstrate a positive correlation between the baseline and three LFMs, suggesting that if a test case is difficult for the baseline (the absolute error is high), it is not necessarily easier for the LFMs.

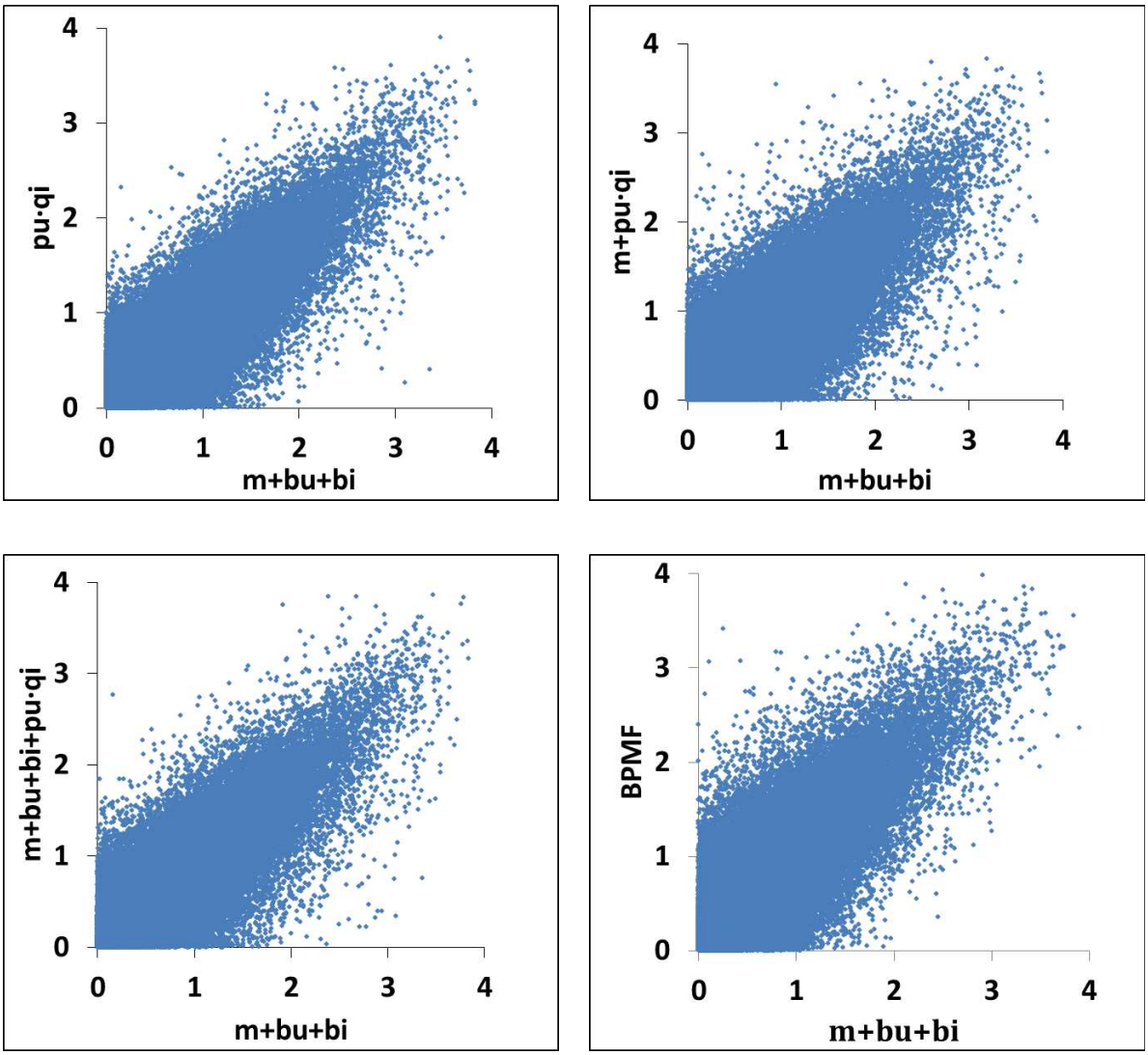


Figure 4.6: Absolute error correlation between baseline and LFMs

4.2 Performance in Different User/Item Categories

4.2.1 User and Item Categories

However, the performance picture changes in quite interesting ways once performance of particular categories of users and items are explored. Similarly as [45] suggests, user and item categories for the MovieLens 1M dataset are defined as follows.

Heavyrater users(HR) who provided more than 270 ratings.

Opinionated users(OP) who provided more than 135 ratings and whose standard deviation of ratings is larger than 1.2.

Blacksheep users(BS) who provided more than 135 ratings and for which the average distance of their ratings from the mean rating of the corresponding items is greater than 1.0.

Coldstart users(CS) who provided no more than 135 ratings.

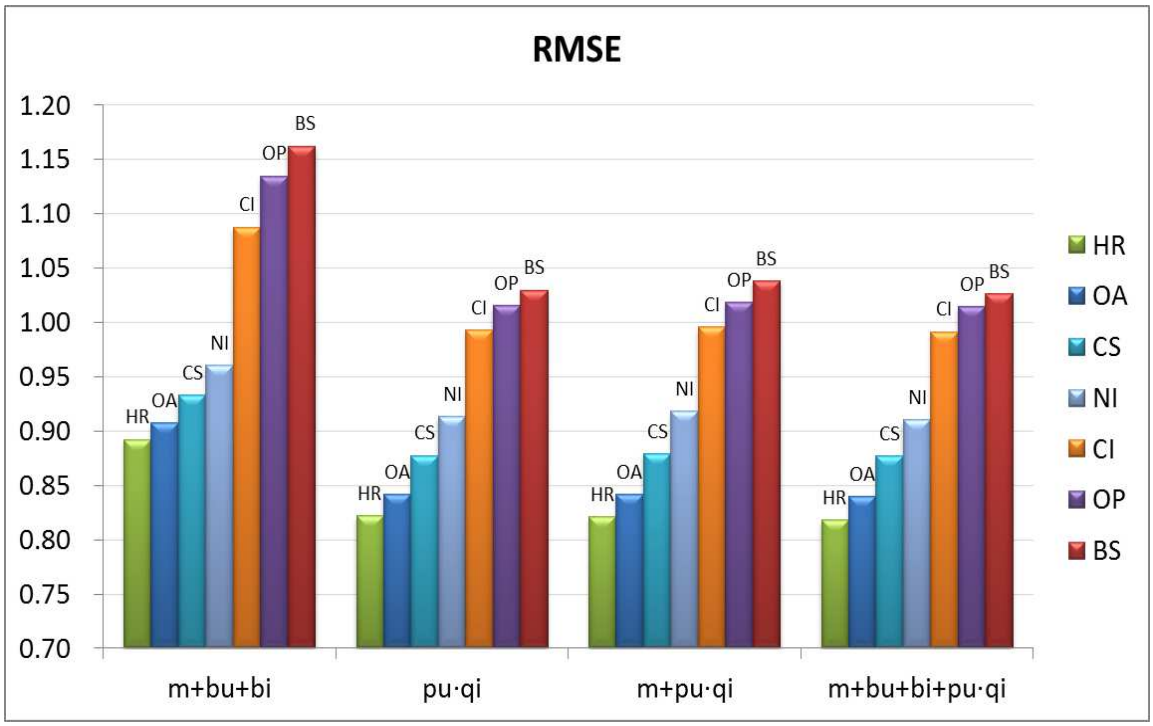
Controversial items(CI) which received ratings whose standard deviation is larger than 1.1.

Niche items(NI) which received less than 135 ratings.

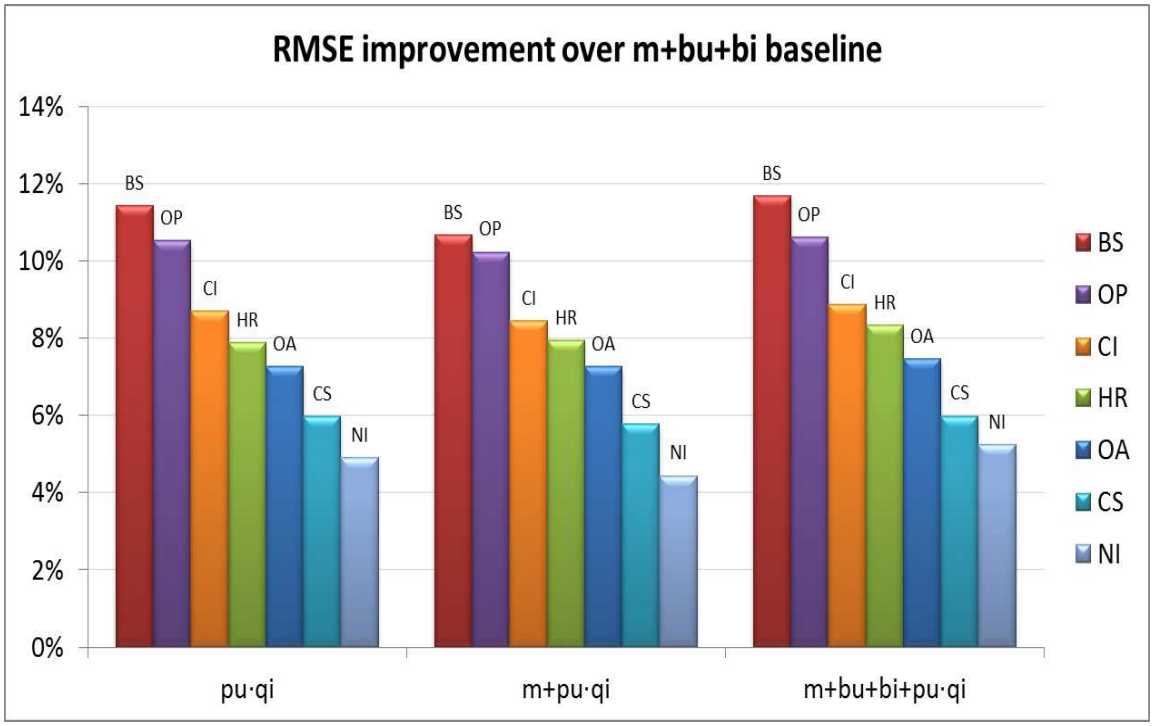
The reasoning of using the above numbers is as follows. MovieLens 1M is a very dense dataset. 270 is the median of the number of each user's ratings in the training set. 135, which is half of 270, is chosen as the threshold for **Coldstart users** as well as **Niche items**. This number covers approximately half of the total users and items, respectively. The choices of standard deviation were inspired by [45]. Similar groups have also been used in [13].

4.2.2 Performance in Different Categories

Figures 4.7 - 4.11 and conclusions drawn from them are described below. In these figures, **OA** stands for overall performance.

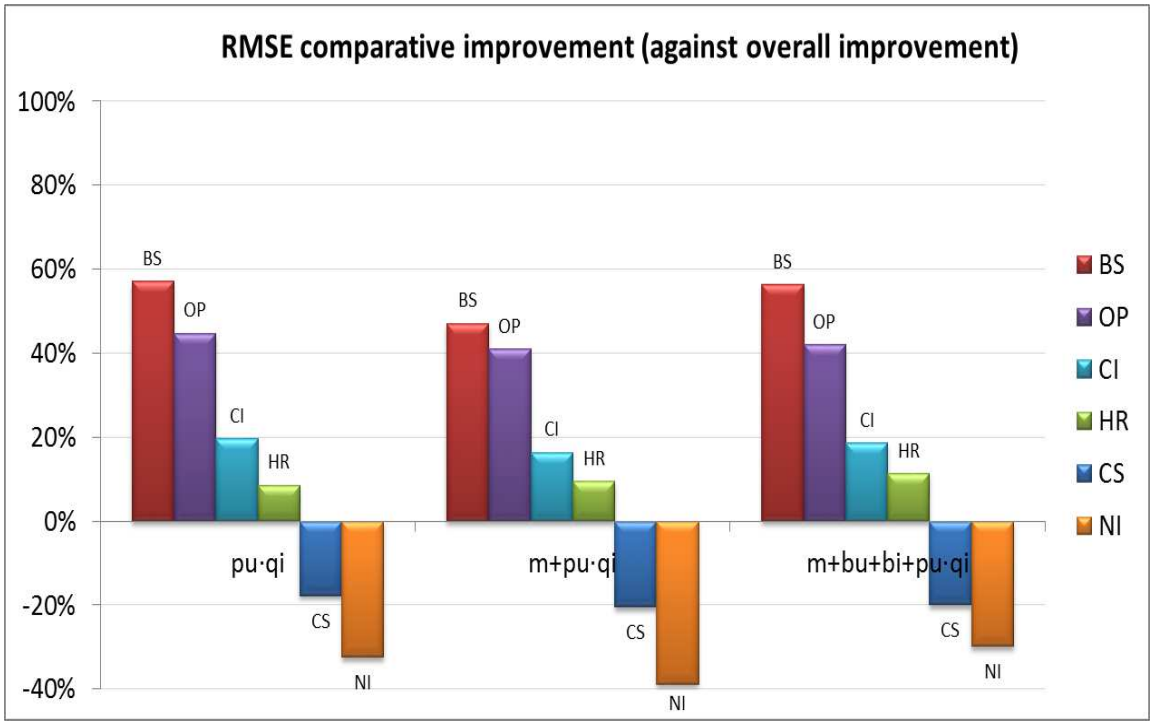


(a)



(b)

Figure 4.7: RMSE ML



(c)

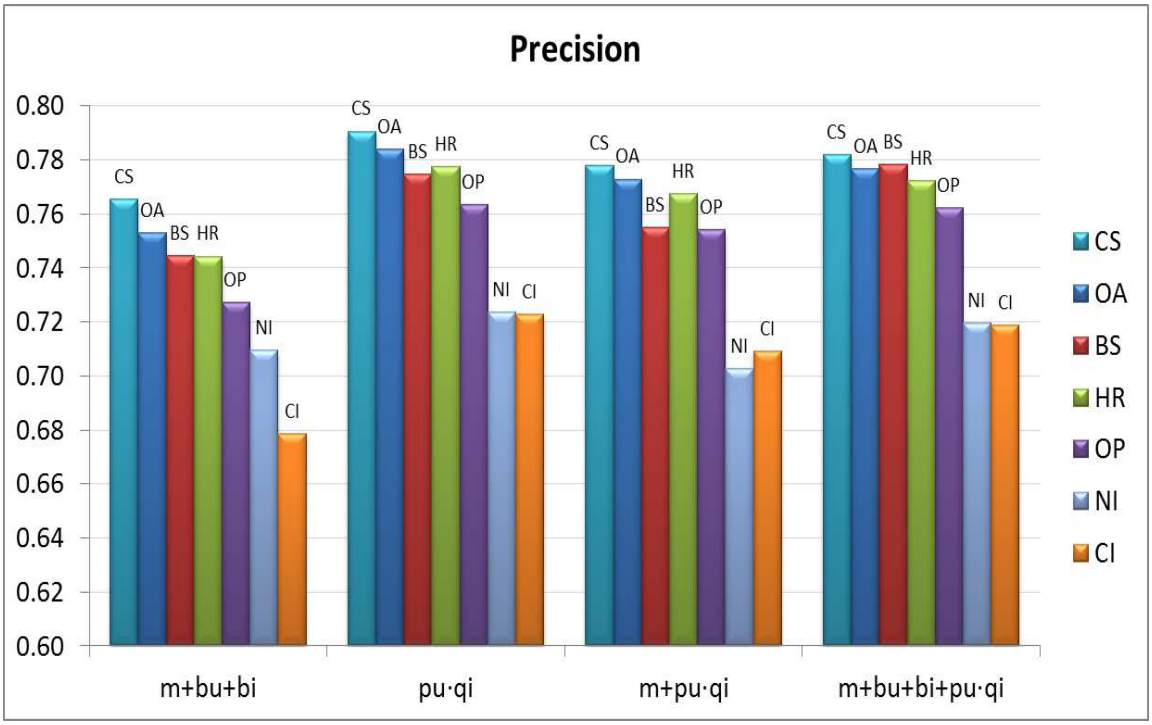
Figure 4.7: RMSE ML

The RMSE values for each category are shown in Figure 4.7 (a). The more on the left a bar is, the smaller (better) the RMSE is. As shown in the figure, the most difficult categories in terms of RMSE are Blacksheep users, Opinionated users, and Controversial items. The improvements for $\mathbf{p}_u \cdot \mathbf{q}_i$ (2), $m + \mathbf{p}_u \cdot \mathbf{q}_i$ (3), $m + b_u + b_i + \mathbf{p}_u \cdot \mathbf{q}_i$ (4) over the baseline $m + b_u + b_i$ (1) are quite clear.

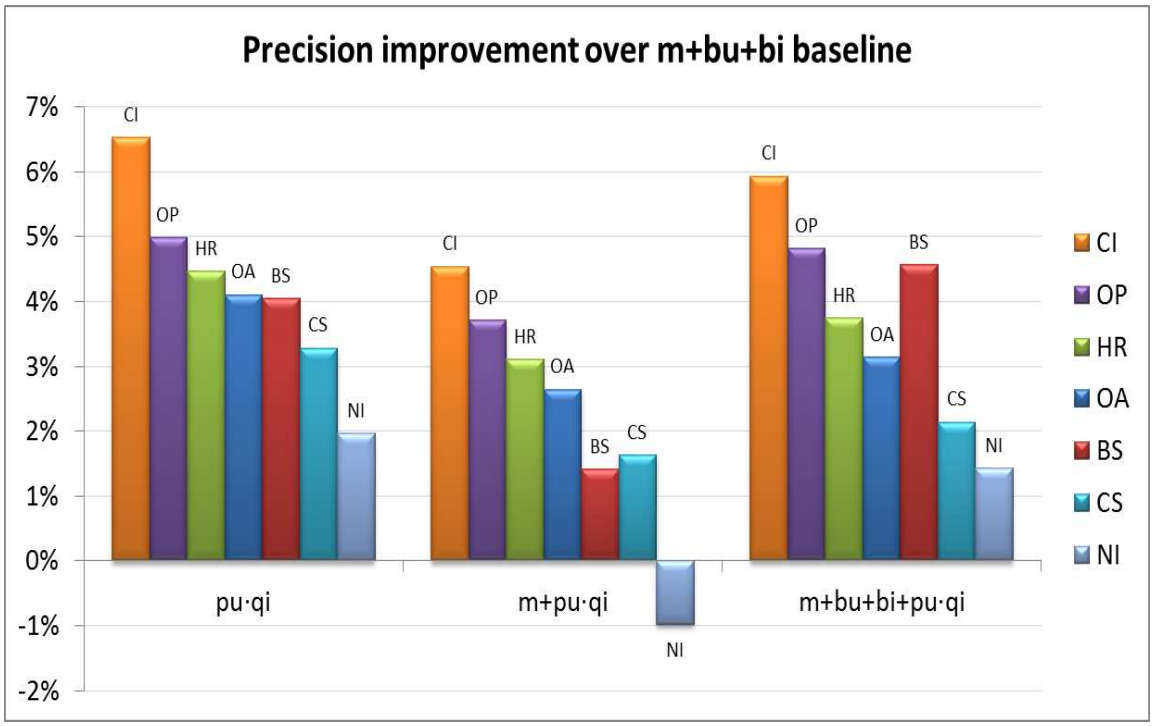
Figure 4.7 (b) gives the amount of improvement over the baseline. The improvement for Blacksheep users is more than 11% and for the Opinionated users is more than 10%. This is quite impressive compared to the fact that the Neflix competition was for improving the RMSE by 10%. The only categories that are far worse than the overall are Coldstart users and Niche items. Meanwhile, the difference in improvement between (2), (3) and (4) is not too pronounced.

Figure 4.7 (c) shows how much greater is the improvement for (2), (3) and (4), for each category, compared to the overall improvement. For example, for Blacksheep users, the improvements for (2) and (4) are close to 60% greater than the overall improvement.

The Precision values for each category are shown in Figure 4.8 (a). Some surpris-



(a)



(b)

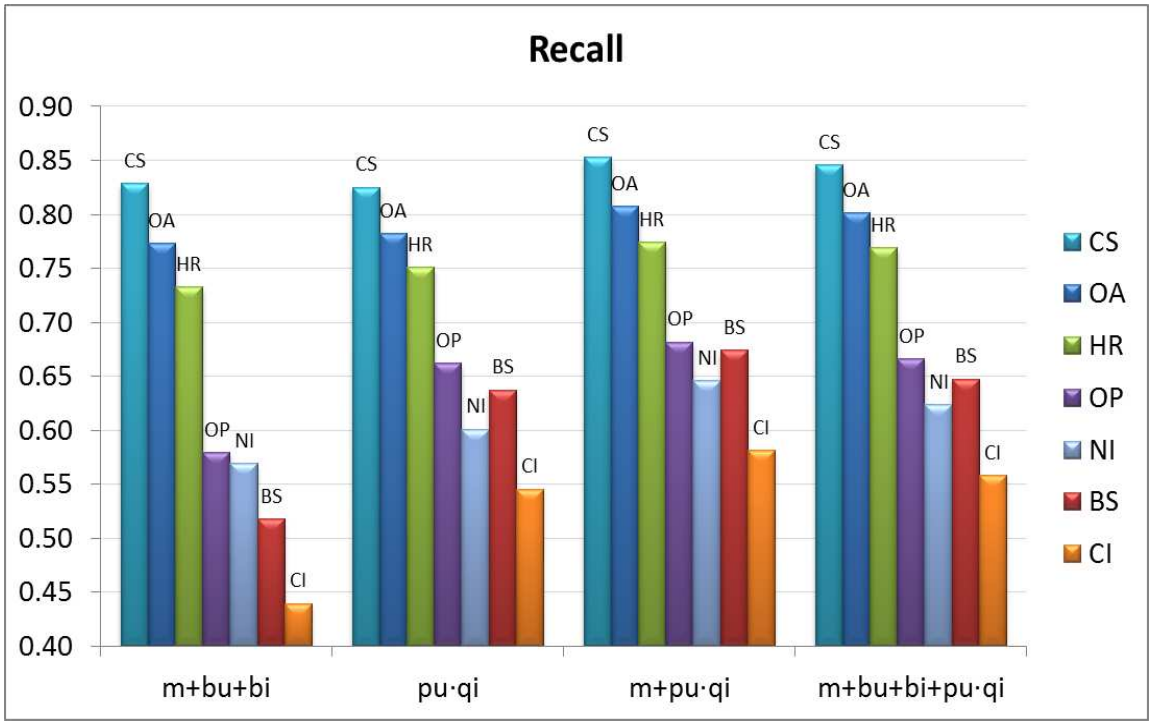
Figure 4.8: Precision ML

ing facts can be observed here, such as the precision for Coldstart users (a difficult category for RMSE) being the highest for all approaches, and especially for (2), which is more than 79%. Also, Blacksheep, Heavyrater, and Opinionated users have a higher precision in all four models. The improvements for (2), (3) and (4) are good (see Figure 4.8 (b)), but not as great as those for RMSE. The precision values for (2), (3) and (4) are better than those for the baseline except for (3) for Niche items. The largest improvement, of about 6.5%, is for (2) for Controversial items.

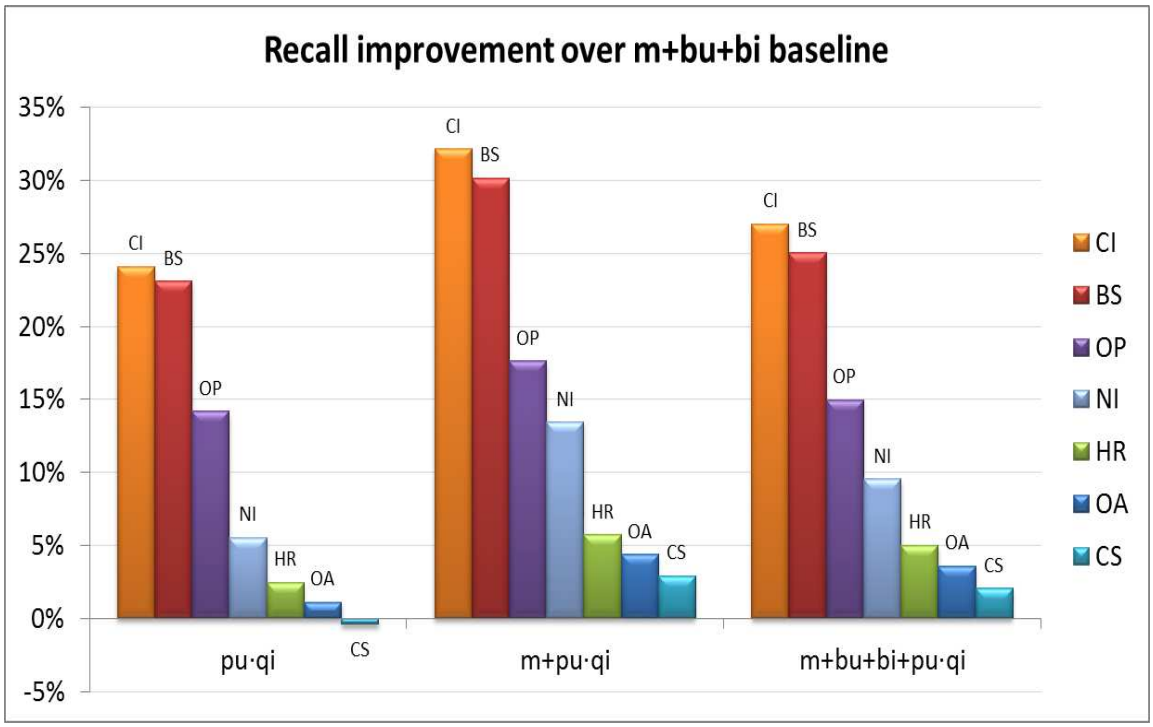
The recall values for each category are shown in Figure 4.9 (a). For example, recall for Coldstart users exhibits the highest value for all approaches, and especially for (2), which is (slightly) more than 85%. The improvements shown in the same figure (bottom), are impressive for some categories, such as those for (3) for Controversial items and Blacksheep users with bars exceeding 30%!

The F measure values for each category are shown in Figure 4.10 (a). Coldstart users still score the highest, with values of more than 0.8, for all approaches. Again, the improvements shown in the same figure (bottom), are impressive for some categories, such as that for (3), for Controversial items, which is close to 20%. Notably, all (2), (3) and (4) improve over the baseline (1) for all the categories.

Accuracy values are given in Figure 4.11 (a). All (2), (3) and (4) have similar accuracy values across all categories, with Blacksheep users faring better than the other categories. Accuracy improvements are shown in the same figure (b). The improvements shown here are similar to those for precision.

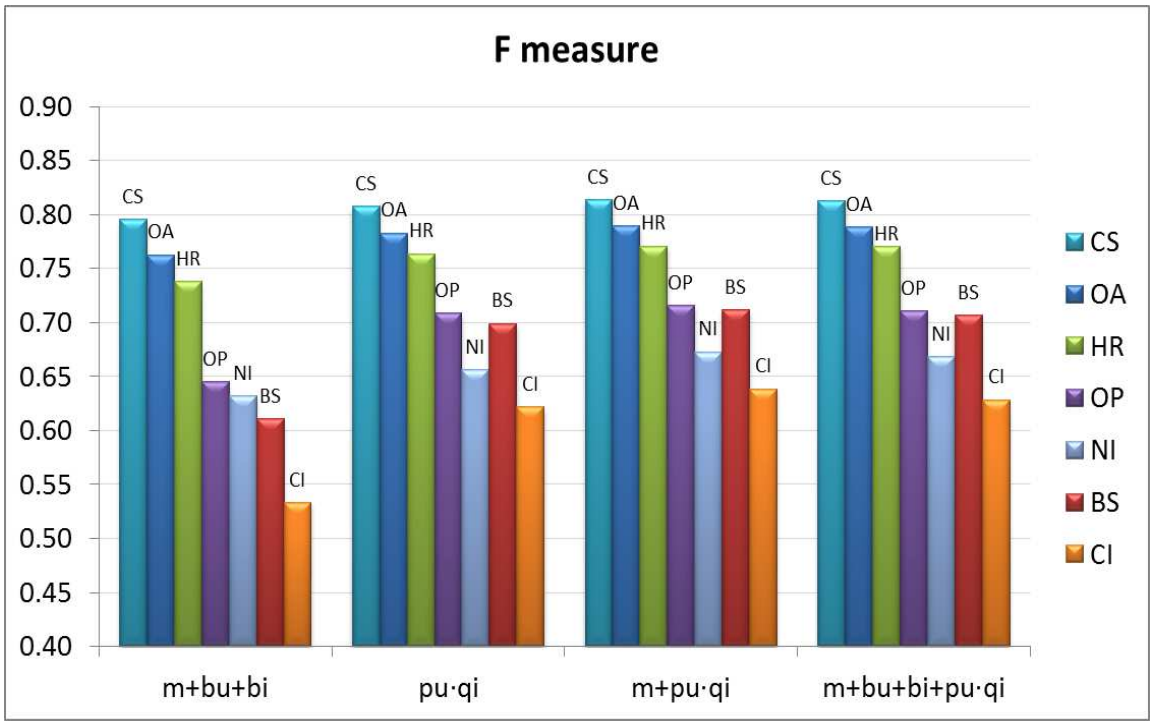


(a)

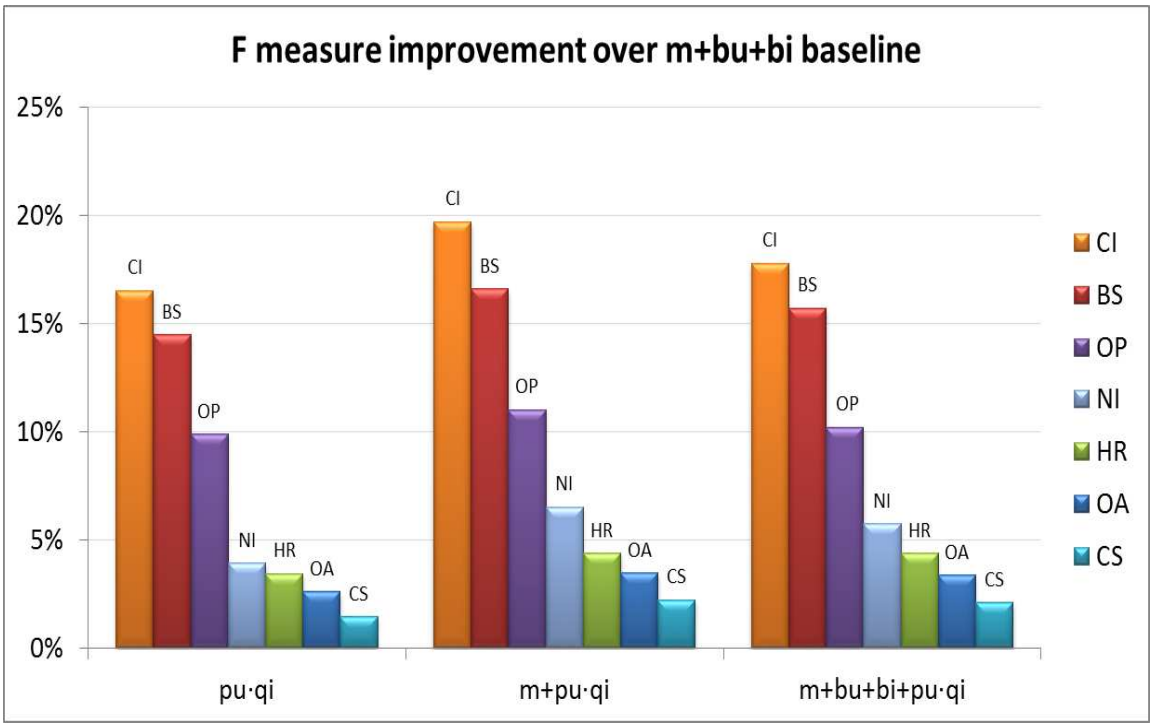


(b)

Figure 4.9: Recall ML

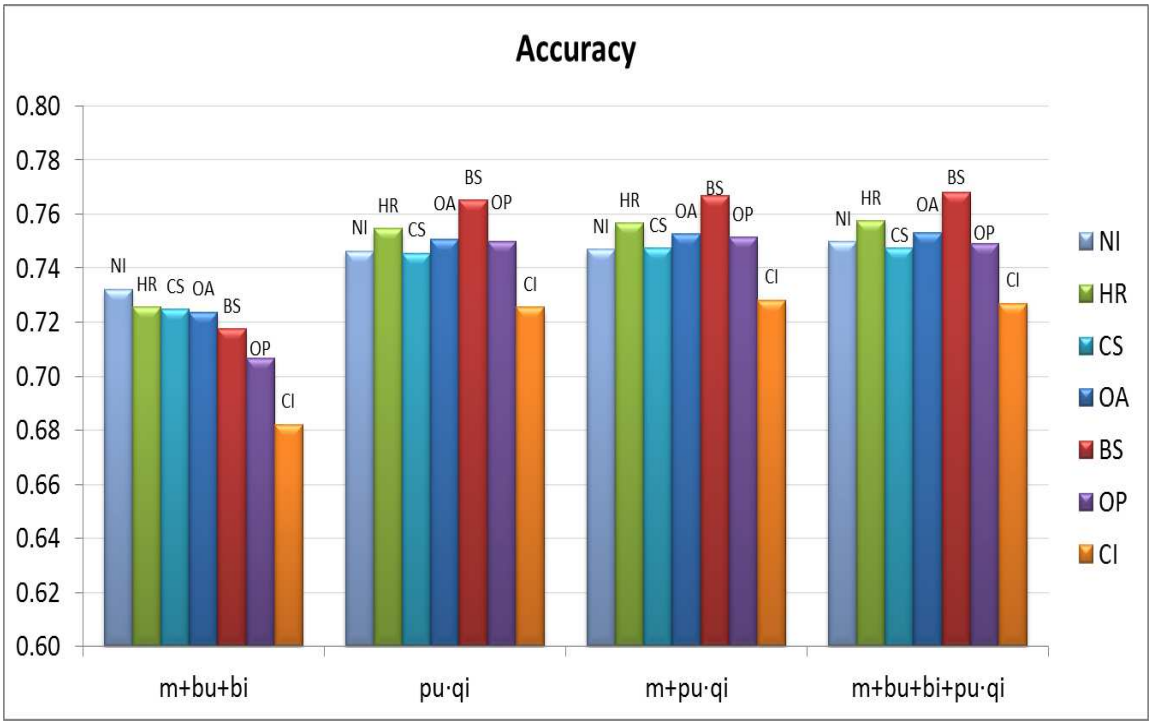


(a)

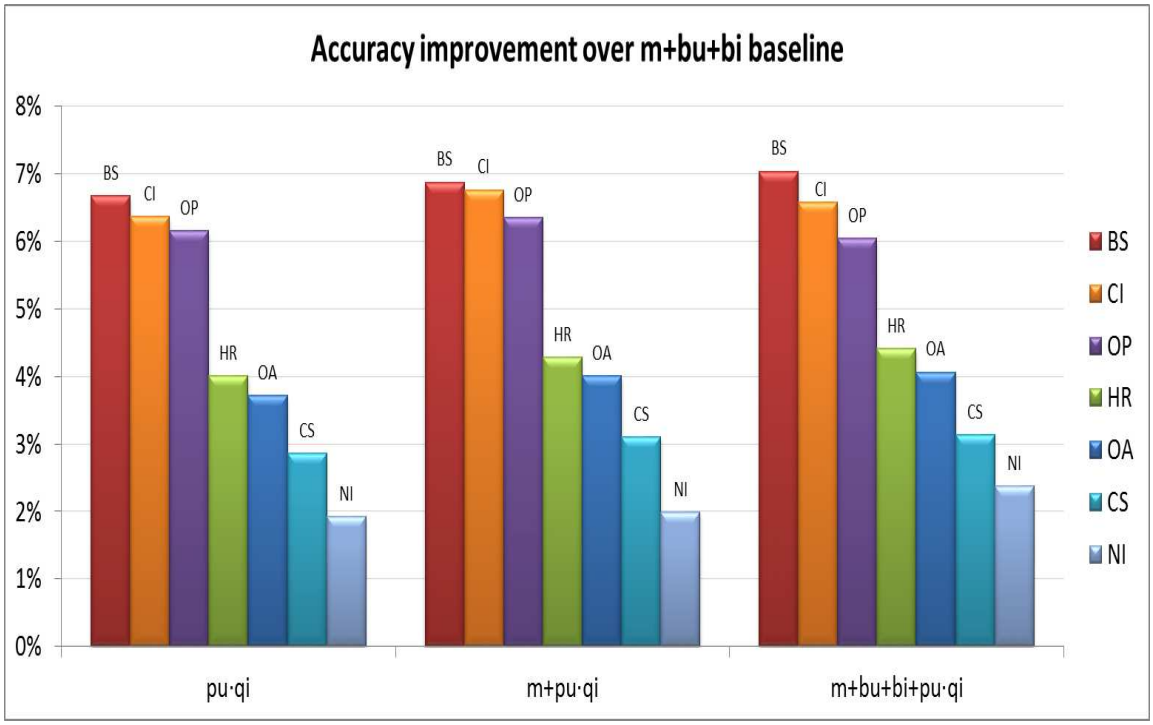


(b)

Figure 4.10: F measure ML



(a)



(b)

Figure 4.11: Accuracy ML

Chapter 5

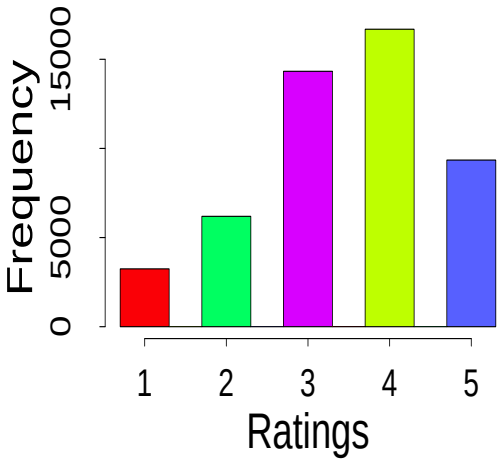
Relationship between Rating Distributions and LFMs

As illustrated in Chapter 4, the performance of LFMs is different for different user/item categories. The question is that what is the possible cause for the phenomenon. To the best of our knowledge, there hasn't been such fine-grained analysis of LFMs in the related research. Here the hypothesis is that the performance of LFMs is influenced by the distribution of ratings in different categories.

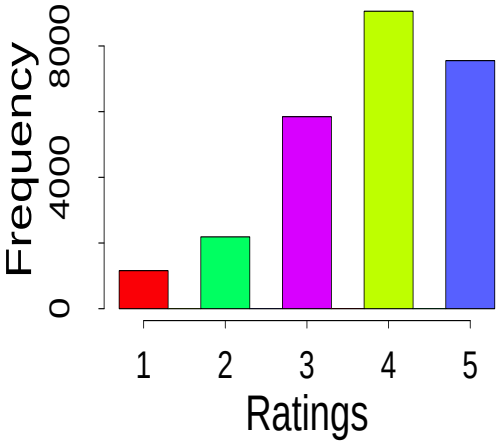
The relationship between RMSE and the distribution of ratings in different categories is discussed in this section. To be specific, the baseline method and three LFMs, $m + b_u + b_i$ (1), $\mathbf{p}_u \cdot \mathbf{q}_i$ (2), $m + \mathbf{p}_u \cdot \mathbf{q}_i$ (3) and $m + b_u + b_i + \mathbf{p}_u \cdot \mathbf{q}_i$ (4) are analyzed here. One train/test split of the 10-fold test is selected to demonstrate rating distributions of different categories. And only figures for test dataset are presented here because training dataset has similar distribution.

5.1 Rating Distributions in Real Data

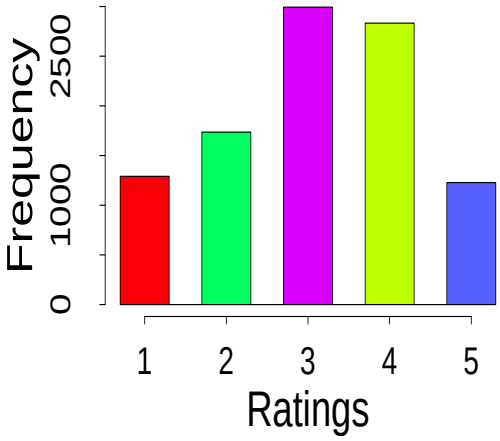
First, the histograms of the groundtruth test ratings for each category are shown in Figure 5.1.



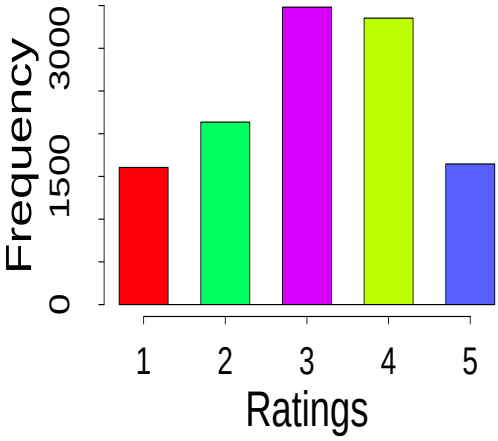
(a) Heavyrater Users



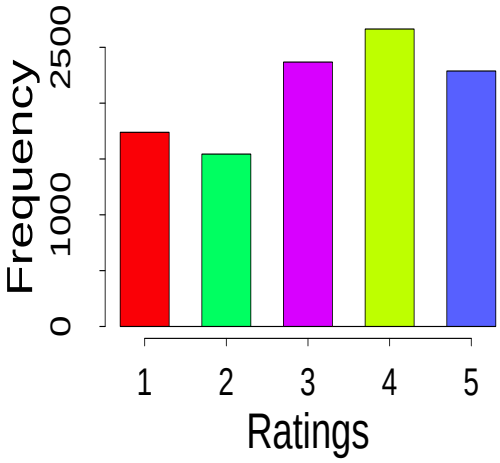
(b) Coldstart Users



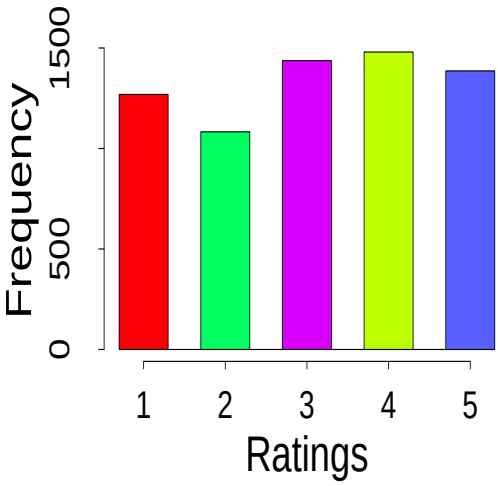
(c) Niche Items



(d) Controversial Items



(e) Opinionated Users

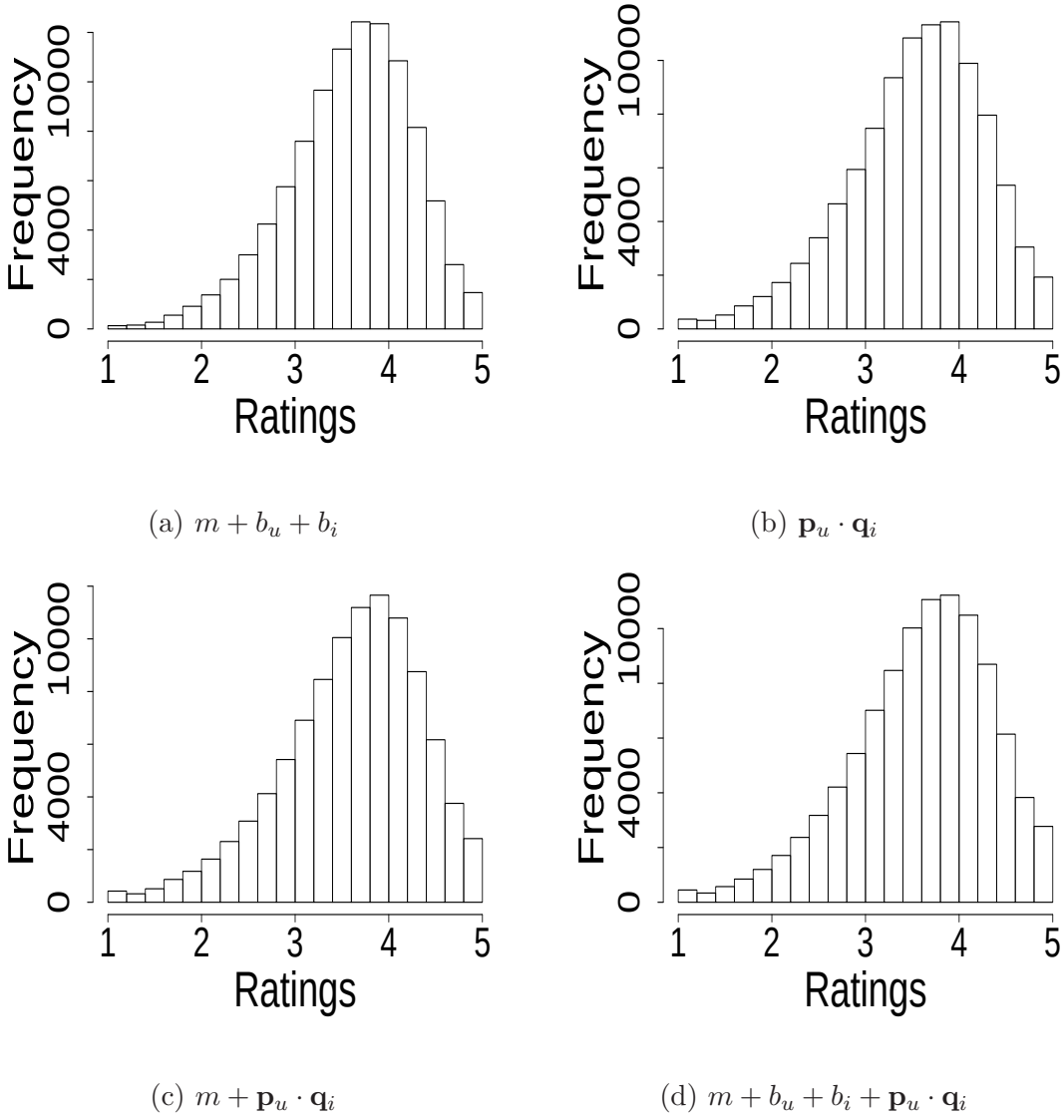


(f) Blacksheep Users

Figure 5.1: Histogram plots of groundtruth ratings for different categories

In Figure 5.1, ratings in Opinionated users and Blacksheep users are more uniformly distributed than the other four categories. Surprisingly, as proved before, LFMs have the worst RMSE for these two categories in Figure 4.7. This implies that the difference in rating distributions of different categories may result in different performance for LFMs.

In order to verify this hypothesis from the model side, histograms of the overall ratings produced by $m + b_u + b_i$ and three LFMs are plotted in Figure 5.2. According to the figure, the four models exhibit a very similar shape of the rating distribution. All of them have a truncated Gaussian-like distribution in the predictions, where the majority of ratings is located around the mean value. It indicates that if the ratings to be predicted are far from Gaussian distribution, then LFMs may fail to give good predictions.



(a) $m + b_u + b_i$

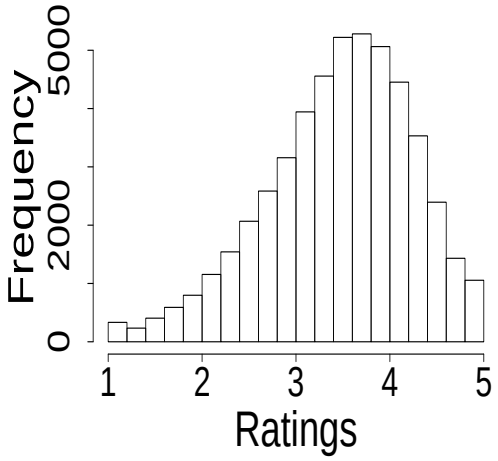
(b) $\mathbf{p}_u \cdot \mathbf{q}_i$

(c) $m + \mathbf{p}_u \cdot \mathbf{q}_i$

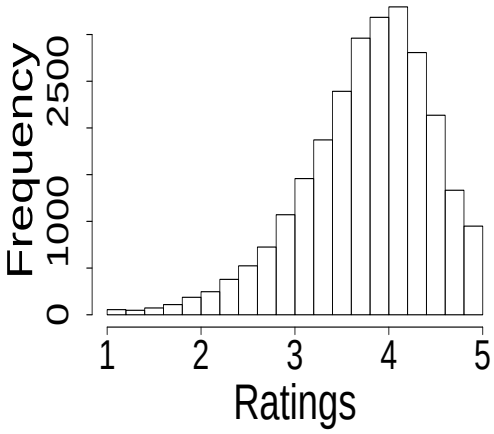
(d) $m + b_u + b_i + \mathbf{p}_u \cdot \mathbf{q}_i$

Figure 5.2: Histogram plots of predicted ratings for different models

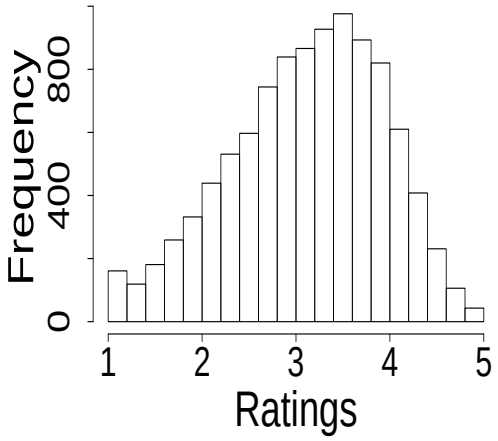
Then the histograms for the predicted ratings of each category produced by $m + b_u + b_i + \mathbf{p}_u \cdot \mathbf{q}_i$ (4) in Figure 5.3 are plotted. Since Figure 4.7 suggests that LFMs have the same relative performance for each category, $HR < OA < CS < NI < CI < OP < BS$ in terms of RMSE, model (4) is chosen as an objective example.



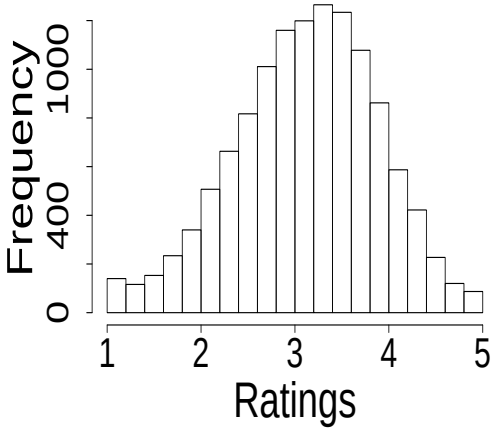
(a) Heavyrater Users



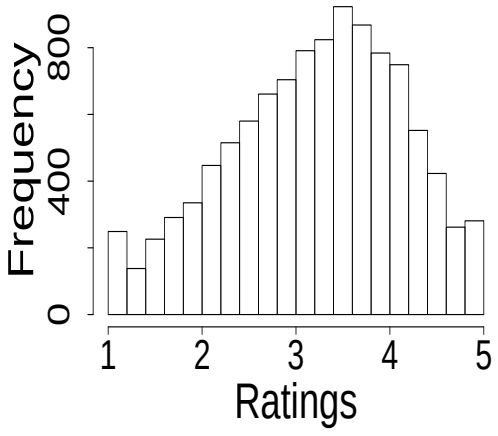
(b) Coldstart Users



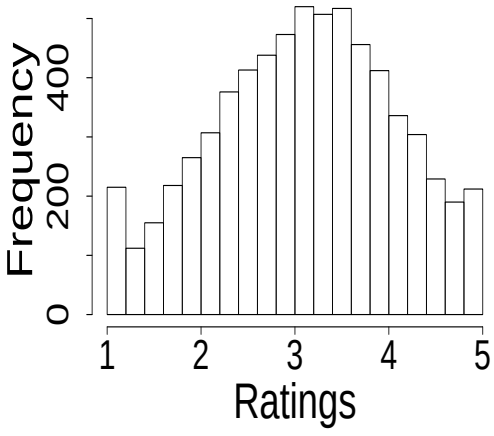
(c) Niche Items



(d) Controversial Items



(e) Opinionated Users



(f) Blacksheep Users

Figure 5.3: Histogram plots of ratings for different categories

Figure 5.3 shows that the predicted rating distribution for each category is similar. In other words, LFMs always output Gaussian-like rating distribution regardless of the type of category. To some degree, this fact has explained the bad performance of LFMs on preprocessed data, where the original data distribution has been changed, as illustrated in Figure 5.4 and Figure 5.5. The original training dataset includes $89,956$ records, while the imputation approaches add $1,470,113$ more records to it, which dramatically changed the rating distribution of the whole dataset. The majority of the original ratings are $3, 4$, and 5. However, after imputation, the new datasets mainly focus on ratings 3 and 4. On the other hand, trimming and winsorization does not change the rating distribution of the dataset, which enables them to achieve similar performance with the original one.

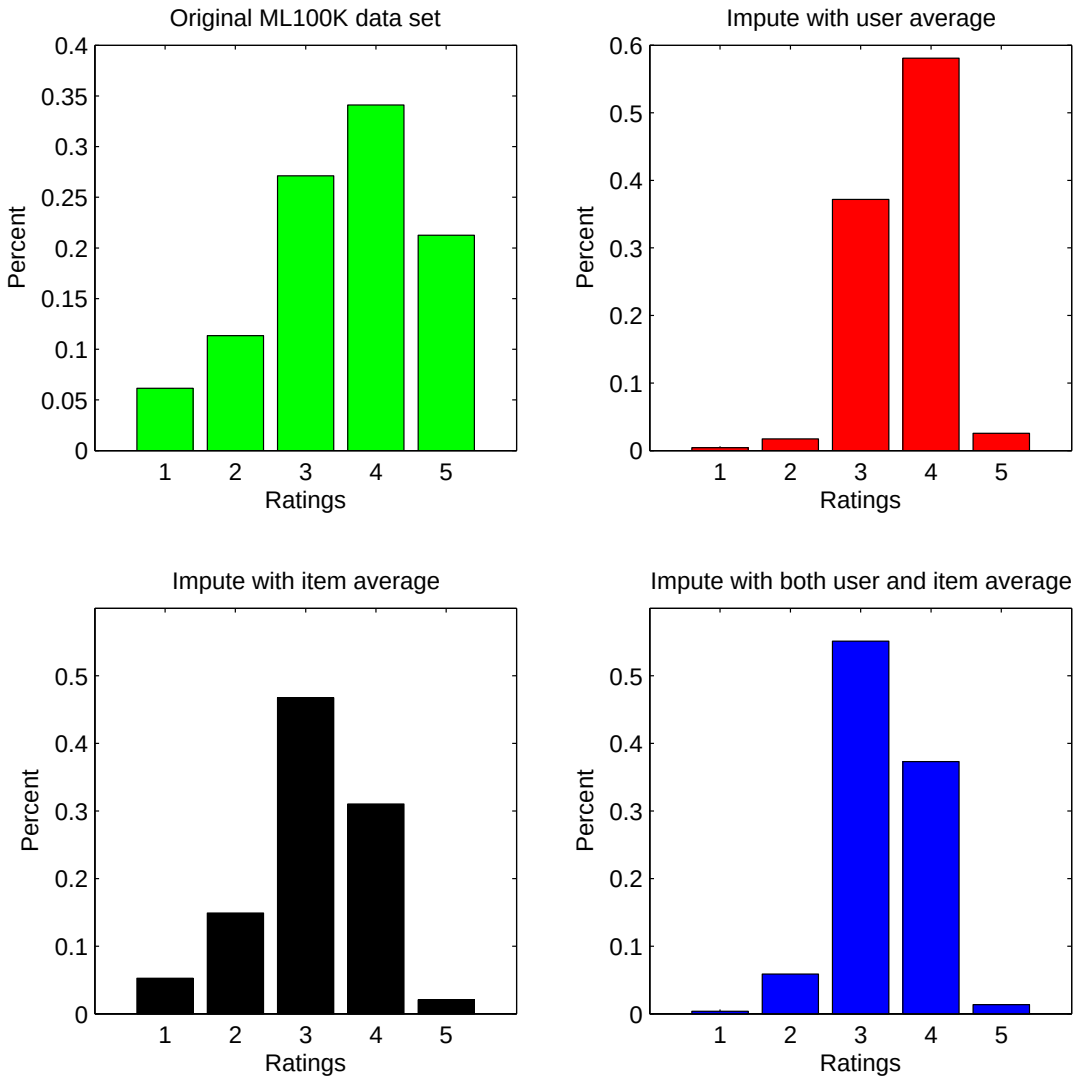


Figure 5.4: Rating distribution of imputed data

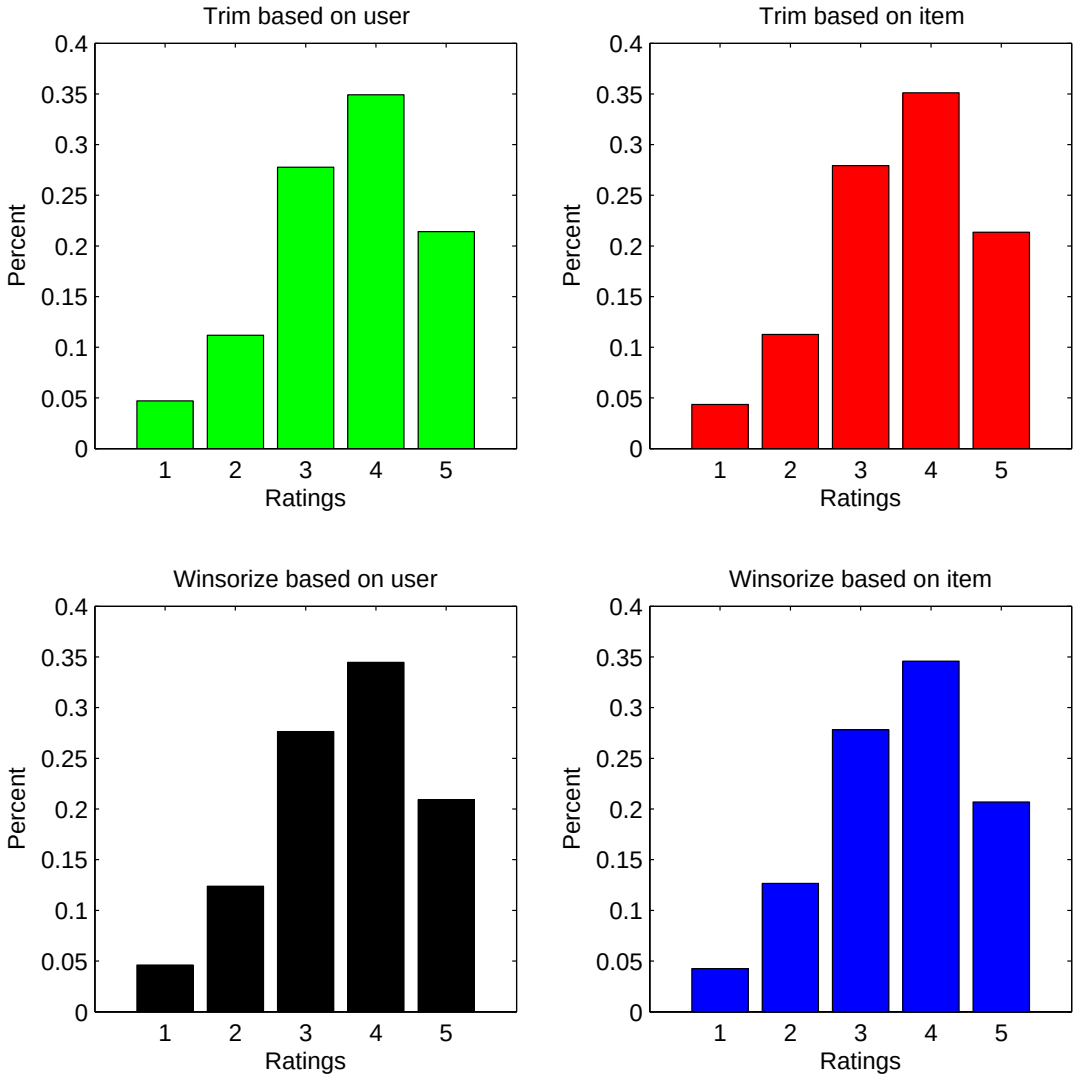


Figure 5.5: Rating distribution of trimmed and winsorized data

5.2 Rating Distributions and Performance in Synthetic Data

Inspired by this observation, a method is proposed to estimate the performance of LFMs given a test rating dataset. This method compares a given discrete rating dataset to a synthetic dataset generated from a Gaussian distribution which is discrete and truncated. Intuitively, the smaller the difference, the better the performance the could be expected from LFMs. The sampling algorithm to create the synthetic dataset

is described below (Algorithm 1).

Algorithm 1: SYNTHETIC Gaussian distributed data generation based on a given dataset

Input: A rating dataset for a certain category

Output: Synthetic ratings based on the features of the input dataset

Compute the statistical features of the input dataset, $length$(the total number of ratings), $mean$ (the overall mean of all the ratings), sd (the standard deviation of all the ratings);

Initialize parameters for a Gaussian distribution according to the statistical features, $\mu = mean, \sigma = sd$;

Initialize an empty list to store the generated ratings, $R_{synthetic}$;

for i *in* $1 : length$ **do**

    Sample a random number rn from a Gaussian distribution $rn \sim \mathcal{N}(\mu, \sigma)$;

    Round rn to an integer number;

    while $rn < 1 \parallel rn > 5$ **do**

        Sample a random number rn from a Gaussian distribution $rn \sim \mathcal{N}(\mu, \sigma)$;

        Round rn to an integer number;

    Add rn into $R_{synthetic}$;

return $R_{synthetic}$;

Once the synthetic dataset is prepared, Kullback-Leibler (KL) divergence [14, 31] is used to evaluate the difference between the two discrete probability distributions. Figure 5.6 shows the KL divergence from each user/item category to its corresponding discrete and truncated Gaussian distribution.

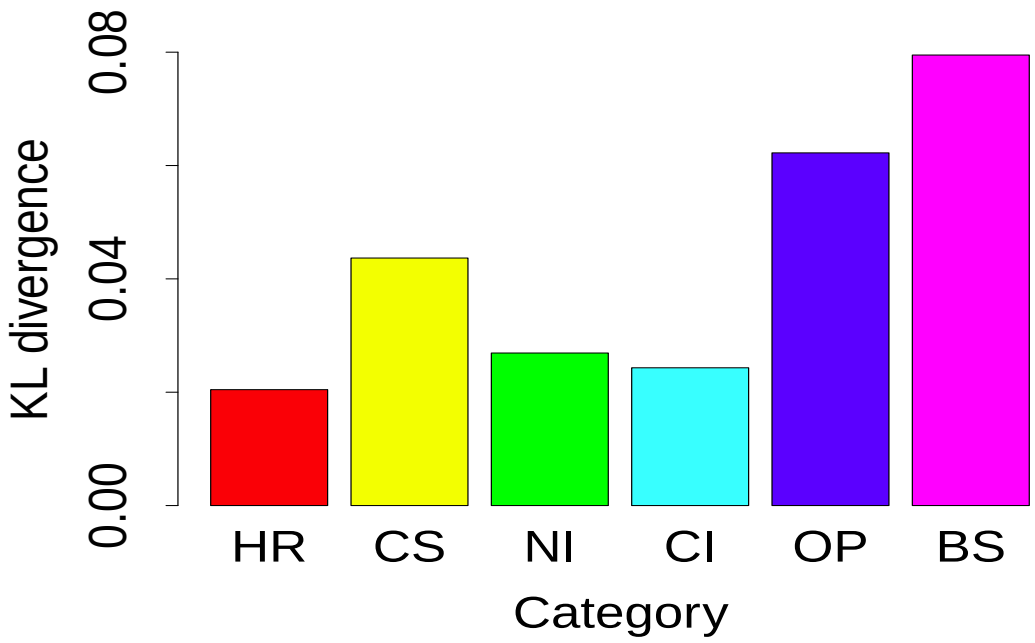


Figure 5.6: Barplot of KL divergence for different categories

It is obvious that the groundtruth ratings of Blacksheep users and Opinionated users are less likely to be Gaussian-like distributed than those of the other four categories. In addition, the rating of Heavyrater users has the smallest KL divergence among all categories. This fact verifies what has been observed in Figure 4.7, where LFMs have the best performance for Heavyrater users and the worst RMSE for Blacksheep users and Opinionated users. However, for Coldstart users, Controversial items and Niche items, due to insufficient samples, the estimation of KL divergence is not accurate enough to confirm the hypothesis. Possible measures other than KL divergence will be explored in future work.

Chapter 6

Conclusions

6.1 Conclusion of this Thesis

LFMs are well-known techniques to produce recommendations that are more efficient and accurate than the baseline method. We started the study by providing an experiment based introduction to LFMs, which includes the choice of data set, data preprocessing and the updating algorithm. Results on Movielens data suggested that standard imputation, trimming and winsorization approaches are not necessary here and SGD is more appropriate in this study. This result might not apply to other datasets. However, people can follow similar procedure to prepare their input data and build their own recommendation models.

Then detailed experiments are conducted to verify the capability of LFMs. Experimental results showed that whereas LFMs improve RMSE against the baseline method by a small percent over the whole dataset, LFMs show very promising advantages when dealing with certain difficult categories of users and items.

Furthermore, we identified the reason for the dramatically different behaviour of LFMs in the user/item groups. We highlighted that LFMs always output Gaussian-like rating prediction regardless of the type of category. We also presented the relationship between the groundtruth rating distribution and the predicted rating distribution for each category. We conclude that LFMs perform better on Gaussian-like distributed rating dataset.

6.2 Future Work

Future work involves expanding our study in several different data sets, such as Epinions, Yelp, and so on. Currently, our study only focuses on benchmark datasets in movie domain. Since our method is capable of investigating datasets with real rating values, it is possible to study the behaviour of LFMs in data set from other domains, either small or large scale datasets. Meanwhile, we will further investigate rating distribution as well as other possibilities that will help us to have a better understanding of the performance of LFMs.

Systematic investigation of the performance of LFMs has potential to improve both our understanding of the characteristics of LFMs and our ability to improve them. By understanding when LFMs fail, we can design and deploy a novel approach to tackle the difficult situations. On one hand, we can develop new data preprocessing techniques to better fit the corresponding models. On the other hand, our findings can be used to design performance prediction methods based on the statistical features of data.

Bibliography

[1] 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.

[2] Chris Anderson. *The long tail: Why the future of business is selling less of more.* Hyperion, 2008.

[3] Thorsten Angermann. *Empirische Analyse von Open Source Empfehlungssystemen.* PhD thesis, 2010.

[4] Shumeet Baluja, Rohan Seth, D Sivakumar, Yushi Jing, Jay Yagnik, Shankar Kumar, Deepak Ravichandran, and Mohamed Aly. Video suggestion and discovery for youtube: taking random walks through the view graph. In *Proceedings of the 17th international conference on World Wide Web*, pages 895–904. ACM, 2008.

[5] Chumki Basu, Haym Hirsh, William Cohen, and Others. Recommendation as classification: Using social and content-based information in recommendation. In *AAAI/IAAI*, pages 714–720, 1998.

[6] James Bennett and Stan Lanning. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35, 2007.

[7] Shlomo Berkovsky, Yaniv Eytani, Tsvi Kuflik, and Francesco Ricci. Enhancing privacy and preserving accuracy of a distributed collaborative filtering. In *Proceedings of the 2007 ACM conference on Recommender systems*, pages 9–16. ACM, 2007.

[8] Daniel Billsus and Michael J Pazzani. Learning Collaborative Information Filters. In *ICML*, volume 98, pages 46–54, 1998.

[9] John S Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 43–52. Morgan Kaufmann Publishers Inc., 1998.

[10] Robin Burke. Hybrid web recommender systems. In *The adaptive web*, pages 377–408. Springer, 2007.

[11] Tianqi Chen, Weinan Zhang, Qiuxia Lu, Kailong Chen, Zhao Zheng, and Yong Yu. {SVDFeature}: A Toolkit for Feature-based Collaborative Filtering. *Journal of Machine Learning Research*, 13:3619–3622, 2012.

[12] Richard Chow, Hongxia Jin, Bart Knijnenburg, and Gokay Saldamli. Differential data analysis for recommender systems. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 323–326. ACM, 2013.

[13] Maria Chowdhury, Alex Thomo, and William W Wadge. Trust-Based Infinitesimals for Enhanced Collaborative Filtering. In *COMAD*, 2009.

[14] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[15] Wilfrid J Dixon et al. Simplified estimation from censored normal samples. *The Annals of Mathematical Statistics*, 31(2):385–391, 1960.

[16] Michael Ekstrand and John Riedl. When recommenders fail: predicting recommender failure for algorithm selection and combination. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 233–236. ACM, 2012.

[17] Michael D Ekstrand, Michael Ludwig, Joseph A Konstan, and John T Riedl. Rethinking the recommender research ecosystem: reproducibility, openness, and LensKit. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 133–140. ACM, 2011.

[18] Simon Funk. Netflix update: Try this at home, 2006. *URL http://sifter. org/˜ simon/journal/20061211. html*, 2011.

[19] Zeno Gantner, Lucas Drumond, Christoph Freudenthaler, Steffen Rendle, and Lars Schmidt-Thieme. Learning attribute-to-feature mappings for cold-start rec-

ommendations. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 176–185. IEEE, 2010.

[20] Zeno Gantner, Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. MyMediaLite: A free recommender system library. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 305–308. ACM, 2011.

[21] Jiawei Han, Micheline Kamber, and Jian Pei. *Data mining: concepts and techniques*. Morgan kaufmann, 3rd edition, 2012.

[22] V Hernandez, J E Roman, A Tomas, and V Vidal. Restarted Lanczos bidiagonalization for the SVD in SLEPc, 2007.

[23] Geoffrey Hinton. A practical guide to training restricted Boltzmann machines. *Momentum*, 9(1), 2010.

[24] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 263–272. IEEE, 2008.

[25] Zan Huang, Daniel Zeng, and Hsinchun Chen. A comparison of collaborative-filtering recommendation algorithms for e-commerce. *IEEE Intelligent Systems*, 22(5):68–78, 2007.

[26] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, 2002.

[27] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–434. ACM, 2008.

[28] Yehuda Koren. Collaborative filtering with temporal dynamics. *Communications of the ACM*, 53(4):89–97, 2010.

[29] Yehuda Koren and Robert Bell. Advances in collaborative filtering. In *Recommender Systems Handbook*, pages 145–186. Springer, 2011.

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

[31] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[32] Aapo Kyrola, Guy Blelloch, and Carlos Guestrin. GraphChi: Large-scale graph computation on just a PC. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 31–46, 2012.

[33] Balaji Lakshminarayanan, Guillaume Bouchard, and Cedric Archambeau. Robust bayesian matrix factorisation. In *International Conference on Artificial Intelligence and Statistics*, pages 425–433, 2011.

[34] Joonseok Lee, Seungyeon Kim, Guy Lebanon, and Yoram Singer. Local Low-Rank Matrix Approximation. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 82–90, 2013.

[35] Joonseok Lee, Mingxuan Sun, and Guy Lebanon. A comparative study of collaborative filtering algorithms. *arXiv preprint arXiv:1205.3193*, 2012.

[36] Joonseok Lee, Mingxuan Sun, and Guy Lebanon. Prea: Personalized recommendation algorithms toolkit. *Journal of Machine Learning Research*, 13:2699–2703, 2012.

[37] Seok Kee Lee, Yoon Ho Cho, and Soung Hie Kim. Collaborative filtering with ordinal scale-based implicit ratings for mobile music recommendations. *Information Sciences*, 180(11):2142–2155, 2010.

[38] Zbigniew Leonowicz, Juha Karvanen, and Sergei L. Shishkin. Trimmed estimators for robust averaging of event-related potentials. *Journal of neuroscience methods*, 142(1):17–26, 2005.

[39] Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80, 2003.

[40] Roderick JA Little and Donald B Rubin. *Statistical analysis with missing data*. John Wiley & Sons, 2nd edition, 2002.

[41] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M Hellerstein. Distributed GraphLab: A framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment*, 5(8):716–727, 2012.

[42] Tariq Mahmood and Francesco Ricci. Improving recommender systems with adaptive conversational strategies. In *Proceedings of the 20th ACM conference on Hypertext and hypermedia*, pages 73–82. ACM, 2009.

[43] John I Marden. *Analyzing and modeling rank data*, volume 64. CRC Press, 1995.

[44] Paolo Massa and Paolo Avesani. Trust-aware collaborative filtering for recommender systems. In *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*, pages 492–508. Springer, 2004.

[45] Paolo Massa and Paolo Avesani. Trust-aware recommender systems. In *Proceedings of the 2007 ACM conference on Recommender systems*, pages 17–24. ACM, 2007.

[46] Raymond J Mooney and Loriene Roy. Content-based book recommending using learning for text categorization. In *Proceedings of the fifth ACM conference on Digital libraries*, pages 195–204. ACM, 2000.

[47] Arkadiusz Paterek. Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of KDD cup and workshop*, volume 2007, pages 5–8, 2007.

[48] Michael J Pazzani and Daniel Billsus. Content-based recommendation systems. In *The adaptive web*, pages 325–341. Springer, 2007.

[49] István Pilászy, Dávid Zibriczky, and Domonkos Tikk. Fast als-based matrix factorization for explicit and implicit feedback datasets. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 71–78. ACM, 2010.

[50] Martin Piotte and Martin Chabbert. The pragmatic theory solution to the netflix grand prize. *Netflix prize documentation*, 2009.

[51] Alexandrin Popescul, David M Pennock, and Steve Lawrence. Probabilistic models for unified collaborative and content-based recommendation in sparse-data

environments. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 437–444. Morgan Kaufmann Publishers Inc., 2001.

[52] Steffen Rendle. Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 995–1000. IEEE, 2010.

[53] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. GroupLens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186. ACM, 1994.

[54] Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to recommender systems handbook. In *Recommender Systems Handbook*, pages 1–35. Springer, 2011.

[55] Ruslan Salakhutdinov and Andriy Mnih. Bayesian probabilistic matrix factorization using Markov chain Monte Carlo. In *Proceedings of the 25th international conference on Machine learning*, pages 880–887. ACM, 2008.

[56] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Analysis of recommendation algorithms for e-commerce. In *Proceedings of the 2nd ACM conference on Electronic commerce*, pages 158–167. ACM, 2000.

[57] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Application of dimensionality reduction in recommender system-a case study. Technical report, DTIC Document, 2000.

[58] J Ben Schafer, Joseph Konstan, and John Riedi. Recommender systems in e-commerce. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 158–166. ACM, 1999.

[59] D Seung and L Lee. Algorithms for non-negative matrix factorization. *Advances in neural information processing systems*, 13:556–562, 2001.

[60] Yue Shi, Alexandros Karatzoglou, Linas Baltrunas, Martha Larson, Nuria Oliver, and Alan Hanjalic. CLiMF: learning to maximize reciprocal rank with collaborative less-is-more filtering. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 139–146. ACM, 2012.

[61] Joseph Sill, Gábor Takács, Lester Mackey, and David Lin. Feature-weighted linear stacking. *arXiv preprint arXiv:0911.0460*, 2009.

[62] Rashmi R Sinha and Kirsten Swearingen. Comparing Recommendations Made by Online Systems and Friends. In *DELOS workshop: personalisation and recommender systems in digital libraries*, volume 106, 2001.

[63] Rui Ping Song, Bo Wang, Guo Ming Huang, Qi Dong Liu, Rong Jing Hu, and Rui Sheng Zhang. A Hybrid Recommender Algorithm Based on an Improved Similarity Method. *Applied Mechanics and Materials*, 475:978–982, 2014.

[64] Andreas Töscher, Michael Jahrer, and Robert M Bell. The bigchaos solution to the netflix grand prize. *Netflix prize documentation*, 2009.

[65] Bin Xu, Jiajun Bu, Chun Chen, and Deng Cai. An exploration of improving collaborative recommender systems via user-item subgroups. *Proceedings of the 21st international conference on World Wide Web - WWW '12*, page 21, 2012.

[66] Xiao Yu, Xiang Ren, Yizhou Sun, Bradley Sturt, Urvashi Khandelwal, Quanquan Gu, Brandon Norick, and Jiawei Han. Recommendation in heterogeneous information networks with implicit user feedback. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 347–350. ACM, 2013.

[67] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. Large-scale parallel collaborative filtering for the netflix prize. In *Algorithmic Aspects in Information and Management*, pages 337–348. Springer, 2008.