Preference modeling and Accuracy in Recommender Systems

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Mohit Sharma

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor of Philosophy

Dr. George Karypis, Advisor

September, 2017

Acknowledgements

This thesis is a culmination of years of effort, and none of it would have been possible without the unconditional support of my parents. I am thankful to my brother for his encouragement and for taking over my responsibilities back home while I was away half way across the world.

A heartful thanks to my advisor Professor George Karypis for his invaluable guidance and support during my graduate studies. I am very grateful to him for his patience and for devoting all the time to help me grow as a researcher. I consider myself very fortunate to have him as a mentor and as a great source of inspiration.

I would like to thank Professors Arindam Banerjee, Joseph Konstan, Rui Kuang, and Zizhuo Wang for taking the time to serve on my preliminary exam and thesis committees.

Over the years, it has been a pleasure to work with the amazing members of the Karypis lab: Agi, Ancy, Asmaa, David, Dominique, Eva, Fan, Glu, Haoji, Jake, Jedi, Jeremy, Maria, Rezwan, Santosh, Sara, Saurav, Shaden, and Shalini. They have helped me get through the ups and downs in grad school by being a constant source of encouragement, support, and laughter.

My special thanks to the wonderful staff at the Department of Computer Science, the Digital Technology Center, and the Minnesota Supercomputing Institute at the University of Minnesota for providing assistance, facilities, and other resources for my research.

I am grateful to my mentors and colleagues during my internships at Technicolor Labs and Samsung Research. I would also like to thank my collaborators Max, Wei-Shou, Nandita, Arpan, Huong, Jiayu, and Junling for their valuable guidance.

Last, but not the least, I would like to thank all my wonderful friends.

Dedication

To mummy, papa and harshit

Abstract

Recommender systems are widely used to recommend the most appealing items to users. In this thesis, we focus on analyzing the accuracy of the state-of-the-art matrix completion-based recommendation methods and develop methods to model users' preferences to address different problems that arise in recommender systems.

Collaborative filtering-based methods are widely used to generate item recommendations to the user. The low-rank matrix completion method is the state-of-the-art collaborative filtering method. We will show that the accuracy and the ranking performance of matrix completion-based methods are affected by the skewed distribution of ratings in the user-item rating matrix. Additionally, we will illustrate that the number of ratings an item has positively correlates with the prediction accuracy and the ranking performance of the matrix completion approach for the item. Furthermore, we show that the users or the items that are present in the tail, i.e., those having few ratings in real datasets, may not have sufficient ratings to estimate the low-rank models accurately by matrix completion approach. We use these insights to develop TruncatedMF, a matrix completion-based approach that outperforms the state-of-the-art matrix completion method for the users and the items in the tail.

Since for new items we do not have any prior preferences from existing users, it is hard to recommend these items to the users. We can use non-collaborative methods that rely on similarities between the new item and the items preferred by a user in the past to model the user preference for the new item. However, these methods consider the item features independently and ignore the interactions among the features of the items while computing the similarities. Modeling the interactions among features can provide more information towards the relevance of an item in comparison to the scenario when the features are considered independently. We develop a new method called User-specific Feature-based factorized Bilinear Similarity Model (UFBSM), that uses all available information across users to capture these interactions among features and learns a *low-rank* user personalized *bilinear* similarity model for Top-n recommendation of new items.

In addition to providing ratings over individual items, the users can also provide ratings on sets of items. A rating provided by a user on a set of items conveys some preference information about the items in the set and enables us to acquire a user's preferences for more items that the number of ratings that the user provided. Moreover, users may have privacy concerns and hence may not be willing to indicate their preferences on individual items explicitly but may be willing to provide a rating to a set of items, as it provides some level of information hiding. We will investigate how do users' item-level preferences relate to their set-level preferences. Also, we will introduce collaborative filtering-based methods that explicitly model the user behavior of providing ratings on sets of items and can be used to recommend items to users.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis focuses on investigating and developing methods to address different problems in the area of recommender systems. Recommender systems are used to help consumers by providing recommendations that are expected to satisfy their tastes. They can identify from a large pool of items those few items that are the most relevant to a user and have become an essential personalization and information filtering technology. They rely on the historical preferences that were either explicitly or implicitly provided for the items and typically employ various machine learning methods to build predictive models from these preferences. For example, e-commerce services (e.g., Amazon, eBay) use them to help consumers by recommending products based on their past transactions, video streaming services (e.g., Netflix, Hulu) utilizes them to help their viewers by providing recommendations based on their previously watched movies or tv shows, and mobile app stores (e.g., Apple, Google Play) use them to recommend apps to their users.

Recommender systems generally use collaborative filtering-based methods to generate recommendations and low-rank matrix completion is the state-of-the-art collaborative filtering method. However, its accuracy is affected due to the sparsity structure of the rating matrices in real-world datasets. Also, standard collaborative filtering methods can not be used for recommendation of new items and to generate recommendations from users' preferences on sets of items. In this thesis, we primarily concentrate on three problems in recommender systems. First, we investigate how the accuracy of matrix completion is affected by the skewed distribution of ratings usually found in rating matrices and use the derived insights to develop a method that performs better for users and items with few ratings. Second, collaborative filtering-based methods can not be applied to recommend new items as they do not have any prior preferences. We develop a method to recommend new items to users based on the item features that take into account the interaction among the item features. Finally, we investigate how a user's preferences on sets of items relate to his/her preferences over individual items and introduce collaborative filtering-based methods that can be used to recommend items to users.

1.1 Key Contributions

In this section, we will give a brief introduction to the contributions made in this thesis.

1.1.1 Accuracy of matrix completion in recommender systems

The collaborative filtering methods for generating recommendations rely on preferences provided by the users over the items in the past. The matrix completion-based approaches are the state-of-the-art collaborative filtering methods that assume the user-item rating matrix is low rank and estimates the missing ratings based on the observed ratings in the matrix. However, the accuracy of these methods is affected by the distribution and the number of observed entries in the matrix.

In this thesis (Chapter 4), we show that the skewed distribution of the user-item rating matrix affects the accuracy and the ranking performance of recommendations generated using matrix completion-based methods. Additionally, we will show that the items having few ratings have low accuracy under matrix completion approach.

1.1.2 Truncated matrix factorization (TruncatedMF)

Certain attributes can describe an item being recommended, and few attributes determine a significant fraction of a user's rating over the item while other attributes can explain remaining rating. However, some users have provided ratings to few items, and some items have received few ratings from the users thus these users and items may not have sufficient ratings to estimate accurately the attributes that determine most of the user's rating over the item.

In this thesis (Chapter 5), we introduce a new method called TruncatedMF which considers the number of ratings received by an item or provided by a user to predict the user's rating over the item.

1.1.3 User-specific feature-based factorized bilinear similarity model

Since state-of-the-art collaborative filtering methods rely on prior preferences by users over items to generate recommendations, it is difficult to recommend new items as they do not have any previous preferences associated with them. The new items in recommender systems are often referred to as cold-start items. We can use non-collaborative filtering methods that rely on similarities between the new items and the items preferred by a user in the past to generate cold-start item recommendations. A major drawback of these methods is that they ignore the interactions among the item attributes and consider them independently while computing similarities between the items. The cold-start item recommendations can benefit from capturing the interactions between item features as modeling these interactions may provide additional information towards the significance of the item.

We will present the method **U**ser-specific **F**eature-based factorized **B**ilinear **S**imilarity **M**odel in Chapter 6 of this thesis, which leverages all the available information across users to model interactions among features and learns a user personalized bilinear similarity low-rank model for Top-n recommendation of new items.

1.1.4 Learning from sets of items in recommender systems

The collaborative filtering approaches used to generate recommendations depend on the preferences provided by users over individual items. However, the users can also indicate their preferences over sets of items rather than individual items and these preferences over sets of items can serve as an additional source of the users' preferences. Such set-level ratings are readily available in most of the existing recommender systems, e.g., ratings on song playlists, music albums, and reading lists. A user's preferences can be acquired for many items by using his/her preferences on different sets of items. Additionally, sometimes the users are not willing to explicitly reveal their true preferences on individual items but may provide a single rating to a set of items as it provides some level of information hiding.

In this thesis (Chapter 7), we will investigate how do a user's set-level ratings relate to the individual item-level ratings and how can we use collaborative filtering-based methods to generate item recommendations by using set-level ratings. To this end, we have collected ratings from active users of Movielens[1], a popular online movie recommender systems and based on our analysis of these collected ratings we will present different models that can predict a user's rating on a set of items as well as on individual items.

1.2 Outline

This thesis is organized as follows:

- Chapter 2 provides notation which is used throughout this thesis.
- Chapter 3 provides details of the existing research related to the different problems and methodologies presented in this thesis.
- Chapter 4 investigates how does the skewed distribution of ratings in rating matrices affects the accuracy and the ranking performance of recommendations generated using matrix completion-based methods.
- Chapter 5 presents TruncatedMF, a new matrix completion-based method which considers the number of ratings that a user or an item has before predicting the rating of the user on the item.
- Chapter 6 presents **U**ser-specific **F**eature-based factorized **B**ilinear **S**imilarity **M**odel method to address Top-*n* cold-start item recommendations problem.
- Chapter 7 investigates how does a user's set-level rating relates to the item-level ratings and presents collaborative filtering-based methods that use set-level ratings to generate item recommendations.

[1] www.movielens.org

- Chapter 8 summarizes the research presented in this thesis and provide concluding remarks along with some future research directions.

1.3 Related Publications

The publications related to the work presented in this thesis are listed as follows:

- **Mohit Sharma**, Jiayu Zhou, Junling Hu and George Karypis. Feature-based factorized bilinear similarity method for cold-start top-n item recommendation. In *SIAM International Conference on Data Mining, 2015.* SDM, 2015.

- David C. Anastasiu, Evangelia Christakapolou, Shaden Smith, **Mohit Sharma** and George Karypis. Big Data and Recommender Systems. In *Big Data Novatica Special Issue, 2016.*

- **Mohit Sharma**, F.Maxwell Harper and George Karypis. Learning from sets of items in recommender systems. In *eKNOW, International Conference on Information, Process, and Knowledge Management, 2017*, IARIA, 2017.

- **Mohit Sharma** and George Karypis. TruncatedMF: Improving recommendations for the tail. In *ACM International Conference on Web Search and Data Mining, 2018*, WSDM, 2018 (under review).

- **Mohit Sharma**, F.Maxwell Harper and George Karypis. Learning from sets of items in recommender systems. In *ACM Transactions on Interactive Intelligent Systems, 2018*, TiiS, 2018 (Ready for submission).

Chapter 2

Notations

All vectors are represented by bold lower case letters and they are row vectors (e.g., $\boldsymbol{p}, \boldsymbol{q}$). The ith component of vector $\boldsymbol{p}$ is denoted by $\boldsymbol{p}[\boldsymbol{i}]$. All matrices are represented by upper case letters (e.g., R, P). The ith row of a matrix P is represented by $\boldsymbol{p_i}$. The $(i,\ j)$ entry of matrix W is denoted by $w_{i,j}$. We use calligraphic letters to denote sets (e.g., $\mathcal{S}$, $\mathcal{T}$), and the size of a set $\mathcal{S}$ is represented by $|\mathcal{S}|$.

For quick reference, all the important symbols used, along with their definition is summarized in Table 2.1.

Table 2.1: Symbols used and definitions.

Symbol	**Definition**
$\mathcal{S}$	Set of items.
$\|\mathcal{S}\|$	Number of items in set $\mathcal{S}$.
u, i	Individual user u and item i.
m, n	Number of users and items.
k	Number of latent factors.
R	User-Item Feedback/Rating Matrix, $R \in \mathbb{R}^{m \times n}$.
$\mathcal{R}_u^+$	Set of items for which user u has provided feedback
$\mathcal{R}_u^-$	Set of items for which user u has not provided feedback
r_{ui}	Rating by user u on item i.
$\hat{r}_{ui}$	Predicted rating for user u on item i.
$r_u^{\mathcal{S}}$	Rating by user u on set $\mathcal{S}$.
$\hat{r}_u^{\mathcal{S}}$	Predicted rating for user u on set $\mathcal{S}$.
P	User latent factor matrix, $P \in \mathbb{R}^{m \times k}$.
Q	Item latent factor matrix, $Q \in \mathbb{R}^{n \times k}$.
$\boldsymbol{p}_u$	Latent factor of user u.
$\boldsymbol{q}_i$	Latent factor of item i.

Chapter 3

Background and Related Work

Recommender systems [1, 2] employ different algorithms to generate recommendations. These algorithms fall into two different classes: content-based methods [3, 4] and collaborative filtering-based methods [5]. Content-based methods rely on the attributes of the users and the items to generate recommendations. Collaborative filtering-based methodsmake use of the user preferences available in the form explicit ratings or implicit feedback. These methods utilize the user or item co-rating information to estimate the user preferences over the items. Collaborative filtering-based approaches can be further divided into two categories, i.e., neighborhood-based methods [6–9] and model-based or latent factor-based methods [10–12]. The neighborhood-based methods learn the user or the item neighborhood based on the co-rating information to generate the recommendations. The model-based approaches learn a model, i.e., the user and the item latent factors, from the rating data and use it to generate the recommendations. Next, we will discuss some of the work that is relevant to this thesis.

Matrix Completion The state-of-the-art methods for recommendations are based on matrix completion [13], and most of them involve factorizing the user-item rating matrix [10, 11, 14]. The Matrix Factorization (MF) method assume that the user-item rating matrix is low-rank and can be computed as a product of two matrices known as the user and the item latent factors. The predicted rating for the user u on the item i

is given by

$$\hat{r}_{ui} = \boldsymbol{p_u} \boldsymbol{q_i}^T. \tag{3.1}$$

The user and the item latent factors are estimated by minimizing a regularized square loss between the actual and predicted ratings

$$\underset{P,Q}{\text{minimize}} \quad \frac{1}{2} \sum_{r_{ui} \in R} (r_{ui} - \hat{r}_{ui})^2 + \frac{\beta}{2} \left(||P||_F^2 + ||Q||_F^2 \right), \tag{3.2}$$

where the matrices $P \in \mathbb{R}^{m \times k}$ and $Q \in \mathbb{R}^{n \times k}$ contains latent factors of the users and the items respectively. The parameter β controls the Frobenius norm regularization of the latent factors to prevent overfitting. This optimization problem can be solved by Stochastic Gradient Descent (SGD) [15].

In a separate body of work [13, 16], it has been shown that in order to complete a $n \times n$ matrix of rank r accurately by matrix completion-based methods, $O(nr \log(n))$ entries should be sampled uniformly at random from the matrix.

There has been some work on locality-based matrix completion methods which assume that different parts of the user-item rating matrix can be approximated accurately by different low-rank models [17–19]. The complete user-item rating matrix can be approximated as a weighted sum of these individual low-rank models.

Cold-start Item Recommendations The prior work to address the cold-start item recommendation can be divided into *non-collaborative* user personalized models and *collaborative* models. The non-collaborative models generate recommendations using only the user's past interaction history and the collaborative models combine information from the preferences of different users.

Billsus and Pazzani [20] developed one of the first user-modeling approaches to identify relevant new items. In this approach they used the users' past preferences to build user-specific models to classify new items as either "relevant" or "irrelevant". The user models were built using item features e.g., lexical word features for articles.

Personalized user models [21] were also used to classify news feeds by modeling short-term user needs using text-based features of items that were recently viewed by user and long-term needs were modeled using news topics/categories. Banos [22] used topic taxonomies and synonyms to build high-accuracy content-based user models.

Recently collaborative filtering techniques using latent factor models have been used to address cold start item recommendation problems. These techniques incorporate item features in their factorization techniques. Regression-based latent factor models (RLFM) [23] is a general technique that can also work in item cold-start scenarios. RLFM learns a latent factor representation of the preference matrix in which item features are transformed into a low dimensional space using regression. This mapping can be used to obtain a low dimensional representation of the cold-start items. User's preference on a new item is estimated by a dot product of corresponding low dimensional representations. The RLFM model was further improved by applying more flexible regression models [24]. AFM [25] learns item attributes to latent feature mapping by learning a factorization of the preference matrix into user and item latent factors $R = PQ^T$. A mapping function is then learned to transform item attributes to a latent feature representation i.e., $R = PQ^T = PAF^T$ where F represents items' attributes and A transforms the items' attributes to their latent feature representation.

A recently introduced approach, which was shown to outperform both RLFM and AFM methods in cold-start Top-n item recommendations is the User-specific Feature-based Similarity Models (UFSM) [26]. In this approach, a linear similarity function is estimated for each user that depends entirely on features of the items previously liked by the user, which is then used to compute a score indicating how relevant a new item will be for that user. In order to leverage information across users (i.e., the transfer learning component that is a key component of collaborative filtering), each user specific similarity function is computed as a linear combination of a small number of *global* linear similarity functions that are shared across users. Moreover, due to the way that it computes the preference scores, it can achieve a high-degree of personalization while using only a very small number of global linear similarity functions.

Predictive bilinear regression models [27] belong to the feature-based machine learning approach to handle the cold-start scenario for both users and items. Bilinear models can be derived from Tucker family [28]. They have been applied to separate "style" and "content" in images [29], to match search queries and documents [30], to perform semi-infinite stream analysis [31], and etc. Bilinear regression models try to exploit the correlation between user and item features by capturing the effect of pairwise associations between them. Let $\boldsymbol{x}_i$ denotes features for user i and $\boldsymbol{x}_j$ denotes features for item j, and a parametric bilinear indicator of the interaction between them is given by $s_{ij} = \boldsymbol{x}_i W \boldsymbol{x}_j^T$ where W denotes the matrix that describes a linear projection from the user feature space onto the item feature space. The method was developed for recommending cold-start items in the real time scenario, where the item space is small but dynamic with temporal characteristics. In another work [32], authors proposed to use a pairwise loss function in the regression framework to learn the matrix W, which can be applied to scenario where the item space is static but large, and we need a ranked list of items.

Learning From Sets of Items There has been little published work on using set-level ratings to improve the accuracy of item-level recommendations. The one exception is a recent study in which relative preference information on different groups of items was collected during a new user signup process and these preferences were then used to assign a user to a set of pre-built recommendation profiles [33]. This approach significantly reduced the time required to learn the user's preferences in order to generate recommendations for the new user. The principal difference from this approach is that in this thesis we try to model the user behavior that determines his/her estimated rating on a set and then use that to develop fully personalized recommendation methods that are not limited to new users.

In addition, there has been some work that has focused on recommending lists of items or bundles of items. For example, recommendation of music playlists [34–36], travel packages [37–40], reading lists [41] and recommendation of lists under user specified budget constraints [42, 43]. However, this research is not directly related to the problems explored in this thesis because our focus is on learning the user's ratings on items in lists from the ratings that the user provided on these lists.

Another relevant work is the problem of energy disaggregation [44], which refers to the task of separating the energy signal of a building into the energy signals of individual appliances that reside in the building. Disaggregated energy consumptions are used to provide feedback to consumers, forecast demands, design energy incentives and detect appliances' malfunction [45, 46]. Similar to the idea of energy disaggregation, in this thesis, we try to separate a user's rating on a set of items into the users' ratings on items in the set and generate item recommendations for the user.

The researchers have also investigated how a user's preference is affected by the position bias, i.e., the position of the items in the user interface showing a list of items [47–50]. In addition to positioning of items, the phrase or caption used to elicit preferences from a user can also affect a user's preference on a set of items [47, 51, 52]. Similarly, a user's rating on the set can be affected by reference points or anchoring biases [53–57], e.g., a user can focus on few items in a set while providing his/her rating on the set. The rating provided by a user can also depend on contextual factors, e.g., a user's mood at the time of providing his/her preference [58].

Furthermore, a user's rating on a set can be affected by the synergy and competition among items in the set. The user may rate the set of items independent of what is his preference for an individual item and instead rate the set depending on how does he perceive the set as a whole. There has been some work that has shown that a bundle of related products may result in better purchase intention than a bundle containing products that are not related [59–63]. Similar to the bundle of products, the items in a set can complement each other and thereby receive a more favorable rating from the user. On the contrary, it could be possible that items in a set compete with each other and thus receive a more critical rating on the set.

In this thesis, we have investigated how does the user provides a rating on a set of items and used the derive insights to develop collaborative filtering-based methods to predict the rating for an individual item in the set.

Chapter 4

Accuracy of matrix completion in recommender systems

The low-rank matrix completion-based approach is the state-of-the-art collaborative filtering based method used for generating recommendations. In this chapter, we show that the skewed distribution of ratings in the user-item rating matrix of real-world datasets affects the accuracy and the ranking performance of the matrix completion approach. Also, we investigate how does the number of ratings that an item has impacts the ability of low-rank matrix completion approaches to correctly estimate the ratings for the item and we show that the prediction accuracy and ranking performance for the item positively correlates with the number of ratings an item has.

4.1 Introduction

Recommender systems commonly use methods based on Collaborative Filtering [8], which rely on the historical preferences of the users over items in order to generate recommendations. These methods predict the ratings for the items not rated by the user and then select the items with the highest predicted ratings as item recommendations. In Top-n recommendations, n unrated items with highest predicted ratings and for small values of n, e.g., 10 and 50, are served as recommendations.

In practice, the users do not provide their ratings to all the items, and hence we observe only partial entries in the rating matrix. For the task of recommendations, we need to complete the matrix by predicting the missing ratings and select the unrated items with high predicted ratings as recommendations for a user. The matrix completion-based methods, discussed in Section 3, estimate the missing ratings based on the observed ratings in the matrix. These methods require entries in the matrix should be sampled uniformly at random for accurate recovery of the underlying low-rank model. However, most real-world rating matrices exhibit a skewed distribution of ratings as some users have provided ratings to few items and certain items have received few ratings from the users. This skewed distribution may result in insufficient ratings for certain users and items, and can thus affect the accuracy of the matrix completion-based methods.

This chapter investigates how does the skewed distribution of the ratings in the user-item rating matrix affects the accuracy and the ranking performance of the matrix completion-based methods and shows that the items having few ratings tend to have lower prediction accuracy. The key contributions of the work presented in this chapter are the following:

1. shows that the skewed distribution of ratings in the user-item rating matrix affects the accuracy of the matrix completion methods.

2. illustrates that the matrix completion-based methods mis-predicts the users' top rated items because of the skewed distribution of ratings in the user-item rating matrix.

3. shows that the false positives in Top-n item recommendations generated by the matrix completion-based methods are not rated significantly low.

4. shows that the number of ratings an item has, i.e., item frequency, affect the accuracy of the matrix completion and the Top-n item recommendations.

4.2 Matrix completion and skewed distribution of ratings

As described in Section 3, the matrix completion-based methods can accurately recover the underlying low-rank model of a given low-rank matrix provided entries are observed uniformly at random from the matrix. However, the ratings in the user-item rating matrix in real-world datasets do not represent a random sample of entries because some items receive few ratings and some users have rated few items, thus leading to a skewed distribution of ratings in the matrix. In the following sections, we will try to answer the question how does the skewed distribution of ratings in real datasets affects the accuracy and the ranking performance of the matrix completion-based methods. Furthermore, we will try to understand how does the performance of matrix completion-based methods changes with the number of ratings an item have.

4.2.1 Experiment design

In order to study how the skewed distribution of ratings in real datasets affects the ability of matrix completion to accurately complete the matrix (i.e., predict the missing entries) we performed a series of experiments using synthetically generated low-rank rating matrices. In order to generate a rating matrix $R \in \mathbb{R}^{n \times m}$ of rank k we followed the following protocol. We started by generating two matrices $A \in \mathbb{R}^{n \times k}$ and $B \in \mathbb{R}^{m \times k}$ whose values are uniformly distributed at random in $[0, 1]$. We then computed the singular value decomposition of these matrices to obtain $A = U_A \varSigma_A V_A^T$ and $B = U_B \varSigma_B V_B^T$. We then let $P = \alpha U_A$ and $Q = \alpha U_B$ and $R = PQ^T$. Thus, the final rank k matrix R is obtained as the product of two randomly generated rank k matrices whose columns are orthogonal. Note that the parameter α was determined empirically in order to produce ratings in the range of $[-10, 10]$.

We used the above approach to generate full rating matrices whose dimensions are those of the four real-world datasets shown in Table 4.1. For each of these matrices we used two approaches to select the subset of their entries that will be given as input to the matrix completion algorithm. The first approach selects the entries that correspond to the actual user-item pairs that are present in the corresponding dataset, whereas the second approach selects the entries uniformly at random from the entire matrix. The number of entries that are selected by both approaches is the same and is the number of non-zeros in the actual dataset (shown in Table 4.1).

The advantages of working with this type of synthetically generated datasets are two-fold. First, by construction we can ensure that the underlying matrix is of known (low) rank. Second, since we know the values of the full matrix, we can easily measure how accurately the low-rank models estimated using matrix completion can complete

Table 4.1: Datasets used in experiments

Dataset	**users**	**items**	**ratings**	μ_u[a]	σ_u[b]	μ_i[c]	σ_i[d]	density (%)[†]
EachMovie (EM)	61,265	1,623	2,811,983	45.89	59.48	1732.58	3882.55	2.8
Flixster (FX)	147,612	48,794	8,196,077	55.52	225.81	167.97	934.47	0.1
Movielens 20M (ML)	229,060	26,779	21,063,128	91.95	190.53	786.55	3269.45	0.3
Netflix (NF)	480,189	17,772	100,480,507	209.252	302.33	4550.75	16908.40	1.1

[a] The number of average ratings per user in the dataset.
[b] The standard deviation of ratings per user in the dataset.
[c] The number of average ratings per item in the dataset.
[d] The standard deviation of ratings per item in the dataset.
[†] The percentage of observed ratings in the dataset.

the entire matrix.

In order to estimate the low-rank factor matrices from the observed entries (Equation 3.2) we used Stochastic Gradient Descent [15] and initialized the factor matrices with the singular vectors of the rating matrix by assuming that the missing entries were rated as 0, which is shown to converge closer to global minimum [64]. For each dataset we generated five different sets of matrices using different random seeds and we performed a series of experiments using synthetically generated low-rank matrices of rank 5, 10, and 20. For each rank, we report the average of performance metrics in each set from the estimated low-rank models over all the synthetic matrices.

To simplify the discussion, we will refer to the set of matrices derived from the actual sparsity structure of the real datasets as *SYN-REAL* and from the randomly sampled entries as *SYN-RAND*. In addition we will refer to the values of the synthetically generated rating matrices as *ground-truth* in order to differentiate them from the values predicted as part of matrix completion.

4.2.2 Accuracy of the estimated low-rank models

Table 4.2 shows the Root Mean Square Error (RMSE) achieved by the models estimated using both the SYN-REAL and SYN-RAND matrices over the complete rating matrix.

As can be seen in the table, the RMSE of the low-rank models estimated using the randomly sampled entries is lower than those estimated using the actual entries. Additionally, the RMSE increases with the increase in the rank for both sets of matrices. This is because, as mentioned in Section 4.1, the required number of observed entries

Table 4.2: RMSE of the estimated low-rank models for different sparsity structures.

Dataset	Rank	RMSE for SYN-REAL matrices	RMSE for SYN-RAND matrices	Dataset	Rank	RMSE for SYN-REAL matrices	RMSE for SYN-RAND matrices
EM	5	0.675	<u>0.028</u>	ML	5	1.377	<u>0.024</u>
	10	1.110	<u>0.052</u>		10	1.222	<u>0.039</u>
	20	1.229	<u>0.165</u>		20	1.872	<u>0.074</u>
FX	5	1.962	<u>0.028</u>	NF	5	0.246	<u>0.023</u>
	10	2.225	<u>0.053</u>		10	0.425	<u>0.034</u>
	20	2.425	<u>0.167</u>		20	0.621	<u>0.052</u>

to complete the matrix accurately increases linearly with the rank of the matrix. The RMSE for the SYN-REAL matrices in NF is lower than that of the others because it has more ratings for both users and items, thus leading to more accurate estimation of low-rank models. The higher RMSE in the SYN-REAL matrices compared to that of the SYN-RAND matrices suggest that the estimated low-rank model fails to recover the missing entries accurately in the SYN-REAL matrices. This failure can result in poor predictions for a user on some items and hence impact the recommendations served to the user.

Effect of item frequency

Since the matrix completion-based methods fail to recover the missing entries accurately in the SYN-REAL matrices, we investigated if the number of ratings an item has, i.e., item frequency, has any influence on the accuracy of the matrix completion-based methods for the item. We ordered all the items in decreasing order by their frequency in the rating matrix. We divided these ordered items into ten buckets and for a user computed the RMSE for items in each bucket based on the error between the predicted rating by the estimated low-rank model and the ground-truth rating. We repeated this for all the users and computed the average of the RMSE of the items in each bucket over all the users. Figure 4.1 shows the RMSEs across the buckets along with the average frequency of the items in the buckets. As can be seen in the figure, the predicted ratings for the frequent items tend to have lower RMSE in contrast to infrequent items for most

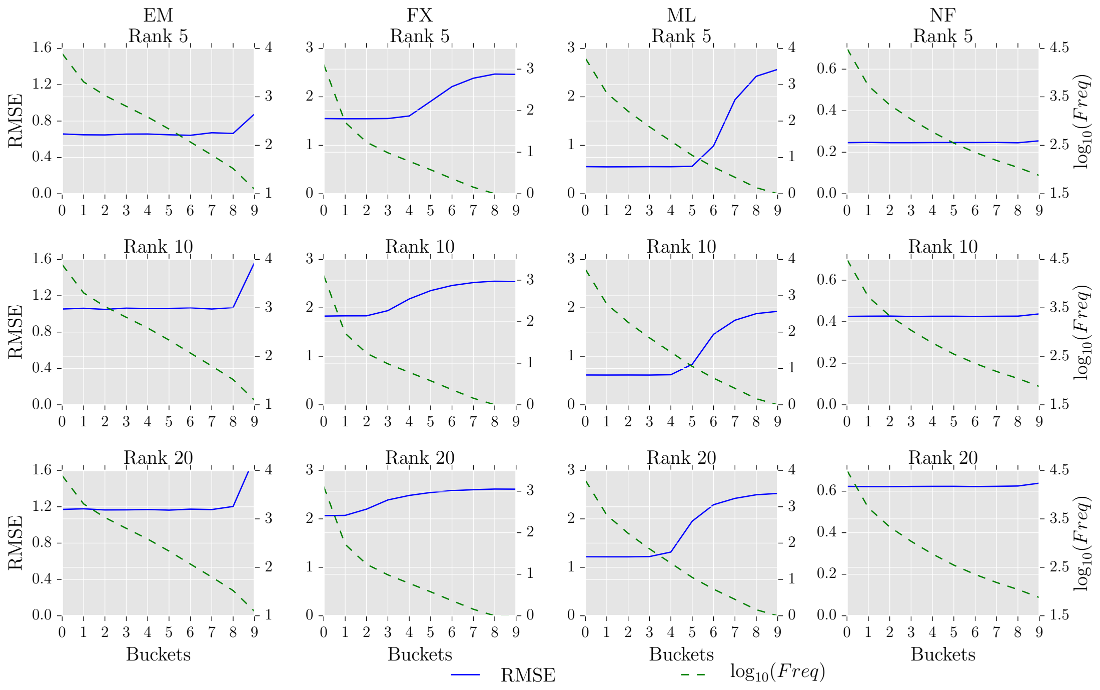


Figure 4.1: RMSE of the predicted ratings as the frequency of the items decreases.

of the datasets. However, in NF dataset because of the higher number of average ratings for both the users and the items the RMSE tends to remain the same over all the items. Figure 4.2 shows the scatter map of items in FX and EM dataset having different frequency against the number of instances where the absolute difference between the original and the predicted rating, i.e., *Mean Absolute Error (MAE)*, is ≤ 0.5. As can be seen in the figure, the number of accurate predictions is significantly lower for items having fewer ratings (≤ 20) compared to that of the items having a greater number of ratings (≥ 30). The higher RMSE of the infrequent items is because they do not have sufficient ratings to estimate their latent factors accurately. Hence for the real datasets, items appearing at the top in ordering by frequency and having high predicted scores will form a reliable set of recommendations to a user.

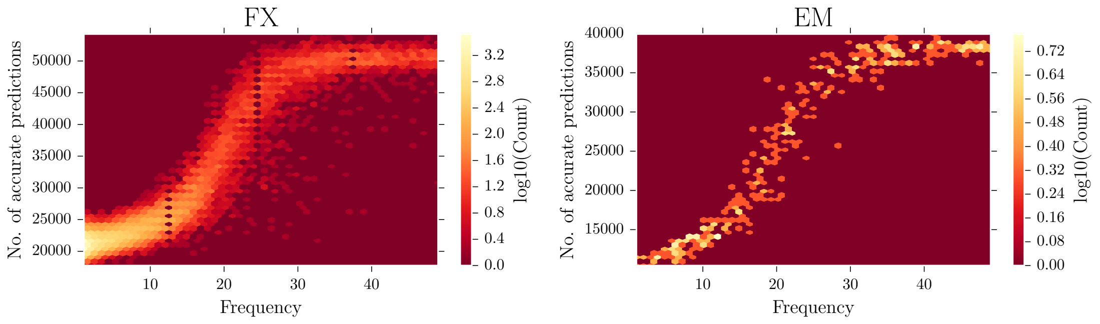


Figure 4.2: Scatter map of items having different frequency against their number of accurate predictions (Mean absolute error (MAE) ≤ 0.5) for low-rank models with rank 20 for FX and EM datasets.

Table 4.3: Overlap between the original top 5% of the items and the predicted top 5% of the items by the estimated low-rank models for real and random sparsity structure.

Dataset	Rank	Real (%)	Random (%)	Dataset	Rank	Real (%)	Random (%)
EM	5	90.77	<u>99.59</u>	ML	5	68.86	<u>99.75</u>
	10	79.94	<u>99.08</u>		10	56.50	<u>99.34</u>
	20	63.69	<u>95.62</u>		20	43.32	<u>98.65</u>
FX	5	39.82	<u>99.57</u>	NF	5	98.31	<u>99.84</u>
	10	27.13	<u>98.98</u>		10	96.19	<u>99.72</u>
	20	18.77	<u>95.94</u>		20	89.99	<u>99.29</u>

4.2.3 Ranking performance of the estimated low-rank models

We define ranking performance of the estimated low-rank models as their ability to predict high the true high rated items for a user. In order to evaluate how the errors in predictions by matrix completion-based methods impact the ranking performance of the estimated low-rank model, we analyzed the top $n\%$ of the items predicted by the estimated low-rank model for a user and investigated whether true high rated items are predicted low or true low rated items are predicted high. In the following analysis, we will refer to the top $n\%$ of the items predicted by the estimated low-rank models as $E_u^{n\%}$ and the top $n\%$ of the items ordered by the ground-truth ratings as $G_u^{n\%}$.

Table 4.3 shows the fraction of items that are common between $G_u^{5\%}$ and $E_u^{5\%}$. As can be seen in the table, the items in $E_u^{5\%}$ for the matrices with real sparsity structure

Table 4.4: Recall@$n\%$* of the top 5% of the ground-truth items in rankings by the estimated low-rank models for datasets with real sparsity structure.

Dataset	Rank	Recall@5%	Recall@10%	Recall@15%	Recall@25%	Recall@50%
EM	5	90.77	94.2	95.45	96.88	98.67
	10	79.95	86.4	89.33	92.87	97.26
	20	63.69	73.93	79.17	85.74	94.44
FX	5	39.82	48.11	53.94	62.93	80.87
	10	27.13	35.66	42.32	53.38	76.45
	20	18.77	26.79	33.52	45.25	70.65
ML	5	68.87	73.20	75.94	80.13	89.43
	10	56.50	62.17	66.09	72.50	87.26
	20	43.33	50.66	55.84	64.41	83.93
NF	5	98.32	99.10	99.30	99.52	99.80
	10	96.19	98.02	98.50	99.03	99.63
	20	89.99	94.76	96.19	97.67	99.19

* The percentage of items in $G_u^{5\%}$ that are present in $E_u^{n\%}$.

miss a significant number of the items in $G_u^{5\%}$. On the contrary, the low-rank models estimated on the matrices with random sparsity miss a comparatively smaller number of the items in $G_u^{5\%}$.

Further, we explored how the low-rank model mis-predicts the original top 5% items for a user, i.e., $G_u^{5\%}$, in real datasets. We computed the position of these items in the ranking of all items by their predicted ratings and based on these positions computed the Recall@$n\%$, i.e., the percentage of items in $G_u^{5\%}$ that are present in $E_u^{n\%}$. In Table 4.4, we present the Recall@$n\%$ of these items in the ranking of all the items by their predicted ratings. For example, as can be seen in the table for ML dataset with rank 5, the 68.87% of the items in $G_u^{5\%}$ are present in $E_u^{5\%}$, 73.20% of these appear in $E_u^{10\%}$, and 89.43% of these are in $E_u^{50\%}$. Similar trend occurs for the remaining datasets, i.e., the items in $G_u^{5\%}$ that are not present in $E_u^{5\%}$ are spread across the entire ranking and this spread increases with the rank of the matrices. Also, the Recall@n is higher for denser datasets, i.e., for EM and NF, when compared to the other datasets. The lower value of Recall@5 indicates that a considerable large number of the highest rated items are not ranked high by the estimated low-rank models.

Since in many cases, $E_u^{5\%}$ contains items that are not ranked high according to

Table 4.5: Recall@$n\%$* of the top 5% items predicted by low-rank models in non-increasing ordering of all items by the ground-truth ratings for datasets with real sparsity structure.

Dataset	Rank	Recall@5%	Recall@10%	Rec@15%	Recall@25%	Recall@50%
EM	5	90.77	94.81	95.96	97.28	98.87
	10	79.95	88.38	91.03	94.13	97.78
	20	63.69	77.34	82.29	88.21	95.40
FX	5	39.82	65.64	72.04	80.00	90.99
	10	27.13	48.25	58.88	69.77	86.20
	20	18.77	33.76	46.03	58.95	79.63
ML	5	68.87	95.63	96.90	98.06	99.23
	10	56.50	89.55	92.67	95.54	98.41
	20	43.33	74.97	81.83	88.88	96.21
NF	5	98.32	99.12	99.31	99.54	99.81
	10	96.19	98.04	98.52	99.04	99.64
	20	89.99	94.80	96.22	97.70	99.20

* The percentage of the items in $E_u^{5\%}$ that are present in the $G_u^{n\%}$.

their ground-truth ratings, we investigated where these items are located in the ground-truth rankings by computing the position of the items in $E_u^{5\%}$ in the ranking of all items by their ground truth ratings. We computed the Recall@$n\%$ of these items, i.e., the percentage of the items in $E_u^{5\%}$ that are present in the $G_u^{n\%}$. Table 4.5 shows the Recall@$n\%$ of the items in $E_u^{5\%}$ in the ranking of all items by the ground-truth ratings in decreasing order. For example, as can be seen in the table for ML dataset with rank 5, the 68.87% of items in $E_u^{5\%}$ are present in $G_u^{5\%}$, 95.63% of these appear in $G_u^{10\%}$, and almost all, i.e., 99.23% of these are in $G_u^{50\%}$. The remaining datasets in the table follows a similar trend, i.e., the items in $E_u^{5\%}$ are present close to the top in the ground-truth ranking. This indicates that the items that are predicted high by the estimated low-rank models are also in general true high rated items.

Effect of item frequency

We investigated how the ranking performance of the estimated low-rank models varies with the frequency of the items in SYN-REAL matrices.

To this end, for each user we computed the Recall@$n\%$ of the items in $G_u^{5\%}$, i.e., the percentage of the items in $G_u^{5\%}$ that are present in $E_u^{n\%}$. Here, n takes the value in

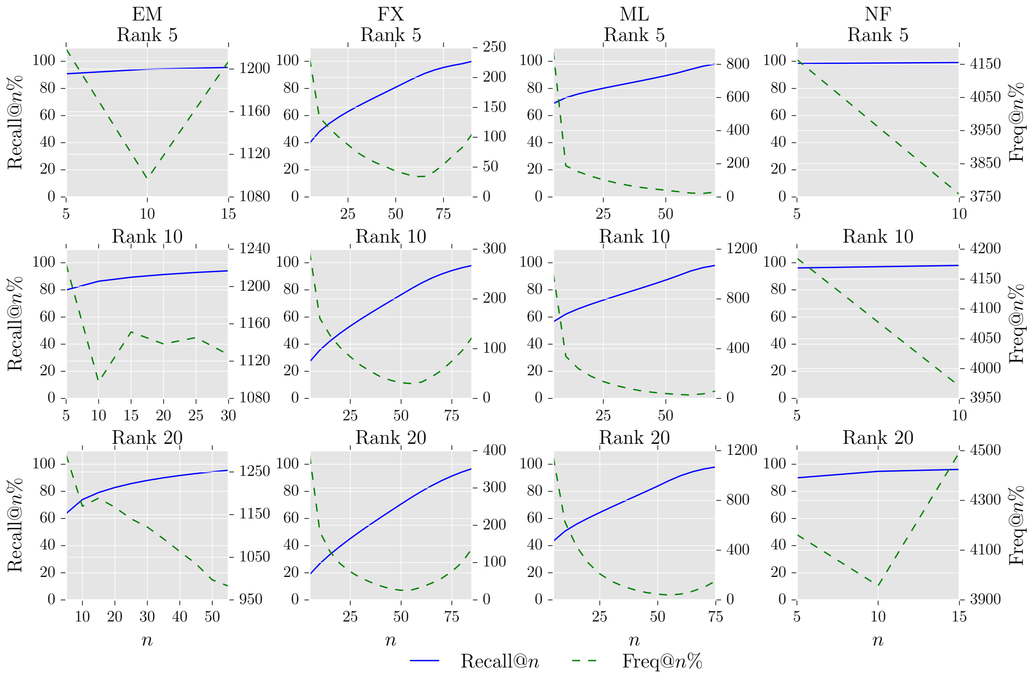


Figure 4.3: Recall@n% and Freq@n% of the top 5% of the ground-truth items in ordering by the predictions from the estimated low-rank models.

[5, 10, 15, ..., 100]. Also, we computed the average frequency of the items in $G_u^{5\%}$ that are present in $E_u^{n\%}$ but are absent in $E_u^{(n-5)\%}$. We will refer to the average frequency of these items as Freq@n%. As can be seen in Figure 4.3, for the datasets with fewer ratings per item, i.e., the ML and the FX datasets, the items having low frequency tend to appear at the end of the ranking. This trend is also seen to some extent in the denser datasets, i.e., the EM and the NF datasets. We hypothesize that it is because the low-rank model, i.e., the user and the item latent factors, is initialized with values close to zero and since items with fewer ratings do not often occur in the updates during the model estimation they are predicted low by the estimated low-rank model. Therefore, the ranking performance of the estimated low-rank models on SYN-REAL matrices is not affected by false positives as both frequent and infrequent items that are rated low will be predicted low while frequent items that are rated high will be predicted high. To test this hypothesis, we initialized the low-rank models with higher values in the range

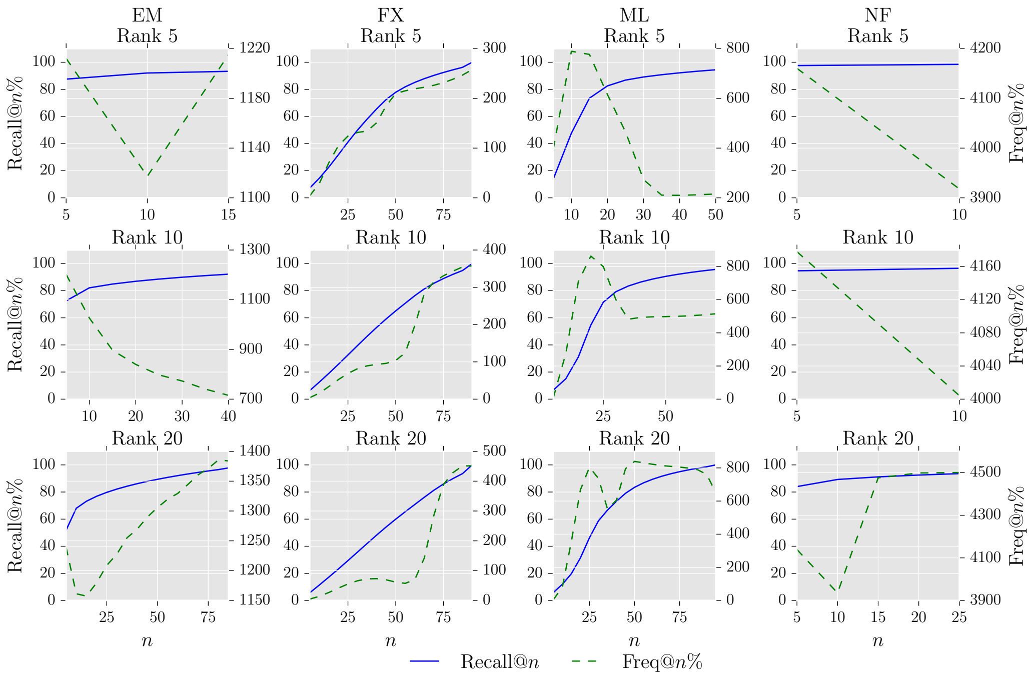


Figure 4.4: Recall@n% and Freq@n% of the top 5% of the ground-truth items in ordering by the predictions from the estimated low-rank models initialized with values in the range [0, 5].

[0,5] and analyzed the learned models. Figure 4.4 shows the Recall@n% of the items in $G_u^{5\%}$ along with their average frequency, i.e., Freq@n%, for the estimated low-rank models initialized with higher values. As can be seen in the figure, unlike the estimated low-rank models initialized with values close to zero, the items having low frequency does not necessarily appear in the later buckets. Additionally, the Recall@n is significantly lower for smaller values of n when compared with that of the estimated low-rank models initialized with values close to 0. This suggests that the model initialization affects both the accuracy and the ranking performance of the matrix completion-based methods.

Previously in Section 4.2.2, we showed that the accuracy of the estimated low-rank models is better for items having high frequency. Considering the analysis of ranking performance, we can reason that the frequent items that are rated high will be predicted high by the estimated low-rank model. Hence, the ranking performance of the estimated low-rank model is better for these items than the items that are rated high but are infrequent.

4.3 Conclusion

In this work, we have investigated the performance of the matrix completion-based low-rank models for estimating the missing ratings in user-item rating matrices having sparsity structure identical to real datasets. We showed in Section 4.2.2 that the matrix completion-based methods because of the presence of skewed distribution of entries in rating matrices in real datasets fail to predict the missing entries accurately in the matrices. Also, we learned that the items with high frequency are predicted more accurately than the others. These findings imply that for a user, the unrated items which have more ratings in the matrix and are predicted high for the user, will form a better set of recommendations than the items with fewer ratings in the rating matrix.

Further, we saw in Section 4.2.3 that the errors in predictions due to the skewed distribution of ratings in the user-item rating matrix affect the ranking performance of the matrix-completion based methods. In particular, under the assumption that the rating that a user will provide to an item determines his ranking preference, our results indicate that the items predicted at the top by matrix completion-based methods miss a large number of true high rated items. In some datasets, the true high rated items are missing even in the top 50% of the predicted items for the user. However, the items that are predicted at the top for a user but are absent from the true high rated items are present close to the true high rated items by the user. Therefore, the ranking based on the predicted ratings is not severely affected by false positives as it does not contain items that are significantly low rated. Additionally, we observed that the infrequent items, irrespective of whether they are true high rated or true low rated, are predicted low by the matrix completion-based methods thereby appearing later in the ranking of the items for recommendations.

Chapter 5

TruncatedMF: Truncated matrix factorization

This chapter focuses on improving the matrix completion-based recommendation methods for users and items present in the tail, i.e., those having few ratings in the user-item rating matrix. We show that the performance of matrix completion in real datasets vary with the number of ratings that a user or an item has, and its accuracy is low for the users or the items with few rating. Furthermore, we use these insights to develop *TruncatedMF*, a matrix completion-based approach, that outperforms the state-of-the-art MF method for the users and the items in the tail.

5.1 Introduction

The matrix completion-based methods, e.g., matrix factorization (MF) [10, 11, 14], are the state-of-the-art collaborative filtering methods that use users' historical preferences over items to generate recommendations. The ratings provided by users over the items can be viewed as a matrix whose rows represent the users, columns denote the items, and entries are the ratings provided by the users over the items. There exists a small number of attributes that describe the items, and a user's rating depends on how the user values those attributes. This makes the user-item rating matrix low-rank, and the number of attributes determines the rank of the matrix. The attributes of the items and the weights provided by a user over these attributes are often referred as the items'

Table 5.1: Datasets used in experiments

Dataset	**users**	**items**	**ratings**	μ_u[a]	σ_u[b]	μ_i[c]	σ_i[d]	density (%)[†]
EachMovie (EM)	61,265	1,623	2,811,983	45.89	59.48	1732.58	3882.55	2.8
Flixster (FX)	147,612	48,794	8,196,077	55.52	225.81	167.97	934.47	0.1
Movielens 10M (ML10)	69.878	10,677	10,000,054	143.10	216.71	936.59	2487.21	0.01
Movielens 20M (ML20)	229,060	26,779	21,063,128	91.95	190.53	786.55	3269.45	0.3
Netflix (NF)	480,189	17,772	100,480,507	209.252	302.33	4550.75	16908.40	1.1

[a] The number of average ratings per user in the dataset.
[b] The standard deviation of ratings per user in the dataset.
[c] The number of average ratings per item in the dataset.
[d] The standard deviation of ratings per item in the dataset.
[†] The percentage of observed ratings in the dataset.

latent factors and the users' latent factors, respectively. MF estimates the user-item rating matrix as the product of the user latent factors and the item latent factors.

In practice, there are few attributes that are responsible for a large portion of the rating provided by a user on an item and the remaining rating can be explained by other attributes. However, certain users have provided ratings to few items, and some items have received few ratings from the users thereby these users or items may not have sufficient ratings to estimate weights for all the attributes accurately. The inaccuracy in the estimation of weights for these users and items can affect the predicted ratings and hence affect the generated recommendations. Therefore, the recommendations for these users and items with few ratings may improve by focusing on few attributes that are responsible for a significant portion of the rating and can be estimated accurately by matrix completion-based methods.

This chapter investigates how does the performance of the matrix completion-based methods changes with the number of ratings that a user or an item has, and shows that the users or the items with few ratings tend to have low accuracy. Additionally, we show that the error in predictions for such users or items increases further with the increase in rank of the low-rank matrix completion-based methods. Furthermore, we use these findings to develop TruncatedMF which considers the number of ratings received by an item or provided by a user to estimate the rating of the user on the item. The exhaustive experiments on the real datasets demonstrate the effectiveness of TruncatedMF over the state-of-the-art MF method for the users and the items with few ratings.

5.2 Effect of frequency on accuracy in real datasets

We investigated how does the performance of matrix completion method vary for items with the different number of ratings in user-item rating matrix and in order to do the analysis we evaluated matrix completion on a random held-out subset of the real datasets shown in Table 5.1. We followed the standard procedure of dividing the available ratings in a dataset at random into training, validation and test splits, i.e., 60% of the ratings were used for learning the low-rank models and rest were used equally for validation and test splits. To learn the model we tried rank in the range [1, 5, 10, 15, 25, 30, 40, 50] and regularization parameters in the range [0.001, 0.01, 0.1, 1]. We performed this procedure three times and selected the model giving the lowest average RMSE on the validation splits. Table 5.2 shows the test RMSE achieved by the selected models for different datasets. In addition to computing RMSE over all the ratings in the test split, we also computed RMSE over the infrequent items in the test split, i.e., the items that have few ratings in the training split. In order to identify infrequent items, we ordered the items in increasing order by the number of ratings in training splits. Next, we divided these ordered items into quartiles and designated the items in the first and the last quartile as the infrequent and the frequent items respectively.

Figures 5.1 and 5.2 show the RMSE for the items and the users in the test, respectively. As can be seen in the figures, for all the datasets the RMSE of the frequent items (or users) is lower than that of the infrequent items (or users) . These results suggest that the matrix completion method fails to estimate the preferences for the infrequent items (or users) accurately in the real datasets. Also as can be seen in Figure 5.1, for FX, ML10 and ML20 datasets, the RMSE of the infrequent items increases with the increase in the rank while that of frequent items decreases with the increase in the rank. Since NF and EM has a high number of average ratings for the items, the RMSE also tends to decrease for the infrequent items with the increase in the rank. Similarly, as can be seen in Figure 5.2, in FX and ML20 datasets the RMSE of the infrequent users increases with the increase in the rank. The increase in RMSE with the increase in ranks suggests that infrequent items or infrequent users may not have sufficient ratings to estimate all the ranks accurately thereby leading to the error in predictions for such users or items.

Table 5.2: Test RMSE for real datasets.

Rank	EM	FX	ML10	ML20	NF
1	1.282	0.903	0.870	0.875	0.926
5	1.158	0.871	0.815	0.824	0.863
10	1.142	<u>0.866</u>	0.804	0.813	0.845
15	1.138	0.864	0.800	0.810	0.839
25	1.134	0.864	0.798	<u>0.809</u>	0.834
30	1.132	0.864	0.797	0.809	0.832
40	1.132	0.864	<u>0.796</u>	0.809	0.831
50	<u>1.131</u>	0.864	0.796	0.809	<u>0.830</u>

5.3 Truncated matrix factorization

In Section 5.2, we showed that the infrequent items and the infrequent users tend to have a high error under matrix completion-based approach. Furthermore, this error increases with the increase in rank of the estimated low-rank models, i.e, increases with the dimension of estimated users' and items' latent factors. We propose to use these observations to devise a approach, which we will refer to as Truncated Matrix Factorization (TMF), to improve the accuracy of the low-rank models for both the users and the items having few ratings. Since for the items and the users with few ratings we may not be able to estimate all the ranks of a low-rank model accurately, we propose to consider only a subset of the ranks for these users or items.

In our approach, the estimated rating for user u on item i is given by

$$\hat{r}_{u,i} = \boldsymbol{p_u}(\boldsymbol{q_i} \odot \boldsymbol{h_{u,i}})^T, \tag{5.1}$$

where $\boldsymbol{p_u}$ denotes the latent factor of user u, $\boldsymbol{q_i}$ represents the latent factor of item i, $\boldsymbol{h_{u,i}}$ is a vector containing 1s in the beginning followed by 0s, and $\odot$ represents the elementwise Hadamard product between the vectors. The vector $\boldsymbol{h_{u,i}}$ is used to select the ranks that are *active* for the (u, i) tuple. The 1s in $\boldsymbol{h_{u,i}}$ denote the active ranks for the (u, i) tuple.

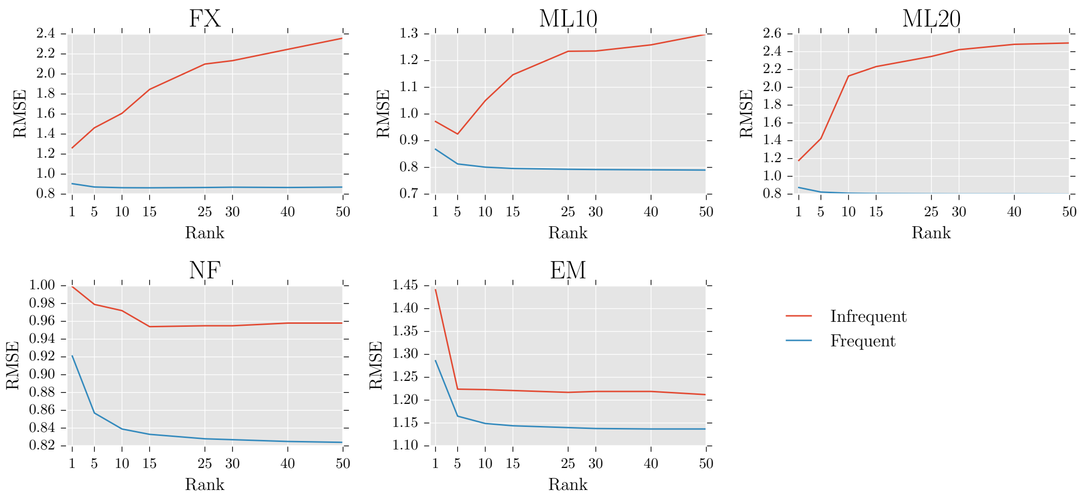


Figure 5.1: Test RMSE of the frequent and infrequent items in real datasets.

5.3.1 Frequency adaptive truncation

A way to select the active ranks, i.e., $\boldsymbol{h_{u,i}}$ for a user-item rating is based on the frequency of the user and the item in the rating matrix. In this approach, for a given rating by a user on an item, first, we determine the number of ranks to be updated based on either the user or the item depending on the one having a lower number of ratings. In order to select the ranks, we normalize the frequency of the user and the item, and use a non-linear activation function, e.g., sigmoid function, to map this frequency of the user or the item in [0, 1]. Finally, we use the product of the output of the activation function and rank of the low-rank model as the number of active ranks selected for the (u, i) tuple. The number of active ranks to be selected is given by

$$k_{u,i} = \begin{cases} \frac{r}{1+e^{-k(f_u - z)}}, & \text{if } f_u \leq f_i \\ \frac{r}{1+e^{-k(f_i - z)}}, & \text{otherwise}, \end{cases} \tag{5.2}$$

where r is the rank of the low-rank model, i.e., dimension of the user and the item latent factors, f_u is the frequency of user u, f_i is the frequency of item i, k controls the steepness of the sigmoid function and z is the value of the sigmoid's midpoint. An

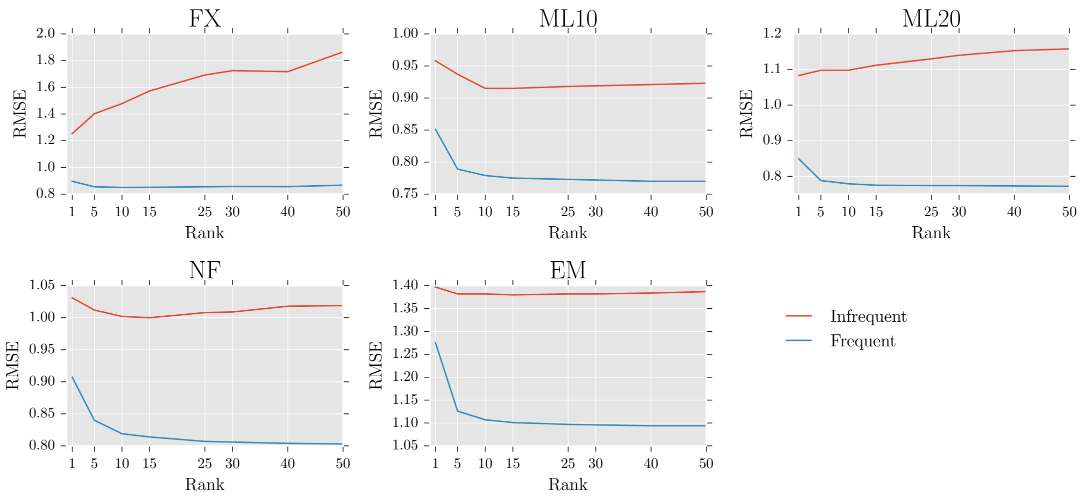


Figure 5.2: Test RMSE of the frequent and infrequent users in real datasets.

additional motivation for using a non-linear activation function, e.g., sigmoid function, is that the shape of the plot in Figure 4.2 is non-linear. Hence, using such a function assists in identifying the users or the items for whom we may estimate only a few ranks accurately. The active ranks to be selected are given by

$$\boldsymbol{h_{u,i}[j]} = \begin{cases} 1, & \text{if } j \leq k_{u,i} \\ 0, & \text{otherwise.} \end{cases} \tag{5.3}$$

We will refer to this method as Truncated matrix factorization (TMF).

5.3.2 Frequency adaptive probabilistic truncation

An alternative way to select the active ranks is to assume that the number of active ranks follows a Poisson distribution with parameter $k_{u,i}$. This method is similar to Dropout [65] technique in neural networks, where parameters are selected probabilistically for updates during learning of the model. Similar to regularization it provides a way of preventing overfitting in learning of the model. The active ranks to be selected

Algorithm 1 Learn TMF

```
procedure LearnTMF
    $r \leftarrow$ rank of low-rank models
    $\eta \leftarrow$ learning rate
    $\lambda \leftarrow$ regularization parameter
    $k \leftarrow$ steepness constant
    $z \leftarrow$ mid-point
    $\mathcal{R} \leftarrow$ all users' ratings on items
    $f \leftarrow$ users' and items' frequency
    $iter \leftarrow 0$
    Init $P$, $Q$ with random values $\in$ [-0.01, 0.01]
    while iter $<$ maxIter or error on validation set decreases do
        for each $r_{u,i} \in \mathcal{R}$ do
            if $f_u \leq f_i$ then
                $k_{u,i} \leftarrow \frac{r}{1+e^{-k(f_u - z)}}$
            else
                $k_{u,i} \leftarrow \frac{r}{1+e^{-k(f_i - z)}}$
            end if
            $\boldsymbol{h_{u,i}} \leftarrow 1$
            for each do$j \in [1, r]$
                if $j > k_{u,i}$ then
                    $\boldsymbol{h_{ui}}[j] \leftarrow 0$
                end if
            end for
            $\hat{r}_{u,i} = \boldsymbol{p_u}(\boldsymbol{q_i} \odot \boldsymbol{h_{u,i}})^T$
            $e_{u,i} \leftarrow \hat{r}_{u,i} - r_{u,i}$
            for each $j \in [1, k_{u,i}]$ do
                $\boldsymbol{p_u}[j] \leftarrow \boldsymbol{p_u}[j] - \eta(2e_{u,i}q_i[j] + 2\lambda\boldsymbol{p_u}[j])$
                $\boldsymbol{q_i}[j] \leftarrow \boldsymbol{q_i}[j] - \eta(2e_{u,i}p_u[j] + 2\lambda\boldsymbol{q_i}[j])$
            end for
        end for
    end while
    return $P, Q$
end procedure
```

Algorithm 2 Learn TMF + Dropout

```
1: procedure LearnTMFDropout
2:     $r \leftarrow$ rank of low-rank models
3:     $\eta \leftarrow$ learning rate
4:     $\lambda \leftarrow$ regularization parameter
5:     $k \leftarrow$ steepness constant
6:     $z \leftarrow$ mid-point
7:     $\mathcal{R} \leftarrow$ all users' ratings on items
8:     $f \leftarrow$ users' and items' frequency
9:     $iter \leftarrow 0$
10:    Init $P$, $Q$ with random values $\in$ [-0.01, 0.01]
11:    while iter < maxIter or error on validation set decreases do
12:        for each $r_{u,i} \in \mathcal{R}$ do
13:            if $f_u \leq f_i$ then
14:                $k_{u,i} \leftarrow \frac{r}{1+e^{-k(f_u-z)}}$
15:            else
16:                $k_{u,i} \leftarrow \frac{r}{1+e^{-k(f_i-z)}}$
17:            end if
18:
19:            $\theta_{u,i} \sim Poisson(k_{u,i})$
20:            $\boldsymbol{h_{u,i}} \leftarrow 1$
21:            for each  do$j \in [1, r]$
22:                if $j > \theta_{u,i}$ then
23:                    $\boldsymbol{h_{ui}}[j] \leftarrow 0$
24:                end if
25:            end for
26:            $\hat{r}_{u,i} = \boldsymbol{p_u}(\boldsymbol{q_i} \odot \boldsymbol{h_{u,i}})^T$
27:            $e_{u,i} \leftarrow \hat{r}_{u,i} - r_{u,i}$
28:            for each $j \in [1, \theta_{u,i}]$ do
29:                $\boldsymbol{p_u}[j] \leftarrow \boldsymbol{p_u}[j] - \eta(2e_{u,i}q_i[j] + 2\lambda\boldsymbol{p_u}[j])$
30:                $\boldsymbol{q_i}[j] \leftarrow \boldsymbol{q_i}[j] - \eta(2e_{u,i}p_u[j] + 2\lambda\boldsymbol{q_i}[j])$
31:            end for
32:        end for
33:    end while
34:    return $P, Q$
35: end procedure
```

are given by

$$\boldsymbol{h_{u,i}}[\boldsymbol{j}] = \begin{cases} 1, & \text{if } j \leq \theta_{u,i} \\ 0, & \text{otherwise,} \end{cases}$$

where $\theta_{u,i} \sim Poisson(k_{u,i})$ and $k_{u,i}$ is given by Equation 5.2. We will call this method as Truncated matrix factorization with Dropout (TMF + Dropout).

5.3.3 Model learning

The parameters of the model, i.e., the user and the item latent factors, can be estimated by minimizing Equation 3.2 as described in Section 3. Algorithms 1 and 2 provides the detailed procedure for TMF and TMF + Dropout, respectively.

5.3.4 Rating prediction

After learning the model the predicted rating for a user u on a item i for TMF model is given by

$$\hat{r}_{u,i} = \boldsymbol{p_u}(\boldsymbol{q_i} \odot \boldsymbol{h_{u,i}})^T, \tag{5.4}$$

where the active ranks, i.e., $\boldsymbol{h_{u,i}}$, is given by Equation 5.3. The predicted rating for the user and the item under TMF + Dropout model is given by the least number of ranks for whom the cumulative distribution function (CDF) for Poisson distribution with parameter k_{ui} obtains approximately the value of 1. The active ranks, i.e., $\boldsymbol{h_{u,i}}$, used for prediction under TMF + Dropout are given by

$$\boldsymbol{h_{u,i}}[\boldsymbol{j}] = \begin{cases} 1, & \text{if } j \leq s \\ 0, & \text{otherwise,} \end{cases} \tag{5.5}$$

where s is the least number of ranks for whom the CDF, i.e, $P(x <= s) \approx 1$, $x \sim Poisson(k_{u,i})$ and $k_{u,i}$ is given by Equation 5.2.

5.4 Experimental Evaluation

In this section, we conduct experiments to demonstrate the effectiveness of the proposed methods on different rating datasets presented in Table 5.1.

5.4.1 Evaluation methodology

To evaluate the performance of the proposed methods we divided the available ratings in different datasets into training, validation and test splits by randomly selecting 20% of the ratings for each of the validation and the test splits. The validation split was used for model selection, and the model that was selected was used to predict ratings on the test split. We repeated this process three times and report the average RMSE across the runs.

In addition to computing RMSE obtained by different methods for the ratings in the test split, we also investigated the performance of the proposed approaches for the items and the users with a different number of ratings in the training split. To this end, we ordered the items and the users in increasing order by their number of ratings in training split and divided them equally into four buckets or quartiles. We will report the RMSE achieved by different methods for ratings in the test split for the users and the items in these quartiles.

5.4.2 Comparison methods

We compared the performance of our proposed approaches against the state-of-the-art MF method described in Chapter 3.

5.4.3 Model selection

We performed grid search to tune the dimensions of the latent factors, regularization hyper-parameters and sigmoid function's parameters, i.e., k and z. We searched for regularization weights (λ) in the range [0.001, 0.01, 0.1, 1, 10], dimension of latent factors (r) in the range [1, 5, 10, 15, 25, 50], steepness constant (k) in the range [1 5, 10, 20, 40] and mid-point (z) in the range [-0.75, -0.50, -0.25, 0, 0.25, 0.50, 0.75]. The final parameters were selected based on the performance on the validation split.

5.5 Results and Discussion

We evaluated the performance of MF method and the proposed methods, i.e., TruncatedMF (TMF) and TruncatedMF with Dropout (TMF + Dropout) on the real datasets presented in Table 5.1.

5.5.1 Performance for rating prediction on real datasets

We investigated the performance of the different methods for the task of rating predictions on different rating datasets. Table 5.3 shows the best performance achieved by the different methods on the datasets for different ranks. As can be seen in the table, the proposed approaches outperform the MF method for the task of rating predictions in different datasets. The performance is significantly better for FX dataset. The better performance of the proposed methods for FX is because in this dataset there exists either the users or the items that have few ratings and the proposed approaches are effective in selecting the few ranks that can be estimated accurately for these users or items. Also, for NF and EM datasets which have a high average number of ratings for the users and the items the performance of the proposed approaches is similar to that of the MF method. This is because the proposed methods use all the available ranks as active ranks in such a dataset and hence essentially reduces to MF method.

Additionally, among the proposed methods, TMF + Dropout outperforms the TMF, and this improved performance of TMF + Dropout illustrates its effectiveness in preventing overfitting and generating better predictions compared to that of the TMF method.

5.5.2 Performance for the users and the items with different number of ratings

We investigated how does the performance of the MF and the proposed methods vary for the users and the items with the different number of ratings. Table 5.4 shows the performance of the different methods for different quartiles of the users and the items ordered by their frequency. As can be seen in the table, the proposed methods outperform the MF method for lower quartiles for most of the datasets. The performance of the proposed methods is significantly better for lower quartiles in FX, ML10 and

ML20 datasets, as these datasets contain users or items with few ratings. The better performance of the proposed methods for the users and the items with few ratings is because unlike MF the proposed approaches consider only a subset of ranks that are estimated accurately while trying to predict ratings that involve infrequent users or items. Also, since NF and EM datasets has a high average number of ratings, the proposed approaches use all the ranks as active ranks thereby giving similar performance as that of the MF method for all the quartiles.

Also, similar to our results for overall rating prediction the TMF + Dropout method outperforms the TMF method for different quartiles for most of the datasets thereby demonstrating its advantage over the TMF method.

5.6 Conclusion

In this chapter, we learned that the user or items with few ratings may not have sufficient ratings to estimate the low-rank models accurately thereby leading to an error in predictions and thus affecting item recommendations. Based on these insights we presented TruncatedMF in Section 5.3, which considers the frequency of both the user and the item to select a subset of ranks to estimate the rating of the user on the item accurately. The experiments on real datasets show that TruncatedMF outperforms the state-of-the-art MF method for rating predictions for the users and the items having few ratings in the user-item rating matrix.

Table 5.3: Test RMSE for real datasets

	EM			FX		
Rank	MF	TMF	TMF	MF	TMF	TMF
			+			+
			Dropout			Dropout
1	1.282	NA	NA	0.903	NA	NA
5	1.158	1.158	1.157	0.871	0.860	0.860
10	1.142	1.142	1.142	0.866	0.851	0.849
15	1.138	1.138	1.139	0.864	0.852	0.847
25	1.134	1.134	1.135	0.864	0.859	0.853
30	1.132	1.133	1.133	0.864	0.862	0.856
40	1.132	1.131	1.132	0.864	0.862	0.861
50	1.131	1.131	1.131	0.864	0.862	0.867

	ML20			NF		
Rank	MF	TMF	TMF	MF	TMF	TMF
			+			+
			Dropout			Dropout
1	0.875	NA	NA	0.926	NA	NA
5	0.824	0.824	0.822	0.863	0.863	0.864
10	0.813	0.813	0.809	0.845	0.845	0.844
15	0.810	0.810	0.805	0.839	0.839	0.838
25	0.809	0.808	0.804	0.834	0.834	0.833
30	0.809	0.808	0.804	0.832	0.833	0.832
40	0.809	0.807	0.804	0.831	0.831	0.831
50	0.809	0.806	0.804	0.830	0.829	0.829

	ML10		
Rank	MF	TMF	TMF
			+
			Dropout
1	0.870	NA	NA
5	0.815	0.815	0.814
10	0.804	0.804	0.802
15	0.800	0.799	0.798
25	0.798	0.797	0.796
30	0.797	0.797	0.796
40	0.796	0.796	0.795
50	0.796	0.795	0.795

Table 5.4: Test RMSE of the proposed approaches for different datasets. We also show the RMSE for the users and the items in different quartiles created in increasing order by their frequency. Q1 refers to the quartile containing the least frequent users or items followed by remaining in Q2, Q3, and Q4.

	EM			FX		
	MF (Rank 50)	TMF (Rank 50)	TMF + Dropout (Rank 50)	MF (Rank 15)	TMF (Rank 10)	TMF + Dropout (Rank 15)
Overall	1.131	1.131	1.131	0.864	0.851	0.847
Item Q1	1.212	1.213	1.212	1.302	1.252	1.256
Item Q2	1.105	1.104	1.105	0.961	0.944	0.944
Item Q3	1.086	1.086	1.087	0.800	0.785	0.780
Item Q4	1.137	1.137	1.137	0.864	0.851	0.847
User Q1	1.387	1.384	1.384	1.292	1.247	1.260
User Q2	1.200	1.201	1.200	1.177	1.144	1.151
User Q3	1.156	1.157	1.156	0.974	0.964	0.964
User Q4	1.094	1.093	1.095	0.853	0.841	0.836

	ML20			NF		
	MF (Rank 25)	TMF (Rank 50)	TMF + Dropout (Rank 25)	MF (Rank 50)	TMF (Rank 50)	TMF + Dropout (Rank 50)
Overall	0.809	0.806	0.804	0.830	0.829	0.829
Item Q1	2.347	2.390	2.115	0.958	0.959	0.956
Item Q2	1.396	1.435	1.123	0.935	0.935	0.932
Item Q3	0.867	0.874	0.851	0.887	0.887	0.884
Item Q4	0.804	0.801	0.801	0.824	0.823	0.823
User Q1	1.130	1.119	1.078	1.019	1.019	1.014
User Q2	0.967	0.969	0.968	0.923	0.923	0.921
User Q3	0.856	0.859	0.863	0.859	0.859	0.858
User Q4	0.774	0.770	0.769	0.803	0.802	0.802

	ML10		
	MF (Rank 40)	TMF (Rank 50)	TMF + Dropout (Rank 40)
Overall	0.796	0.795	0.795
Item Q1	1.259	1.250	1.198
Item Q2	0.855	0.853	0.848
Item Q3	0.811	0.810	0.809
Item Q4	0.791	0.790	0.791
User Q1	0.921	0.919	0.915
User Q2	0.864	0.864	0.863
User Q3	0.820	0.820	0.820
User Q4	0.770	0.770	0.770

Chapter 6

User-specific feature-based factorized bilinear similarity model for cold-start Top-n item recommendation

Recommending new items to existing users has remained a challenging problem due to the absence of user's past preferences for these items. The user personalized non-collaborative methods based on item features can be used to address this item *cold-start* problem. These methods rely on similarities between the target item and user's previous preferred items. While computing similarities based on item features, these methods overlook the interactions among the features of the items and consider them independently. Modeling interactions among features can be helpful as some features, when considered together, provide a stronger signal on the relevance of an item when compared to case where features are considered independently. In this work we introduce the User-specific Feature-based factorized Bilinear Similarity Model (UFBSM), which learns a *low-rank* user personalized *bilinear* similarity model to generate Top-n recommendation of new items. UFBSM model leverages all available information across users to model these interactions among features. Results on benchmark dataset shows that UFBSM can improve upon traditional linear collaborative methods for cold-start

Top-n item recommendation.

6.1 Introduction

Top-n recommender systems are used to identify from a large pool of items those n items that are the most relevant to a user and have become an essential personalization and information filtering technology. They rely on the historical preferences that were either explicitly or implicitly provided for the items and typically employ various machine learning methods to build *content-agnostic* predictive models from these preferences. However, when new items are introduced into the system, these approaches cannot be used to compute personalized recommendations, because there are no prior preferences associated with those items. As a result, the methods used to recommend new items, referred to as (item) *cold-start* recommender systems, in addition to the historical information, take into account the characteristics of the items being recommended; that is, they are *content aware*. The items' characteristics are typically captured by a set of domain-specific features. For example, a movie may have features like genre, actors, and plot keywords; a book typically has features like content description and author information. These item features are intrinsic to the item and as such they do not depend on historical preferences.

In this work, we extend UFSM, previously described in Section 3, in order to account for interactions between the different item features. We believe that such interactions are important and quite common. For example, in an e-commerce website, the items that users tend to buy are often designed to go well with previously purchased items (e.g., a pair of shoes that goes well with a dress). The set of features describing items of different type will be different (e.g., shoe material and fabric color) and as such a linear model can not learn from the data that for example a user prefers to wear leather shoes with black clothes. Being able to model such dependencies can lead to item cold-start recommendation algorithms that achieve better performance.

Towards this goal, we present a method called **U**ser-specific **F**eature-based factorized **B**ilinear **S**imilarity **M**odel (UFBSM) that uses bilinear models to capture pairwise dependencies between the features. Like UFSM, UFBSM estimates a user-specific similarity function by linearly combining a small number of global similarity functions.

However, unlike UFSM's linear global similarity functions, UFBSM's global similarity functions are bilinear. A challenge associated with such bilinear models is that the number of parameters that needs to be estimated becomes quadratic on the dimensionality of the item's feature space, which is problematic given the very sparse training data. UFBSM overcomes this challenge by assuming that the pairwise relations can be modeled as a combination of a linear component and a low rank component. The linear component allows it to capture the direct relations between the features whereas the low rank component allows it to capture the pairwise relations. The parameters of these models are estimated using stochastic gradient descent and a ranking loss function based on Bayesian personalized ranking (BPR) that optimizes the area under the receiver operating characteristic curve.

We performed extensive empirical studies to evaluate the performance of the proposed UFBSM on two benchmark datasets and compared it against state-of-the-art models for item cold-start recommendation. In our results UFBSM optimized using the BPR loss function can improve upon other methods in terms of recommendation quality, especially for datasets in which there are considerable historical data for each item and datasets whose items are described by relatively small number of features.

This chapter is a generalized extension of Feature-based factorized Bilinear Similarity Model (FBSM) [66].

6.2 Feature-based similarity model

To solve the cold-start item recommendation problem the current state-of-art method UFSM models direct interaction between features. However, it fails to capture dependencies betweeen features. Modeling these dependencies can enable user to discover new items whose characteristics complement the items that he or she has liked in the past. To this end, we developed UFBSM that instead of learning linear feature-based similarity functions, it uses bilinear models to estimate the feature-based similarity of the items. Bilinear models estimate the contribution of all feature-pairs towards preferences of a user over items and as such allows UFBSM to identify relations between different pairs of features that correlate with a user liking an item. Thus, such a model can potentially identify items whose features complement each other and which can lead to better recommendations.

Following UFSM, UFBSM computes the preference score for a new item i for user u as

$$\hat{r}_{ui} = \sum_{j \in \mathcal{R}_u^+} \text{sim}_u(i, j), \tag{6.1}$$

where $\text{sim}_u(i, j)$ is the user-specific similarity function given by

$$\text{sim}_u(i, j) = \sum_{z=1}^{l} m_{u,z} \ \text{gsim}_z(i, j), \tag{6.2}$$

where $\text{gsim}_z(.)$ is the zth global similarity function, l is the number of global similarity functions, and $m_{u,z}$ is a scalar that determines how much the zth global similarity function contributes to u's overall similarity function.

However, unlike UFSM, in UFBSM the similarity between two items i and j under the zth global similarity function $\text{gsim}_z(.)$ is estimated as

$$\text{gsim}_z(i, j) = \boldsymbol{f}_i W_z \boldsymbol{f}_j^T, \tag{6.3}$$

where W_z is the interaction matrix of the bilinear model that captures the correlation among different pairs of item features under the global similarity function $\text{gsim}_z(.)$. The diagonal of W_z determines linear interactions among features while the off-diagonal elements of W_z capture the pairwise relations among features. The user-specific similarity function in Equation 6.2 can be expanded as

$$\text{sim}_u(i, j) = \sum_{z=1}^{l} m_{u,z} \, \boldsymbol{f}_i W_z \boldsymbol{f}_j^T. \tag{6.4}$$

A key challenge in estimating the bilinear model matrices W_zs is that the number of parameters that needs to be estimated is quadratic on the number of features. In order to overcome this challenge, we model W_z as the sum of diagonal weights and a low-rank approximation of the off-diagonal weights:

$$W_z = D_z + V_z^T V_z, \tag{6.5}$$

where D_z is a diagonal matrix and V_z is a matrix of rank h. Note that in Equation 6.5 the model assumes that W_z is symmetric. Using this low-rank approximation the number of parameters that need to be estimated is reduced significantly when compared with the full-rank formulation of Equation 6.3. The global similarity function $\mathrm{gsim}_z(.)$ is thus given by:

$$\begin{aligned}\mathrm{gsim}_z(i,j) &= \boldsymbol{f}_i W_z \boldsymbol{f}_j^T = \boldsymbol{f}_i (D_z + V_z^T V_z) \boldsymbol{f}_j^T \\ &= \boldsymbol{f}_i D_z \boldsymbol{f}_j^T + \sum_{k=1}^{n_F} \sum_{p=1}^{n_F} f_{ik} f_{jp} \boldsymbol{v_{z,k}} \cdot \boldsymbol{v_{z,p}}.\end{aligned} \tag{6.6}$$

The first part of Equation 6.6 captures the direct relations between features, whereas the second part of Equation 6.6 captures the effect of off-diagonal elements of W_z by inner product of latent factor of features. This also gives us a flexible model where we can regularize diagonal weights and feature latent factors separately.

Note that unlike the bilinear model discussed in Section 3, which models relations between user and item features [27], UFBSM is trying to find correlations within features of items itself. The advantage of modeling interactions among item features is especially attractive when there are no explicit user features available. Note that it is not hard to encode the user features in the proposed bilinear model such that the similarity function is parameterized by user features, and we leave a detailed study to an extension of this chapter.

6.2.1 Parameter Estimation

UFBSM is parameterized by $\Theta = [M, D_1, \ldots, D_l, V_1, \ldots, V_l]$, where $D_1, \ldots, D_l$, $V_1, \ldots, V_l$ are the parameters of the global similarity functions and M is $n_{\mathcal{U}} \times l$ matrix of user's weight on global similarity functions. The inputs to the learning process are: (i) the preference matrix R, (ii) the item-feature matrix F, and (iii) the dimension of latent factor of features. There are many loss functions that can be used to estimate Θ, among which the Bayesian Personalized Ranking (BPR) loss function [67] is designed especially for ranking problems. In the Top-n recommender systems, the predicted preference scores are used to rank the items in order to select the highest scoring n items. The BPR loss function [25, 67] tries to learn item preference scores such that the items that a user liked have higher preference scores than the ones he/she did not like, regardless of the actual item preference scores. Thus it can better model the problem than other loss functions such as least squares loss and in general lead to better empirical performance. Therefore, we adopted BPR as the loss function to estimate model parameters Θ and it is given by

$$\mathcal{L}_{bpr}(\Theta) \equiv -\sum_{u \in U} \sum_{\substack{i \in \mathcal{R}_u^+, \\ j \in \mathcal{R}_u^-}} \ln \sigma(\hat{r}_{ui}(\Theta) - \hat{r}_{uj}(\Theta)), \tag{6.7}$$

where $\sigma(x)$ is the sigmoid function and $\hat{r}_{ui}$ is the estimated value of the user u's preference for the item i. The estimated rating $\hat{r}_{ui}$ is given by

$$\hat{r}_{ui} = \sum_{j \in \mathcal{R}_u^+ \setminus i} \text{sim}_u(i, j), \tag{6.8}$$

which is identical to Equation 6.1 except that item i is excluded from the summation.

The model parameters $\Theta = [M, D_1, \ldots, D_z, V_1, \ldots, V_z]$ are estimated via an optimization of the following objective function:

$$\min_{\Theta} \mathcal{L}_{bpr}(\Theta) + \lambda \sum_{z=1}^{l} \|V_z\|_F^2 + \beta \sum_{z=1}^{l} \|D_z\|_2^2 + \gamma \|M\|_F^2, \tag{6.9}$$

where we penalize the frobenious norm of the model parameters in order to control the model complexity and improve its generalizability.

To optimize Equation 6.9 we used stochastic gradient descent(SGD) [15], which is well-suited for large-scale datasets. The update steps for D_z, V_z, m_{uz} are based on triplets (u, i, j) sampled from the training data. For each triplet, we need to compute the estimated relative rank as $\hat{r}_{uij} = \hat{r}_{ui} - \hat{r}_{uj}$. If we let $\tau_{u,ij} = 1/(1 + e^{\hat{r}_{u,ij}})$, then the updates for model parameters for each SGD iteration is given by:

$$D_z = D_z + \alpha_1 \Big(\tau_{u,ij} \nabla_{D_z} \hat{r}_{u,ij} - \beta D_z\Big), \tag{6.10}$$

$$\boldsymbol{v}_p^z = \boldsymbol{v}_p^z + \alpha_2 \Big(\tau_{u,ij} \nabla_{\boldsymbol{v}_p^z} \hat{r}_{u,ij} - \lambda \boldsymbol{v}_p^z\Big), and \tag{6.11}$$

$$\boldsymbol{m}_{u,z} = \boldsymbol{m}_{u,z} + \alpha_3 \Big(\tau_{u,ij} \nabla_{\boldsymbol{m}_{u,z}} \hat{r}_{u,ij} - \gamma \boldsymbol{m}_{u,z}\Big). \tag{6.12}$$

where α_1, α_2 and α_3 are the learning rates of the SGD.

6.2.2 Performance optimizations

In our approach, the direct computation of gradients is time-consuming and is prohibitive when we have high-dimensional item features. For example, the relative rank $\hat{r}_{u,ij}$ given by

$$\begin{aligned}\hat{r}_{u,ij} = \sum_{z=1}^{l} \boldsymbol{m}_{u,z} \Bigg(&\Big(\boldsymbol{f}_i (D_z + V_z^T V_z)\Big(\Big(\sum_{q \in \mathcal{R}_u^+} \boldsymbol{f}_q\Big) - \boldsymbol{f}_i\Big)^T\Big) - \\ &\Big(\boldsymbol{f}_j (D_z + V_z^T V_z)\Big(\sum_{q \in \mathcal{R}_u^+} \boldsymbol{f}_q^T\Big)\Big)\Bigg),\end{aligned} \tag{6.13}$$

has complexity of $O(|\mathcal{R}_u^+| n_F h l)$, where h is the dimensionality of latent factors, n_F is the number of features and l is number of similarity functions.

To efficiently compute these, let $\boldsymbol{f_u} = \sum_{q \in \mathcal{R}_u^+} \boldsymbol{f}_q$, which can be precomputed once for all users. Then, Equation 6.13 becomes

$$\begin{aligned}\hat{r}_{u,ij} &= \sum_{z=1}^{l} \boldsymbol{m}_{u,z} \Bigg(\Big(\boldsymbol{f}_i (D_z + V_z^T V_z)(\boldsymbol{f}_u - \boldsymbol{f}_i)^T\Big) - \\ &\qquad \Big(\boldsymbol{f}_j (D_z + V_z^T V_z)\boldsymbol{f}_u^T\Big)\Bigg) \\ &= \sum_{z=1}^{l} \boldsymbol{m}_{u,z} \Bigg(\Big((\boldsymbol{f}_i - \boldsymbol{f}_j) D_z \boldsymbol{f}_u^T - \boldsymbol{f}_i D_z \boldsymbol{f}_i^T\Big) + \\ &\qquad \Big((\boldsymbol{f}_i - \boldsymbol{f}_j)(V_z^T V_z)\boldsymbol{f}_u^T - \boldsymbol{f}_i V_z^T V_z \boldsymbol{f}_i^T\Big)\Bigg) \\ &= \sum_{z=1}^{l} \boldsymbol{m}_{u,z} \Bigg(\Big(\boldsymbol{\delta}_{ij} D_z \boldsymbol{f}_u^T - \boldsymbol{f}_i D_z \boldsymbol{f}_i^T\Big) + \\ &\qquad \Big(\boldsymbol{\delta}_{ij}(V_z^T V_z)\boldsymbol{f}_u^T - \boldsymbol{f}_i V_z^T V_z \boldsymbol{f}_i^T\Big)\Bigg) \\ &= \sum_{z=1}^{l} \boldsymbol{m}_{u,z} \Bigg(\Big(\boldsymbol{\delta}_{ij} D_z \boldsymbol{f}_u^T - \boldsymbol{f}_i D_z \boldsymbol{f}_i^T\Big) + \\ &\qquad \Big((V_z \boldsymbol{\delta}_{ij}^T)(V_z \boldsymbol{f}_u^T)^T - (V_z \boldsymbol{f}_i^T)(V_z \boldsymbol{f}_i^T)^T\Big)\Bigg),\end{aligned}$$

where $\boldsymbol{\delta}_{ij} = \boldsymbol{f}_i - \boldsymbol{f}_j$.

The complexity of computing the relative rank then becomes $O(ln_F h)$, which is lower than complexity of Equation 6.13.

The gradient with respect to user weight is given by

$$\frac{\partial \hat{r}_{u,ij}}{\partial \boldsymbol{m}_{u,z}} = \Bigg(\Bigg(\boldsymbol{\delta}_{ij} D_z \boldsymbol{f}_u^T - \boldsymbol{f}_i D_z \boldsymbol{f}_i^T\Bigg) + \Bigg((V_z \boldsymbol{\delta}_{ij}^T)(V_z \boldsymbol{f}_u^T)^T - (V_z \boldsymbol{f}_i^T)(V_z \boldsymbol{f}_i^T)^T\Bigg)\Bigg), \tag{6.14}$$

which has a complexity of $O(n_F h)$.

The gradient of the diagonal component is given by

$$\frac{\partial \hat{r}_{u,ij}}{\partial D_z} = m_{u,z}\Big(\boldsymbol{\delta}_{ij} \otimes \boldsymbol{f}_u - \boldsymbol{f}_i \otimes \boldsymbol{f}_i\Big), \tag{6.15}$$

where $\otimes$ represents elementwise scalar product. The complexity of Equation 6.15 is given by $O(n_F)$.

The gradient of the low rank component is given by

$$\frac{\partial \hat{r}_{u,ij}}{\partial \boldsymbol{v}_p^z} = m_{u,z}\Big(\boldsymbol{\delta}_{ij,p}(V_z \boldsymbol{f}_{\boldsymbol{u}}^T) + \boldsymbol{f}_{up}(V_z \boldsymbol{\delta}_{ij}^T) - 2\boldsymbol{f}_{ip}(V_z \boldsymbol{f}_{\boldsymbol{i}}^T)\Big), \tag{6.16}$$

whose complexity is $O(n_F h)$.

Hence, the complexity of gradient computation for all the parameters is given by $O(l(n_F h + n_F + n_F h)) \approx O(ln_F h)$. We were able to obtain both the estimated relative rank and all the gradients in $O(ln_F h)$, which is linear with respect to feature dimensionality as well as the size of latent factors and the number of global similarity functions. This allows the UFBSM$_{bpr}$ to process large-scale datasets.

We summarize the proposed UFBSM$_{bpr}$ algorithm in Algorithm 3.

6.3 Experimental Evaluation

In this section we perform experiments to demonstrate the effectiveness of the proposed algorithm.

Algorithm 3 UFBSM$_{bpr}$-Learn

```
procedure UFBSM $_{bpr}$_LEARN
    $l \leftarrow$ Number of global similarity functions
    $\lambda \leftarrow V_1, \dots, V_l$ regularization weights
    $\beta \leftarrow D_1, \dots, D_l$ regularization weights
    $\gamma \leftarrow$ M regularization weight
    $\alpha_1, \alpha_2, \alpha_3 \leftarrow D$'s, $V$'s and $M$'s learning rates
    Initialize $\Theta = [D_1, \dots, D_l, V_1, \dots, V_l, M]$ randomly

    while not converged do
        for each user $u$ do
            sample a pair $(i, j)$ s.t. $i \in \mathcal{R}_u^+$, $j \in \mathcal{R}_u^-$
            compute $\hat{r}_{u,ij} = \hat{r}_{ui} - \hat{r}_{uj}$
            compute $\nabla_{D_z} \hat{r}_{u,ij}$
            compute $\nabla_{\boldsymbol{v}_p^z} \hat{r}_{u,ij}$
            compute $\nabla_{\boldsymbol{m}_{u,z}} \hat{r}_{u,ij}$
            update $D_z$ using (6.10)
            update $\boldsymbol{v}_p^z \forall p$ using (6.11)
            update $\boldsymbol{m}_{u,z}$ using (6.12)
        end for
    end while

    return $\Theta = [D_1, \dots, D_l, V_1, \dots, V_l, M]$
end procedure
```

6.3.1 Datasets

We used four datasets to evaluate the performance of UFBSM.

CiteULike (CUL)[1] aids researchers by allowing them to add scientific articles to their libraries. For users of the CUL, the articles present in their library are considered as preferred articles i.e., 1 in a preference matrix while rest are considered as implicit 0 preferences.

MovieLens-HetRec (ML-HR (genre)) is the dataset described in [68]. The ratings were binarized by treating all ratings greater than 2 as 1 and below or equal to 2 as 0. The movie genres were used as the item's content.

Amazon Books (ABB) is a dataset collected from amazon about best-selling books and their ratings. The ratings are binarized by treating all ratings greater than equal

[1] http://citeulike.org/

Table 6.1: Statistics for the datasets used for testing

Dataset	**users**	**items**	**features**	**preferences**	**prefs/user**	**prefs/item**	**density**
ML-HR (genre)	2,113	10,109	20	855,598	404.9	84.6	4.01%
CUL	3,272	21,508	6,359	180,622	55.2	8.4	0.13%
Book-Crossing	17,219	36,546	8,946	574,127	33.3	15.7	0.09%
ABB	13,097	11,077	5,766	175,612	13.4	15.9	0.12%

to 3 as 1 and ratings below 3 as 0. Each is accompanied with a description which was used as item's content.

Book-Crossing (BX) dataset is extracted from Book Crossing data [69] such that user has given at least four ratings and each book has received the same amount of ratings. Description of these books were collected from Amazon using ISBN and were used as item features.

Various statistics about these datasets are shown in Table 6.1. Also comparing the densities of the datasets we can see that the MovieLens-HetRec dataset have significantly higher density than other dataset. For the ABB, CUL and Book-Crossing dataset, the words that appear in the item descriptions were collected, stop words were removed and the remaining words were stemmed to generate the terms that were used as the item features. All words that appear in less than 20 items and all words that appear in more than 20% of the items were omitted. The remaining words were represented with TF-IDF scores. The item feature matrix was normalized row-wise in datasets.

6.3.2 Comparison methods

We compared UFBSM against non-collaborative personalized user modeling methods and collaborative methods.

1. **Cosine-Similarity (CoSim)** This is a personalized user-modeling method. The preference score of user u on target item i is estimated using Equation 6.8 by using *Cosine Similarity* to compute similarity between the items.

2. **User-specific Feature-based Similarity Models (UFSM)** As mentioned before in Section 3, this method [26] learns personalized user model by using all the past preferences from users across the dataset. It outperformed other state of

the art collaborative latent factor based methods e.g., RLFM [23], AFM [25] by significant margin.

3. **RLFMI** We used the Regression-based Latent Factor Modeling(RLFM) technique implemented in LibFM [70] that accounts for inter-feature interactions. We used LibFM with SGD learning to obtain results.

6.3.3 Evaluation Methodology

For each dataset we split the corresponding user-item preference matrix R into three matrices R_{train}, R_{val} and R_{test}. R_{train} contains a randomly selected 60% of the columns (items) of R, and the remaining columns were divided equally among R_{val} and R_{test}. Since items in R_{test} and R_{val} are not present in R_{train}, this allows us to evaluate the methods for item cold-start problems as users in R_{train} do not have any preferences for items in R_{test} or R_{val}. The models are learned using R_{train} and the best model is selected based on its performance on the validation set R_{val}. The selected model is then used to estimate the preferences over all items in R_{test}. For each user the items are sorted in decreasing order and the first n items are returned as the Top-n recommendations for each user. The evaluation metrics as described later are computed using these Top-n recommendation for each user.

After creating the train, validation and test split, there might be some users who do not have any items in the validation or the test split. In that case we evaluate the performance on the splits for only those users who have at least one item in the corresponding test split. This split-train-evaluate procedure is repeated three times for each dataset and the evaluation metric scores are averaged over these runs before being reported in results.

We used two metrics to assess the performance of the various methods: Recall at n (Rec@n) and Discounted Cumulative Gain at n (DCG@n). Given the list of the Top-n recommended items for user u, Recall@n measures how many of the items liked by u appeared in that list, whereas the DCG@n measures how high the relevant items were placed in the list. The Recall@n is defined as

$$REC@n = \frac{|\{\text{Items liked by user}\} \cap \{\text{Top-n items}\}|}{|\text{Top-n items}|}$$

The DCG@n is defined as

$$DCG@n = \text{imp}_1 + \sum_{p=2}^{n} \frac{\text{imp}_p}{\log_2(p)},$$

where the imp_p of the item with rank p in the Top-n list is

$$\text{imp}_p = \begin{cases} 1/n, & \text{if item at rank } p \in R^+_{u,test} \\ 0, & \text{if item at rank } p \notin R^+_{u,test}. \end{cases}$$

The main difference between Recall@n and DCG@n is that DCG@n is sensitive to the rank of the items in the Top-n list. Both the Recall@n and the DCG@n are computed for each user and then averaged over all the users.

6.3.4 Model Training

UFBSM's model parameters are estimated using training set R_{train} and validation set R_{val}. After each major SGD iteration of Algorithm 3 we compute the Recall@n on validation set and save the current model if current Recall@n is better than those computed in previous iterations. The learning process ends when the optimization objective converges or no further improvement in validation recall is observed for 10 major SGD iterations. At the end of learning process we return the model that achieved the best Recall@n on the validation set.

To estimate the model parameters of UFBSM$_{bpr}$, we draw samples equal to the number of preferences in R for each major SGD iteration. Each sample consists of a user, an item preferred by user and an item not preferred by user. If a dataset does not contain items not preferred by user then we sample from items for which his preference is not known.

6.4 Results and Discussion

In this section we will compare UFBSM with the other competing methods and discuss the effect of increasing the number of global similarity functions and the dimension of feature's factor. We will also analyze pairs of feature whose bilinear interaction contributes the most towards the performance on the dataset ML-HR (genre).

6.4.1 Comparison with previous methods

We compared the performance of UFBSM with other methods described in Section 6.3.2. Results are shown in Table 6.2 for different datasets. We tried different values for various parameters e.g., latent factors and regularization parameters associated with methods and report the best results found across datasets.

These results show that the relative performance of the UFBSM over UFSM is dataset dependent. For some datasets, UFBSM is able to improve the performance whereas for the others it does not lead to any improvements.

In order to better characterize and understand the nature of the datasets for which UFBSM leads to better results, we analyzed for each dataset the set of users for which the UFBSM leads to an increase, no change and a decrease in performance over UFSM. We present these results in Table 6.3. For the ML-HR (genre) dataset more users benefited from changing to UFBSM from UFSM, while for the other datasets the number of users that benefited from both the methods remains the same. ML-HR (genre) in comparison to other datasets has more preferences per item, hence UFBSM takes advantage of availability of more data while UFSM fails to do the same.

6.4.2 Effect of increasing the number of global similarity functions

Table 6.4 shows the performance achieved by using the different number of global similarity functions across different datasets.

For high-dimensional feature datasets (CiteULike, ABB and Book-Crossing) the performance remains similar on increasing the number of global similarity functions. The UFBSM method performs reasonably well using fewer global similarity functions. Therefore similar to UFSM, UFBSM can capture diverse preferences from the users successfully.

For the low-dimensional feature dataset (ML-HR (genre)) the performance decreases with number of global similarity functions. UFBSM requires a single global similarity function to achieve the best performance on ML-HR (genre).

6.4.3 Effect of increasing the dimension of feature's factor

Table 6.5 shows the effect of increasing the dimension of feature's factor on the performance across different datasets. For all datasets smaller dimension of feature's factor are enough to achieve the best performance; however, it may use multiple global similarity functions for the same.

6.4.4 Pairwise feature interaction analysis

For ML-HR (genre) dataset we identified pairs of features (genres), such that the bilinear interaction among these pairs contribute most towards the performance of UFBSM. Some of these significant pairs of feature are shown in Table 6.6. We identified these pairs by removing the bilinear interaction between all the possible pair of features and report those pairs whose removal led to a significant change in the performance. The pairs which led to a decrease in the performance are the ones whose interaction contribute towards generating better recommendations, on the other hand the pairs which led to an increase in the performance are redundant e.g., Animation and Children. We looked at how often these feature pairs occur together for items and found that the frequency of these feature pairs' co-occurrence is small and that is the reason why a linear model (UFSM) fails to capture the interaction between these pairs, whereas a bilinear model (UFBSM) performs better.

6.4.5 Discussion

In our experiments the performance of the bilinear model is found to be dependent on the datasets. It outperformed the linear model by a significant margin when the dimension of the features in the datasets were small. The linear model is designed to recommend those items that have common features with the items preferred by the user. The bilinear model in addition to common features among items also take into account the features that do not co-occur among the items. The low dimensional feature dataset e.g., ML-HR (genre) contains features which are disjoint and do not co-occur frequently among the items. The high dimensional features in our datasets are derived from text and these may contain terms which co-occur frequently among the items. For example, in the task of scientific articles recommendation, consider a user who prefers articles related to *machine learning* and *high-performance computing*. The articles related to *high-performance computing* may contain terms that are related to *machine learning* e.g., an article describing parallel implementation of the LASSO method. Due to these terms the linear model can recommend articles that are related to machine learning from a high performance computing article. While in the task of movies recommendation the movies are presented by their corresponding genres, consider a user who prefers *children* and *IMAX* movies. Here the bilinear model can recommend *IMAX* movies to a person who prefers *children* movies while linear model can not perform such recommendations.

6.5 Conclusion

We presented here UFBSM for the personalized cold-start Top-n item recommendation. It tries to learn multiple global bilinear similarity functions between items, represented by their features, by using all the information available across users. It captures the interaction among the item features by using a bilinear model. The computation complexity of the bilinear model estimation is significantly reduced by modeling the similarity as sum of the diagonal component and off-diagonal component. The off-diagonal components are further estimated as dot product of latent spaces of features. Results on benchmark datasets shows that UFBSM can improve upon the existing linear collaborative methods used for the cold-start item recommendation.

Table 6.2: Performance of UFBSM and Other Techniques on different datasets

Method	CiteULike			ML-HR (genre)		
	Params	Rec@10	DCG@10	Params	Rec@10	DCG@10
CoSim	-	0.1791	0.0684	-	0.0050	0.0199
RLFMI	h=75, λ=0.001	0.0874	0.0424	h=35, λ=0.05	<u>0.012</u>	<u>0.0466</u>
UFSM$_{bpr}$	l=3, μ_1=0.25, γ=0.1	0.2017	0.0791	l=1, μ_1=10, γ=50	0.0074	0.0233
UFBSM$_{bpr}$	λ=0.25, β=10, γ=0.1, h=5, l=1	<u>0.2026</u>	0.0791	λ=50, β=50, γ=50, h=5, l=1	<u>0.012</u>	0.0418
		ABB			Book-Crossing	
CoSim	-	0.1732	0.0221	-	0.1485	0.0148
RLFMI	h=100, λ=0.01	0.014	0.0020	h=100, λ=0.01	0.063	0.0072
UFSM$_{bpr}$	l=3, μ_1=0.1, γ=0.1	<u>0.2054</u>	0.0280	l=2, μ_1=0.1, γ=0.01	0.1979	<u>0.0211</u>
UFBSM$_{bpr}$	λ=1, β=10000, γ=1, h=5, l=3	0.2046	<u>0.0283</u>	λ=0.1, β=1, γ=0.01, h=1, l=1	<u>0.1985</u>	<u>0.0211</u>

The "Params" column shows the main parameters for each method. For UFSM$_{bpr}$, l is the number of similarity functions, and μ_1 is the regularization parameter. For UFBSM$_{bpr}$, l is the number of similarity functions, λ, β and γ are regularization parameters and h is dimension of feature latent factors. The "Rec@10" and "DCG@10" columns show the values obtained for these evaluation metrics. The entries that are underlined represent the best performance obtained for each dataset.

Table 6.3: User level investigation for datasets

Dataset	UFBSM against UFSM	users	items	average user preferences	average item preferences
ML-HR (genre)	BETTER	897	5791	348	54
	SAME	1149	5526	154	32
	WORSE	62	4403	435	6
ABB	BETTER	99	3530	48	1
	SAME	6178	9074	15	10
	WORSE	85	1728	23	1
CiteULike	BETTER	32	2491	94	1
	SAME	3136	12901	32	8
	WORSE	30	2170	82	1
Book-Crossing	BETTER	18	4044	260	1
	SAME	2838	34042	95	8
	WORSE	8	3313	428	1

Table 6.4: Effect of increasing number of global similarity functions

No. of Global Similarity Functions	CiteULike		ML-HR (genre)		ABB		Book-Crossing	
	Rec@10	Factor Dim.	Rec@10	Factor Dim.	Rec@10	Factor Dim.	Rec@10	Factor Dim.
1	0.2026	5	0.0119	5	0.2053	0	0.1985	1
2	0.2023	5	0.0098	3	0.2052	0	0.1982	1
3	0.2017	0	0.0083	10	0.2054	0	0.1980	1
5	-	-	0.0076	10	-	-	-	-
7	-	-	0.0082	10	-	-	-	-

The “Rec@10” columns shows the best recall obtained for given number of global similarity functions. “Factor Dim.” column shows the dimension of factors at which best recall was achieved. “Factor Dim.” of 0 corresponds to UFSM method.

Table 6.5: Effect of increasing the dimension of feature's factor

Dimension of feature's factor	CiteULike		ML-HR (genre)		ABB		Book-Crossing	
	Rec@10	No. of global similarity func.	Rec@10	No. of global similarity func.	Rec@10	No. of global similarity func.	Rec@10	No. of global similarity func.
0	0.2017	1	0.0074	1	0.2054	3	0.1979	2
1	0.2015	1	0.0079	3	0.2044	3	0.1985	1
3	0.2021	1	0.0098	2	0.2045	3	0.1980	1
5	0.2026	1	0.0119	1	0.2046	3	0.1981	2
7	-	-	0.0115	1	-	-	-	-
10	-	-	0.009	1	-	-	-	-

The "Rec@10" columns shows the best recall obtained for given dimension of feature's factor.

Table 6.6: Significant feature pairs

Feature 1	Feature 2	Recall change(%)	Feature 1	Feature 2	Recall change(%)
Drama	IMAX	-28.672	Crime	IMAX	2.485
Children	Drama	-19.790	Animation	Fantasy	2.488
Drama	Musical	-13.953	Adventure	Musical	2.602
Fantasy	Drama	-11.596	Drama	Documentary	2.802
Children	IMAX	-6.185	Adventure	Animation	3.369
Children	Comedy	-6.118	Animation	Musical	4.084
Children	Musical	-5.641	Animation	Comedy	4.468
Drama	Western	-4.902	Adventure	Drama	4.476
Romance	IMAX	-3.912	Animation	Children	5.484
Animation	Film-Noir	-3.360	Animation	Drama	7.451

Chapter 7

Learning from Sets of Items in Recommender Systems

Most of the existing recommender systems use the ratings provided by users on individual items. An additional source of preference information is to use the ratings that users provide on sets of items. The advantages of using preferences on sets are twofold. First, a rating provided on a set conveys some preference information about each of the set's items, which allows us to acquire a user's preferences for more items that the number of ratings that the user provided. Second, due to privacy concerns, users may not be willing to reveal their preferences on individual items explicitly but may be willing to provide a single rating to a set of items, since it provides some level of information hiding. This chapter investigates two questions related to using set-level ratings in recommender systems. First, how users' item-level ratings relate to their set-level ratings. Second, how collaborative filtering-based models for item-level rating prediction can take advantage of such set-level ratings. We have collected set-level ratings from active users of Movielens on sets of movies that they have rated in the past. Our analysis of these ratings shows that though the majority of the users provide the average of the ratings on a set's constituent items as the rating on the set, there exists a significant number of users that tend to consistently either under- or over-rate the sets. We have developed collaborative filtering-based methods to explicitly model these user behaviors that can be used to recommend items to users. Experiments on real data and on synthetic data that resembles the under- or over-rating behavior in the real data, demonstrate that these models can recover the overall characteristics of the underlying data and predict the user's ratings on individual items.

7.1 Introduction

Recommender systems help consumers by providing suggestions that are expected to satisfy their tastes. They are successfully deployed in several domains such as e-commerce (e.g., Amazon, Ebay), multimedia content providers (e.g., Netflix, Hulu) and mobile app stores (e.g., Apple, Google Play). Collaborative filtering [71, 72] which takes advantage of users' past preferences to suggest relevant items, is one of the key methods used by recommender systems.

Most collaborative filtering approaches rely on past preferences provided by users on individual items. An additional source of preferences is the user's preferences on sets of items. Example of such set-level ratings includes ratings on song playlists, music albums, reading lists, and watchlists. A rating provided by the user on a set of items conveys some information about the user's preference on each of the set's items and, as a result, it is a mechanism by which some information about user's preferences can be acquired for many items. At the same time, due to privacy concerns, users that are not willing to explicitly reveal their true preferences on individual items may provide a single rating to a set of items, since it provides some level of information hiding.

This chapter investigates two questions related to using set-level preferences in recommender systems. First, how users' item-level ratings relate to the ratings that they provide on a set of items. Second, how collaborative filtering-based methods can take advantage of such set-level ratings towards making item-level rating predictions.

To answer the first question, we collected ratings on sets of movies from users of Movielens, a popular online movie recommender system[1]. Our analysis of these ratings leads to two key findings. First, for the majority of the users, the rating provided on a set can be accurately approximated by the average rating that they provided on the set's constituent items. Second, there is a considerable user population that tends to consistently either over- or under-rate the set, especially for sets that contain items

[1]www.movielens.org

on which the user's item-level ratings are diverse. Using these insights, we developed different models that can predict a user's rating on a set of items as well as on individual items. Furthermore, these methods can use ratings on both the sets and the items and lead to better results for the users that have either both or only one type of ratings. These methods solve these problems in a coupled fashion by estimating models to predict the item-level ratings and by estimating models that combine these individual ratings to derive set-level ratings.

The key contributions of the work are the following:

(i) collection and analysis of a dataset that contains users' ratings both on individual items and on various sets containing these items;

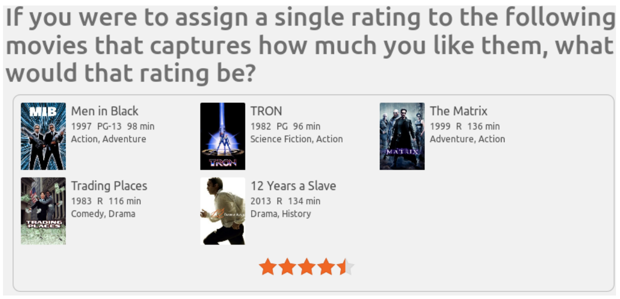
(ii) introduction of *Variance Offset Average Rating Model* (VOARM) and *Extremal Subset Average Rating Model* (ESARM) to model a user's consistency to over- or under-rate the set of items as a function of his/her ratings on the set's constituent items; and

(iii) development of collaborative filtering-based methods that take advantage of VOARM and ESARM in order to estimate users' preferences on sets of items as well as on individual items.

7.2 Movielens set ratings dataset

7.2.1 Data collection

Movielens is a recommender system that utilizes collaborative filtering algorithms to recommend movies to their users based on their preferences. We developed a set rating widget to obtain ratings on a set of movies from the Movielens users. The set rating widget could be rated from 0.5 to 5 with a precision of 0.5. For the purpose of data collection, we selected users who were active since January 2015 and have rated at least 25 movies. The selected users were encouraged to participate by contacting them via email. The sets of movies that we asked a user to rate were created by selecting five movies at random without replacement from the movies that they have already rated. Furthermore, we limited the number of sets a user can rate in a session to 50, though users can potentially rate more sets in different sessions. The set rating widget went live on February 2016 and, for the purpose of this study, we used the set ratings that were provided until April 2016.


If you were to assign a single rating to the following movies that captures how much you like them, what would that rating be?

Men in Black
1997 PG-13 98 min
Action, Adventure

TRON
1982 PG 96 min
Science Fiction, Action

The Matrix
1999 R 136 min
Adventure, Action

Trading Places
1983 R 116 min
Comedy, Drama

12 Years a Slave
2013 R 134 min
Drama, History

Figure 7.1: The interface used to elicit users' ratings on a set of movies.

7.2.2 Data processing

From the initially collected data, we removed users who have rated sets within a time interval of less than one second to avoid users who might be providing the ratings at random. After this pre-processing, we were left with ratings from 854 users over 29,516 sets containing 12,549 movies. Figure 7.2 shows the distribution of the number of sets rated by the users, which shows that roughly half of the users have rated at least 45 sets in a session.

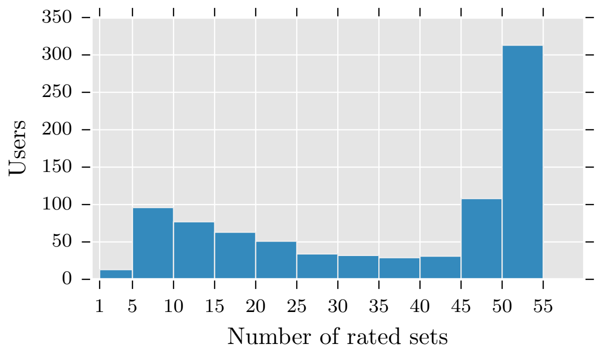
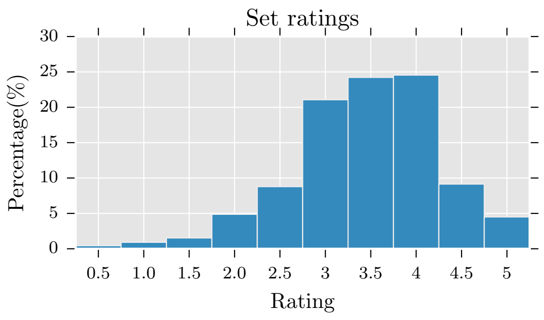


Figure 7.2: The distribution of number of sets rated by the users.



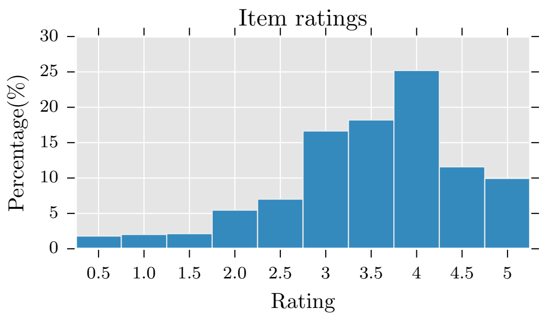


Figure 7.3: The distribution of the provided set ratings (left) and the ratings of their constituent items (right).

7.2.3 Analysis of the set ratings

We investigated whether ratings are distributed uniformly or if some ratings tend to appear more than others. Figure 7.3 (left) depicts the distribution of the collected ratings over all the sets. The majority of the ratings lie between 3.0 and 4.0. Since, by construction, we know the actual ratings that these users provided on the actual movies. Figure 7.3 (right) shows the distribution of the ratings of the movies that were contained in all these sets. By comparing these distributions we can see that the average item-rating (3.50) is somewhat higher than the average set-based rating (3.44) but the overall variance of the set-based ratings (0.65) is lower than that of the item ratings (1.01).

In order to analyze how consistent a user's rating on a set is with the ratings provided by the user on the movies in the set, we computed the difference of the average of the user's ratings on the items in the set and the rating assigned by a user to the set. We will refer to this difference as *mean rating difference* (MRD). Figure 7.4 (left) shows the distribution of the MRD values in our datasets. The majority of the sets have an MRD within a margin of 0.5 indicating that the users have rated them close to the average of their ratings on set's items. The remaining of the sets have been rated either significantly lower or higher from the average rating. We refer to these sets as the under- and the over-rated sets, respectively. Moreover, an interesting observation from the results in Figure 7.4 (right), is that the number of under-rated sets is more than that of the over-rated sets.

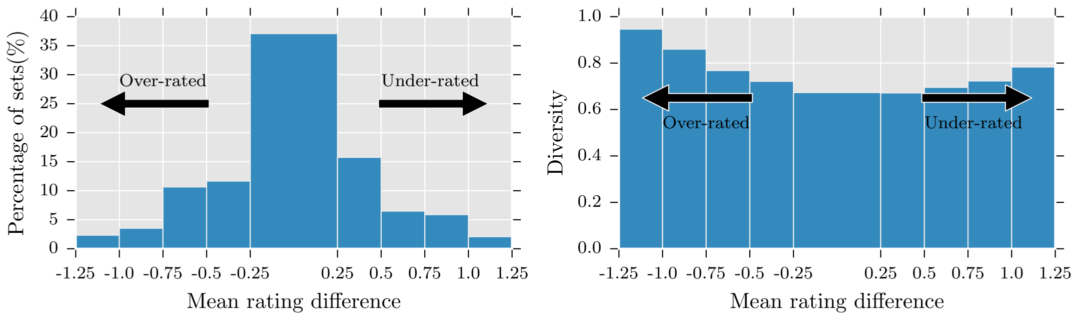


Figure 7.4: Histogram of percentage of sets (left) and diversity (right) against mean rating difference (MRD).

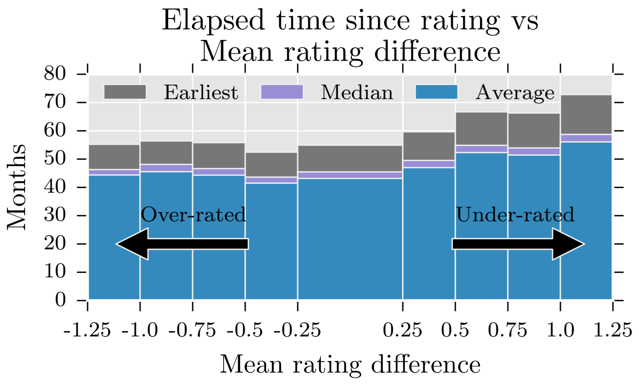


Figure 7.5: Histogram of elapsed time in months against mean rating difference.

In order to understand what can lead to a set being under- or over-rated, we investigated if the *diversity* of the ratings of the individual movies in a set could lead a user to under- or over-rate the set. We measured the diversity of a set as the standard deviation of the ratings that a user has provided to the individual items of the set. As shown in Figure 7.4 (right), the sets that contain more diverse ratings (i.e., higher standard deviations) tend to get under- or over-rated more often when compared to less diverse sets. This trend was found to be statistically significant (p-value of 0.01 using t-test).

Furthermore, we investigated whether the recently rated items carry more weight than the items rated a long time ago. To this end, we computed the difference between the timestamp of the earliest rating of the movies in the set and the year 2016, i.e.,

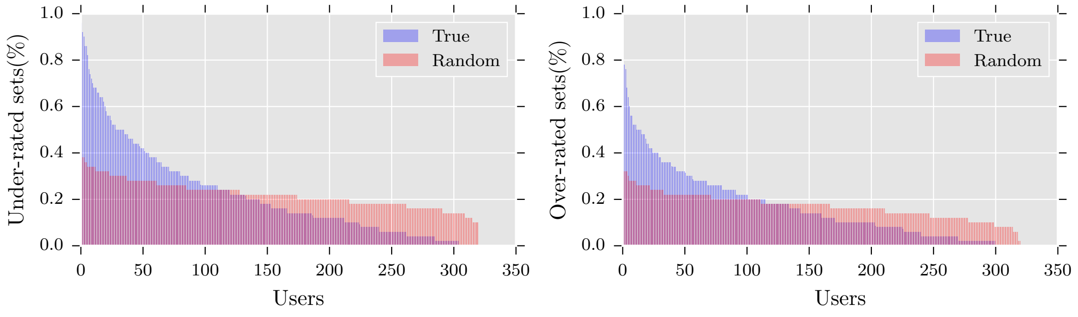


Figure 7.6: Fraction of under-rated and over-rated sets across users in true and random population.

when the users were asked to rate the sets. Similarly, we computed the median and average age of movies in a set. Interestingly as shown in Figure 7.5, the under-rated sets contained movies whose ratings were provided on average five years before the survey while the remaining sets contained the movies whose ratings were provided on average four years before the survey. This difference among the sets was found to be statistically significant (p-value $<$ 1e-16 using t-test). This suggests that the user's preference for a movie rated in the past carries lower weight than the recently rated movie. The user's higher preference for a recent movie is not surprising as it has been shown that the user tends to rate a movie close to the middle of the scale as the time between viewing a movie and rating it increases [73].

Additionally, we studied if there are users that tend to consistently under- or over-rate sets. To this end, we selected users who have rated at least 50 sets and computed the fraction of their under- and over-rated sets. We also computed the fraction of under- and over-rated sets across a random population of the same size. We generated this random population by randomly permuting the under-rated and over-rated sets across the users. Figure 7.6 shows the fraction of under- and over-rated sets for both the true and random population of user. In the true population, some users tend to under- or over-rate sets significantly more than that of the random population. Using the Kolmogorov-Smirnov 2 sample test, we found this behavior of true population to be statistically different (p-value $<$ 1e-16) from that of random population.

The above analysis reveals that our dataset contains users that when they are asked to assign a single rating to a set of items, some of them consistently assign a rating that is lower than the average of the ratings that they provided to the set's constituent items (they under-rate), whereas others assign a rating that is higher (they over-rate). Thus some users are very demanding (or picky) and tend to focus on the worst items in the set, whereas other users are less demanding and tend to focus on the best items in the set.

7.3 Methods

In this section, we will investigate different approaches that capture the user behavior of providing ratings on sets. We will describe various methods that use the set ratings alone or in combination with individual item ratings towards solving two problems: (i) predict a rating for a set of items, and (ii) predict a rating for individual items. Our methods solve these problems in a coupled fashion by estimating models for predicting the ratings that users will provide to the individual items and by estimating models that use these item-level ratings to derive set-level ratings.

7.3.1 Modeling users' ratings on sets

In order to estimate the preferences on individual items from the preferences on the sets, we need to make some assumptions on how a user derives a set-level rating from the ratings of the set's constituent items. Informed by our analysis of the data described in Section 7.2, we investigated three approaches of modeling that.

Average Rating Model (ARM)

The first approach assumes that the rating that a user provides on a set reflects his/her average rating on all the items in the set. Specifically, the estimated rating of user u on set $\mathcal{S}$ is given by

$$\hat{r}_u^{\mathcal{S}} = \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} r_{ui}. \tag{7.1}$$

As the analysis in Section 7.2 showed, such a model correlates well with the actual ratings that the users provided on majority of the sets, especially when the ratings of the constituent items are not very different.

Extremal Subset Average Rating Model (ESARM)

In order to capture the user-specific *pickiness* illustrated in Section 7.2.3, this approach postulates that a user rates a set by considering only a subset of the set's items. If a user tends to consistently under-rate each set, then that subset will contain some of each set's lowest-rated items. Analogously, if a user tends to consistently over-rate each set, then that subset will contain some of each set's highest-rated items. Moreover, this approach further postulates that given such subsets, the rating that a user will assign to the set as a whole will be the average of his/her ratings on the individual items of the subset. The parameter in this model that captures the level of a user's pickiness is the size of the subset and whether or not it will contain the least- or the highest-rated items. We will call these subsets having least- and highest- rated items as extremal subsets. The set rating of an extremely picky user will be determined by the average rating of one or two of the least rated items, whereas the set rating of a user that is not picky at all will be determined by the average rating of one or two of the highest rated items.

If e_i denotes the average rating of items in ith extremal subset and n_s denotes number of items in set $\mathcal{S}$, then $\langle e_1, \ldots, e_{n_s}, \ldots, e_{2n_s-1}\rangle$ represents the average rating on all the extremal subsets; for $1 \leq i \leq n_s$, e_i is the average rating of i least rated items, for $n_s \leq i \leq 2n_s - 1$, e_i is the average rating of the $2n_s - i$ highest rated items and e_{n_s} is the average rating of all the items in the set. Then $\hat{r}_u^{\mathcal{S}}$ is given by

$$\hat{r}_u^{\mathcal{S}} = \sum_{i=1}^{2n_s-1} w_{u,i} e_i, \tag{7.2}$$

where $w_{u,i}$ is a non-negative weight of user u on ith extremal subset and the weights sum to 1. The weight $w_{u,i}$ measures the influence of the items in ith extremal subset towards estimating the user's rating on set $\mathcal{S}$. One of the weights corresponds to the extremal subset that is responsible for majority of the user's rating on set, and it is higher than others, i.e.,

$$\begin{aligned}
&\sum_{i=1}^{2n_s-1} w_{u,i} = 1, \\
&w_{u,j} < w_{u,j+1}, \forall j < k, \\
&w_{u,j+1} < w_{u,j}, \forall j \geq k, \\
&w_{u,k} > c, c > 0,
\end{aligned} \tag{7.3}$$

where c is the minimum weight of the extremal subset having the highest contribution towards the user's rating on set.

Note that this model assume that the size of all the sets is the same, however it can be generalized to sets of different sizes by constructing the extremal subsets for fixed number of quantiles in a set.

Variance Offset Average Rating Model (VOARM)

This approach captures the user-specific *pickiness* by assuming that a user rates a set by considering both the average rating of the items in the set and also the diversity of the set's items. In this model, the set's rating is determined as the sum of the average rating of the set's items and a quantity that depends on the sets diversity (e.g., the standard deviation of the set's ratings) and the user's level of pickiness. If a user is very picky, that quantity will be negative and large, resulting to the set being (severely) under-rated. On the other hand, if a user is not picky at all, that quantity will be positive and large, resulting to the set being (severely) over-rated.

If β_u denotes the pickiness level of user u, then the estimated rating on a set is given by

$$\hat{r}_u^{\mathcal{S}} = \mu_s + \beta_u \sigma_s, \tag{7.4}$$

where μ_s and σ_s are the mean and the standard deviation of the ratings of items in the set $\mathcal{S}$. Both μ_s and σ_s are given by

$$\mu_s = \frac{1}{|S|} \sum_{i \in \mathcal{S}} r_{ui}, \quad \sigma_s = \sqrt{\frac{1}{|S|} \sum_{i \in \mathcal{S}} (r_{ui} - \mu_s)^2}. \tag{7.5}$$

7.3.2 Modeling user's ratings on items

In order to model a users' ratings on the items, similar to matrix factorization method described in Section 3, we assume that the underlying user-item rating matrix is low-rank, i.e., there is a low-dimensional latent space in which both the users and the items can be compared to each other. Thus, the estimated rating of user u on item i, i.e., $\hat{r}_{ui}$, is given by Equation 3.1.

7.3.3 Combining set and item models

Our goal is to estimate the item-level ratings by learning the user and item latent factors of Equation 3.1; however, the ratings that we have available from the users are at the set-level. In order to use the available set-level ratings, we need to combine Equation 3.1 with Equations 7.1, 7.2 and 7.4. To solve the problem, we assume that the actual item-level ratings used in Equations 7.1, 7.2 and 7.4 correspond to the estimated ratings given by Equation 3.1. Hence, the estimated set-level ratings in Equations 7.1, 7.2 and 7.4 are finally expressed in terms of the corresponding user and item latent factors.

7.3.4 Model learning

The parameters of the models that estimate item- and set-level ratings are the user and item latent vectors (p_u and q_i), in the case of ESARM method the users' weights on extremal subsets (W) and in the case of the VOARM method the user's pickiness level (β_u). These parameters are estimated using the user-supplied set-level ratings by minimizing a square error loss function given by

$$\mathcal{L}_{rmse}(\Theta) \equiv \sum_{u \in U} \sum_{s \in \mathcal{R}_u^s} (\hat{r}_u^{\mathcal{S}}(\Theta) - r_u^{\mathcal{S}})^2, \tag{7.6}$$

where Θ represents model parameters, U represents all the users, $\mathcal{R}_u^s$ contains all the sets rated by user u, $r_u^{\mathcal{S}}$ is the original rating of user u on set $\mathcal{S}$ and $\hat{r}_u^{\mathcal{S}}$ is the estimated rating of user u on set $\mathcal{S}$.

To control model complexity, we add regularization of the model parameters thereby

Algorithm 4 Learn ARM

```
1: procedure LearnARM
2:     $\eta \leftarrow$ learning rate
3:     $\lambda \leftarrow$ regularization parameter
4:     $\mathcal{R}^s \leftarrow$ all users' ratings on sets
5:     $iter \leftarrow 0$
6:     Init $P$, $Q$ with random values $\in [0,1]$
7:     while iter < maxIter or error on validation set decreases do
8:         $\mathcal{R}^s \leftarrow$ shuffle($\mathcal{R}^s$)
9:         for each $r_u^s \in \mathcal{R}^s$ do
10:            $\hat{r}_u^{\mathcal{S}} \leftarrow \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \boldsymbol{p}_u \boldsymbol{q}_i^T$
11:            $e_u^s \leftarrow (\hat{r}_u^{\mathcal{S}} - r_u^{\mathcal{S}})$
12:            $\boldsymbol{v}_k \in \mathbb{R}^k \leftarrow 0$
13:            for each item $i \in s$ do
14:                $\boldsymbol{v}_k \leftarrow \boldsymbol{v}_k + \boldsymbol{q}_i$
15:            end for
16:            $\boldsymbol{p}_u \leftarrow \boldsymbol{p}_u - \eta(\frac{e_u^s}{|\mathcal{S}|}\boldsymbol{v}_k + \lambda \boldsymbol{p}_u)$      ▷ Update user's latent representation
17:            for each item $i \in s$ do
18:                $\boldsymbol{q}_i \leftarrow \boldsymbol{q}_i - \eta(\frac{e_u^s}{|\mathcal{S}|}\boldsymbol{p}_u + \lambda \boldsymbol{q}_i)$      ▷ Update item's latent representation
19:            end for
20:        end for
21:        $iter \leftarrow iter + 1$
22:    end while
23: end procedure
```

Algorithm 5 Learn ESARM

```
1: procedure LEARNESARM
2:     $\eta \leftarrow$ learning rate
3:     $\lambda \leftarrow$ regularization parameter
4:     $\mathcal{R}^s \leftarrow$ all users' ratings on sets
5:     $n_s \leftarrow$ number of items in set
6:     $n_{es} \leftarrow 2n_s - 1$                    ▷ number of possible extremal subsets
7:     $iter \leftarrow 0$
8:     Init $P$, $Q$ with random values $\in [0,1]$
9:     Init $W$ with random values $\forall$ user $u \in U$, s.t., $\sum_{i=1}^{n_{es}} w_{u,i} = 1$
10:    while iter < maxIter or error on validation set decreases do
11:        $\mathcal{R}^s \leftarrow$ shuffle($\mathcal{R}^s$)
12:
13:        for each $r_u^{\mathcal{S}} \in \mathcal{R}^s$ do
14:            $\hat{r}_u^{\mathcal{S}} \leftarrow 0$
15:            $\mathcal{E}_s \leftarrow$ All possible extremal subsets for set $\mathcal{S}$
16:
17:            $\nabla p_u \in \mathbb{R}^f \leftarrow 0$
18:            for each subset $i \in \mathcal{E}_s$ do
19:                $\hat{e}_i \leftarrow 0,\ q_{sum} \in \mathbb{R}^f \leftarrow 0$
20:                for each item $j \in i$ do
21:                    $\hat{e}_i \leftarrow \hat{e}_i + p_u q_j^T,\ q_{sum} \leftarrow q_{sum} + q_j$
22:                end for
23:                $\hat{e}_i \leftarrow \frac{\hat{e}_i}{|i|},\ q_{sum} \leftarrow \frac{q_{sum}}{|i|}$
24:                $\hat{r}_u^{\mathcal{S}} \leftarrow \hat{r}_u^{\mathcal{S}} + w_{u,i}\hat{e}_i$
25:                $\nabla p_u \leftarrow \nabla p_u + w_{u,i} q_{sum}$
26:            end for
27:            $e_u^s \leftarrow (\hat{r}_u^{\mathcal{S}} - r_u^{\mathcal{S}})$
28:            $\nabla p_u \leftarrow 2e_u^s \nabla p_u + 2\lambda p_u$        ▷ update user's latent representation
29:            $p_u \leftarrow p_u - \eta \nabla p_u$
30:
31:            $\nabla q \leftarrow 2e_u^s p_u$
32:            for each subset $i \in \mathcal{E}_s$ do
33:                for each item $j \in i$ do
34:                    $q_j \leftarrow q_j - \eta(\frac{w_{u,i}\nabla q}{|i|} + 2\frac{\lambda}{n_s} q_j)$   ▷ update items' latent representation
35:                end for
36:            end for
37:        end for
38:
39:        for each $u \in U$ do
40:            Update $\boldsymbol{w}_u$ using constraint quadratic programming as described
41:            in Section 7.3.4.
42:        end for
43:
44:        $iter \leftarrow iter + 1$
45:    end while
46: end procedure
```

Algorithm 6 Learn VOARM

```
1: procedure LearnVOARM
2:     $\eta \leftarrow$ learning rate
3:     $\lambda \leftarrow$ regularization parameter
4:     $\mathcal{R}^s \leftarrow$ all users' ratings on sets
5:     $iter \leftarrow 0$
6:     Init $P$, $Q$ and $\beta$s with random values $\in$ [0,1]
7:     while iter < maxIter or error on validation set decreases do
8:         $\mathcal{R}^s \leftarrow \text{shuffle}(\mathcal{R}^s)$
9:         for each $r_u^{\mathcal{S}} \in \mathcal{R}^s$ do
10:            $\hat{\mu}_s \leftarrow \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \boldsymbol{p}_u^T \boldsymbol{q}_i$
11:            $\hat{\sigma}_s \leftarrow \epsilon + \sqrt{\frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} (\boldsymbol{p}_u^T \boldsymbol{q}_i - \hat{\mu}_s)^2}$
12:            $\hat{r}_u^{\mathcal{S}} \leftarrow \hat{\mu}_s + \beta_u \hat{\sigma}_s$
13:            $e_u^s \leftarrow (\hat{r}_u^{\mathcal{S}} - r_u^{\mathcal{S}})$
14:            $\boldsymbol{q} \in \mathbb{R}^f \leftarrow 0,\ \boldsymbol{v} \in \mathbb{R}^f \leftarrow 0$
15:            for each item $i \in \mathcal{S}$ do
16:                $\boldsymbol{q} \leftarrow \boldsymbol{q} + \boldsymbol{q}_i$
17:                $\boldsymbol{v} \leftarrow \boldsymbol{v} + (\boldsymbol{p}_u \boldsymbol{q}_i^T)\boldsymbol{q}_i$
18:            end for
19:            $\nabla p_u \leftarrow \frac{\boldsymbol{q}}{|\mathcal{S}|} + \frac{\beta_u \boldsymbol{v}}{\hat{\sigma}_s |\mathcal{S}|} - \frac{\beta_u \mu_s \boldsymbol{q}}{\hat{\sigma}_s |\mathcal{S}|}$
20:            $\nabla q \leftarrow \frac{2 e_u^s p_u}{\mathcal{S}}$
21:            for each item $i \in \mathcal{S}$ do
22:                $t \leftarrow 1 + \frac{\beta_u p_u^T q_i}{\hat{\sigma}_s} - \frac{\beta_u \mu_s}{\hat{\sigma}_s}$
23:                $q_i \leftarrow q_i - \eta(t \nabla q + 2\lambda q_i)$          ▷ Update item's latent representation
24:            end for
25:            $p_u \leftarrow p_u - \eta(2 e_u^s \nabla p_u + 2\lambda p_u)$          ▷ Update user's latent representation
26:            $\beta_u \leftarrow \beta_u - \eta(2 e_u^s \hat{\sigma}_s + 2\lambda \beta_u)$          ▷ Update $\beta_u$
27:        end for
28:        $iter \leftarrow iter + 1$
29:    end while
30: end procedure
```

leading to an optimization process of the following form

$$\underset{\Theta}{\text{minimize}}\, \mathcal{L}_{rmse}(\Theta) + \lambda(||\Theta||^2), \tag{7.7}$$

where λ is the regularization parameter. The L2-regularization is added to reduce the model complexity thereby improving its generalizability. This optimization problem can be solved by Stochastic Gradient Descent (SGD) [74] algorithm.

Note that for the ESARM model, we need to solve this optimization problem with linear and non-negative constraints on user weights $\boldsymbol{w_u}$. If we know the users' and the items' latent factors then the user weights can be determined by solving the Equation 7.7 as a constraint quadratic programming [75] for each user. We can determine a user's weights by solving multiple quadratic programs, each corresponding to a different extremal subset having the highest weight, and selecting the solution that has lowest RMSE over the user's sets. Hence, for ESARM we solve for W and $\{\boldsymbol{p_u}, \boldsymbol{q_i}\}$ alternately at each SGD iteration. In ESARM, the minimum weight of the extremal subset having highest contribution towards ratings on sets, i.e., c, can be specified in the range [0,1]. Also, in the VOARM method we add a fixed constant, i.e., ϵ in [0, 1], to computed σ for robustness. Algorithms 4, 5 and 6 shows the steps used to learn the ARM, ESARM and VOARM models, respectively.

If we also have ratings for the individual items, then we can incorporate these ratings into model estimation by treating each item as a set of size one.

7.4 Experimental Evaluation

7.4.1 Dataset

We evaluated the proposed methods on two datasets: (i) the dataset analyzed in Section 7.2, which will be referred to as ML-RealSets, and (ii) a set of synthetically generated datasets that allow us to assess how well the optimization algorithms can estimate accurate models and how their accuracy depends on various data characteristics.

The synthetic datasets were derived from the Movielens 20M dataset[2] [76] which contains 20 million ratings from approximately 229,060 users on 26,779 movies. For

[2]https://grouplens.org/datasets/movielens/20m/

experiment purposes, we created a synthetic low-rank matrix of rank 5 as follows. We started by generating two matrices $A \in \mathbb{R}^{n \times k}$ and $B \in \mathbb{R}^{m \times k}$, where n is number of users, m is number of items and $k = 5$, whose values are uniformly distributed at random in $[0, 1]$. We then computed the singular value decomposition of these matrices to obtain $A = U_A \Sigma_A V_A^T$ and $B = U_B \Sigma_B V_B^T$. We then let $P = \alpha U_A$, $Q = \alpha U_B$ and $R = PQ^T$. Thus, the final rank k matrix R is obtained as the product of two randomly generated rank k matrices whose columns are orthogonal. Note that the parameter α was determined empirically in order to produce ratings in the range of $[-10, 10]$.

Since we know the complete synthetic low-rank matrix we can generate the rating corresponding to an observed user-item pair in the real dataset from the complete rating matrix. We randomly selected 1000 users without replacement from the dataset and for each user we created sets containing five movies. The movies in a user's set were selected at random without replacement from the movies rated by that user. For each user, we created at least k such sets of movies, where $k \in [40, 60, 80, 100, 140]$. We generated rating for a user on a set by following two approaches:

(i) ESARM-based rating: For each user, we chose one of the extremal subsets at random and used that to generate ratings for all the sets. The set is assigned an average of the user's ratings on the items in the chosen extremal subset of the items in the set.

(ii) VOARM-based rating: For each user, we chose the user's level of pickiness (the β_u parameter) at random from the range [-2.0, 2.0]. The set is assigned an average of the user's ratings on the items in the set, and also we offset this rating by adding a quantity computed by scaling the standard deviation of ratings in the set by the randomly chosen user's level of pickiness.

For all these datasets, we added random $\mathcal{N}(0, 0.1)$ Gaussian noise while computing ratings at both the item and set-level for the users. For each approach, we generated 15 different synthetic datasets, each by varying the user-item latent factors and the users' pickiness.

7.4.2 Evaluation methodology

To evaluate the performance of the proposed methods we divided the available set-level ratings for each user into training, validation and test splits by randomly selecting five set-level ratings for each of the validation and test splits. The validation split was used for model selection. In order to assess the performance of the methods for item recommendations, we used a test set that contained for each user the items that were not present in the user's sets (i.e., these were absent from the training, test, and validation splits) but were present in the original user-item rating matrix used to generate the sets. We used Root Mean Square Error (RMSE) to measure the accuracy of the rating prediction over items and sets.

7.4.2.1 Comparison methods

In addition to the evaluation of the proposed methods, i.e., ARM, ESARM and VOARM, we also present the results for the following non-personalized methods:

(i) SetAvg: This method predicts a user's ratings on items and sets as the average of the user's ratings on sets. The rating of user u on set $\mathcal{S}$ is given by

$$\hat{r}_u^s = \frac{1}{|\mathcal{Q}_u|} \sum_{k \in \mathcal{Q}_u} \hat{r}_{uk}, \tag{7.8}$$

where $\mathcal{Q}_u$ represents all the sets rated by user u. The rating of user u on item i is given by

$$\hat{r}_{ui} = \frac{1}{|\mathcal{Q}_u|} \sum_{k \in \mathcal{Q}_u} \hat{r}_{uk}, \tag{7.9}$$

where $\mathcal{Q}_u$ represents all the sets rated by user u.

(ii) Item average: This method estimates the rating for an item as the average of the ratings provided by the users on the item. The rating $\hat{r}_i$ for an item i is given by

$$\hat{r}_i = \frac{1}{|\mathcal{U}_i|} \sum_{u \in \mathcal{U}_i} r_{ui}, \tag{7.10}$$

where $\mathcal{U}_i$ denotes the set of users who have rated item i.

(iii) UserMeanSub: This method estimates the rating for an item as the sum of average rating on sets and average of user mean subtracted item ratings. The rating $\hat{r}_i$ for an item i is given by

$$\hat{r}_i = \mu_s + \frac{1}{|\mathcal{U}_i|} \sum_{u \in \mathcal{U}_i} \big(r_{ui} - \frac{1}{|\mathcal{I}_u|} \sum_{k \in \mathcal{I}_u} r_{uk}\big), \tag{7.11}$$

where μ_s is the average of the ratings on all the sets, $\mathcal{I}_u$ represents the set of items rated by user u.

In practice, a significant proportion of the ratings provided by users on items depends on factors that are associated with either users or items, and do not depend on interactions between the users and the items. For example, some users have a tendency to rate higher than others, and some items receive higher ratings than others. For the real set-level rating dataset, that we obtained from Movielens users, we determined these factors by estimating user- and item-biases [72] as part of the model learning.

7.4.3 Model selection

We performed grid search to tune the dimensions of the latent factors and regularization hyper-parameters for the latent factors. We searched for regularization weights (λ) in the range [0.001, 0.01, 0.1, 1, 10], ϵ in the range [0.1, 0.25, 0.5, 1] and c in the range [0, 0.25, 0.50, 0.75, 0.90] for both the synthetic and the real datasets. We searched for the dimension of latent factors (f) in the range [1, 5, 10, 15, 25, 50, 75, 100] for real datasets, and used 5 as the dimension of latent factors for synthetic datasets. The final parameters were selected based on the performance on the validation split.

7.5 Results and Discussion

The experimental evaluation of the various methods that we developed is done in three phases. First, we investigated how well the proposed models can explain the users' ratings over sets in the dataset we obtained from a subset of Movielens users (described in Section 7.2). Second, we evaluated the performance of the methods using the synthetically generated datasets in order to assess how well the underlying optimization algorithms can recover the underlying data generation models and achieve good prediction performance at either the set- or item-level. Note that unless otherwise specified, we report the average of RMSEs of all the synthetic datasets as the final RMSE values for each rank and proposed approach. Finally, we evaluated the prediction performance achieved by the proposed methods at both the set- or item-level in the real dataset.

7.5.1 Fit of different rating models

In order to determine how well the proposed models can explain the ratings that the users in our dataset provided, we performed the following analysis. We selected sets with standard deviation of at least 0.5, and included only those users who have rated at least 20 such sets. This left us with 17,552 sets rated by 493 users.

To study the ESARM model, for each set rated by a user we created all the possible subsets having either k lowest or k highest rated items for all the possible values of $k \in [1,\ 5]$, i.e., nine extremal subsets. We computed the error between the average rating of items in the extremal subsets and the rating provided by a user on a set. Similarly, we computed the error over the remaining sets for a user and selected that subset among the nine extremal subsets corresponding to which the user has lowest Root Mean Square Error (RMSE) for all the sets. Figure 7.7 shows the number of users and their corresponding extremal subset that obtained lowest RMSE for their sets. As can be seen in the figure, there are certain users for whom the lowest RMSE on sets corresponds to either k lowest or k highest rated items in a set, where $k < 5$. This indicates that while providing a rating to a set of items, the user may get influenced more by a subset of the items in a set rather than all the items in the set.

Further, to investigate VOARM model, we computed the user's level of pickiness (β_u) as

$$\beta_u = \frac{1}{n_s} \sum_{s=1}^{n_s} \frac{r_u^s - \mu_s}{\sigma_s}, \tag{7.12}$$

where n_s is the number of sets rated by user u, r_u^s denotes the rating provided by user u on set s, μ_s is the mean rating of the items in set s and σ_s is the standard deviation of the ratings of the items in set s. Figure 7.8 shows the histogram of the users' level of pickiness. As can be seen from the figure, certain users tend to under- or over-rate

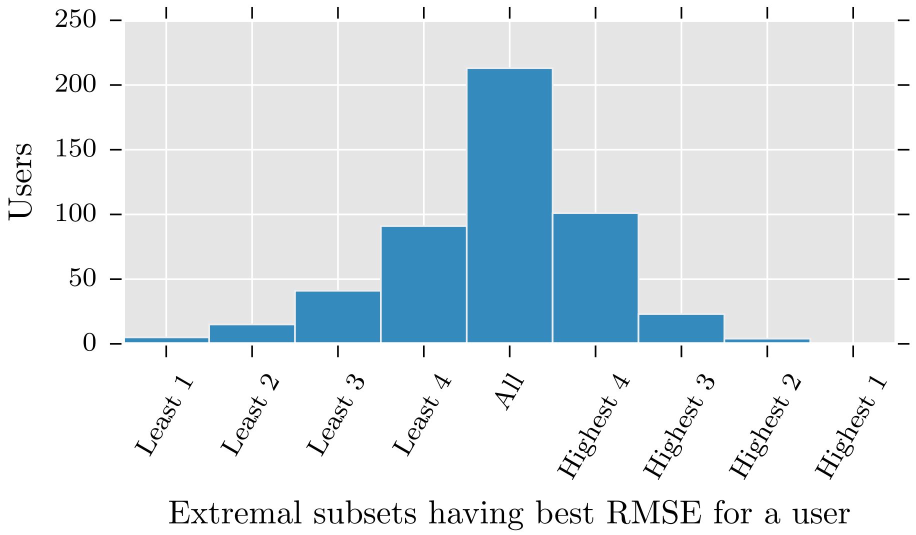


Figure 7.7: The number of users for which their pickiness behavior is explained by the corresponding least- and highest-rated subsets of items.

Table 7.1: Fit of different rating models on the data

	ARM	ESARM	VOARM
RMSE	0.597	0.509	0.521

sets with high standard deviation, and interestingly more users tend to under-rate sets than over-rate them.

Additionally, we computed how well the above rating models, i.e., ESARM and VOARM, compare against the ARM model where a user rates a set as the average of the ratings that he/she gives to the set's items. We used the user-specific pickiness determined in above analysis for the ESARM and the VOARM models to estimate a user's rating on a set. Table 7.1 shows the RMSE of the estimated ratings according to different models and as can be seen in the table both the ESARM and the VOARM give a better fit to the real data than ARM, thereby suggesting that modeling users' level of pickiness could lead to better estimates.

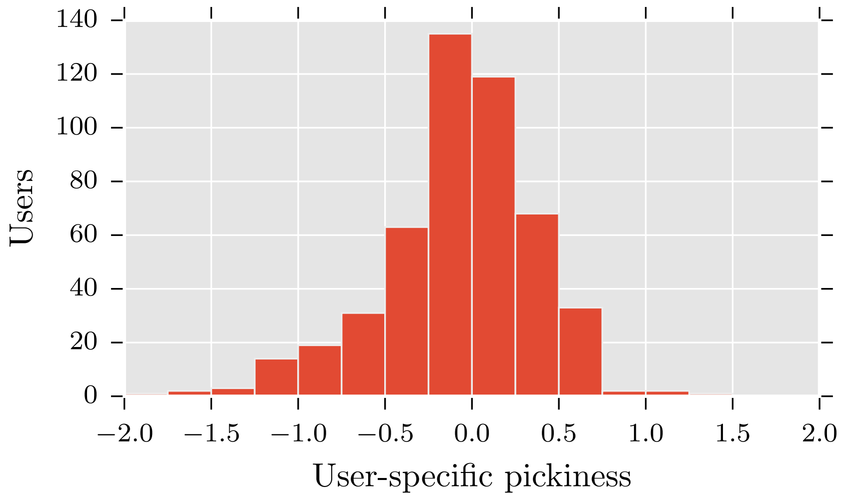


Figure 7.8: The number of users and their computed level of pickiness.

7.5.2 Performance on the synthetic datasets

7.5.2.1 Accuracy of set- and item-level predictions

We investigated the performance of the proposed methods for both item- and set-level predictions on the synthetic datasets. In addition to the performance of each method on its corresponding dataset, we also show the performance of the ARM and SetAvg methods in Figures 7.9 and 7.10.

Figure 7.9 shows that ESARM outperforms all other methods for both set- and item-level predictions for datasets with a large number of sets. However, for datasets with fewer sets, ARM outperforms ESARM and SetAvg for the set- and item-level predictions. Figure 7.10 shows that VOARM outperforms all other methods for both set- and item-level predictions. Unlike ESARM, VOARM performs better than other methods even for the case when we have fewer sets, and this suggests that ESARM needs a larger number of sets than VOARM to recover the underlying characteristics of the data.

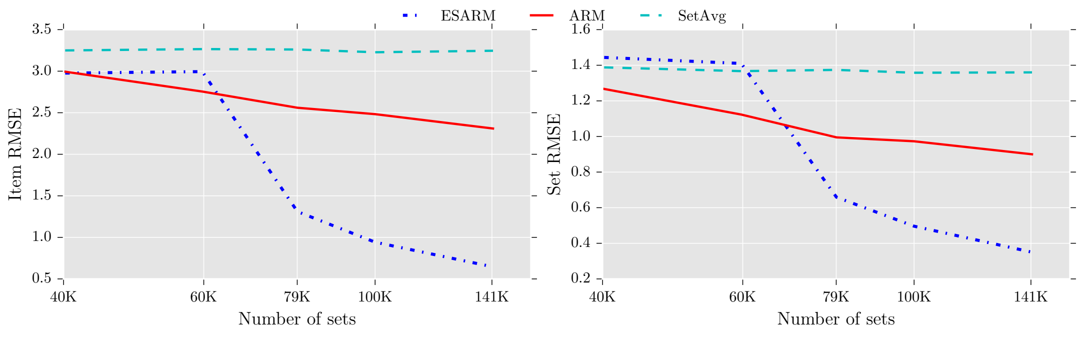


Figure 7.9: The average RMSE obtained by the proposed methods on ESARM-based datasets with different number of sets.

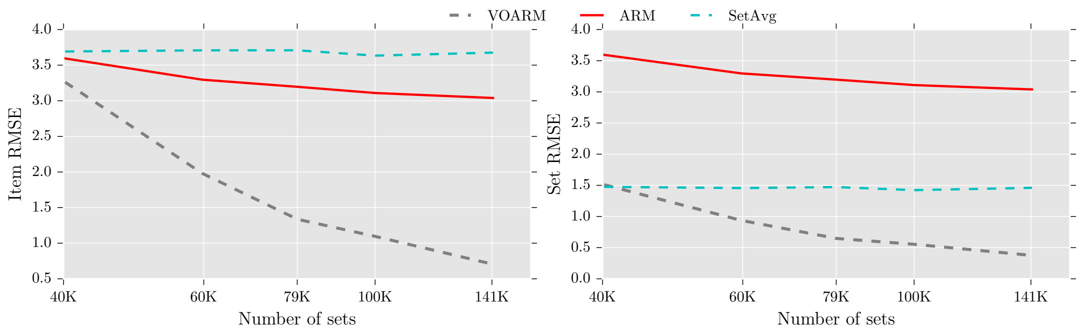


Figure 7.10: The average RMSE obtained by the proposed methods on VOARM-based datasets with different number of sets.

7.5.2.2 Recovery of underlying characteristics

We examined how well ESARM and VOARM recover the known underlying characteristics of the users in the datasets. Figure 7.11 plots the Pearson correlation coefficient of the actual and the estimated weights that model the users' level of pickiness in VOARM (i.e., β_u parameters). The high values of Pearson correlation coefficients in the figure suggests that VOARM is able to recover the overall characteristics of the underlying data. Additionally, this recovery of underlying characteristics increases with the increase in the number of sets. In order to investigate how well ESARM can recover the underlying characteristics, we computed the fraction of users for whom the extremal subset having the highest weight (w_{ui}) is same as that of the extremal subset used to generate the rating on sets. Figure 7.12 shows the percentage of users for whom the

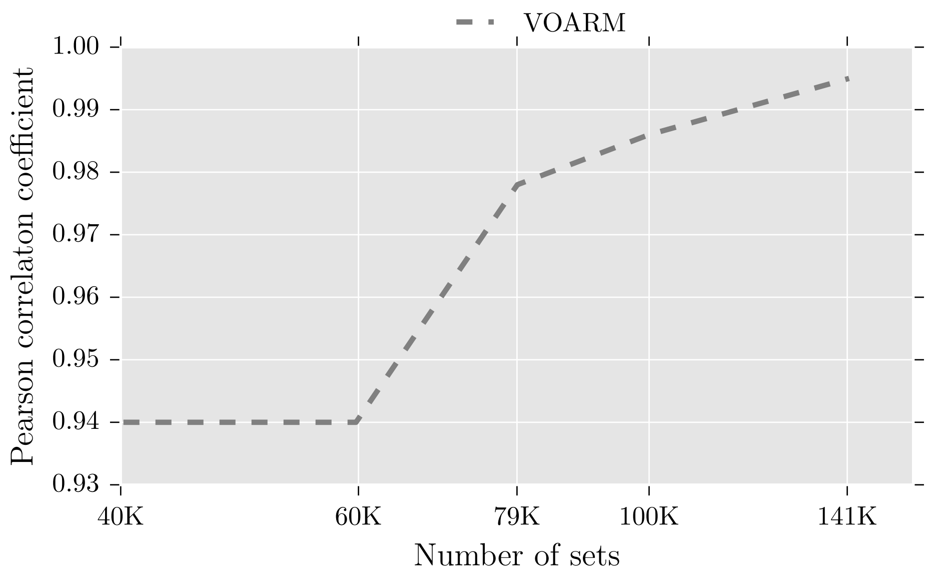


Figure 7.11: Pearson correlation coefficients of the actual and the estimated parameters that model a user's level of pickiness in the VOARM model.

extremal subsets are recovered by ESARM. As can be seen in the figure, the fraction of users recovered by ESARM increases significantly with the increase in the number of sets. The better performance of ESARM on the larger datasets suggests that in order to recover the underlying characteristics of the data accurately, ESARM needs significantly more data than required by VOARM method.

7.5.2.3 Effect of adding item-level ratings

In most real-world scenarios, in addition to set-level ratings, we will also have available ratings on individual items as well, e.g., users may provide ratings on music albums and as well as on tracks in the albums. Also, there may exist some users that are not concerned about keeping their item-level ratings private. To assess how well ESARM and VOARM can take advantage of such item-level ratings we performed three sets of experiments. In the first experiment, we added in the synthetic datasets a set of item-level ratings for the same set of users for which we have approximately $100K$ set-level ratings. The number of item-level ratings was kept to $k\%$ of their set-level ratings, where $k \in [5, 75]$, and the items that were added were disjoint from those that were part of the

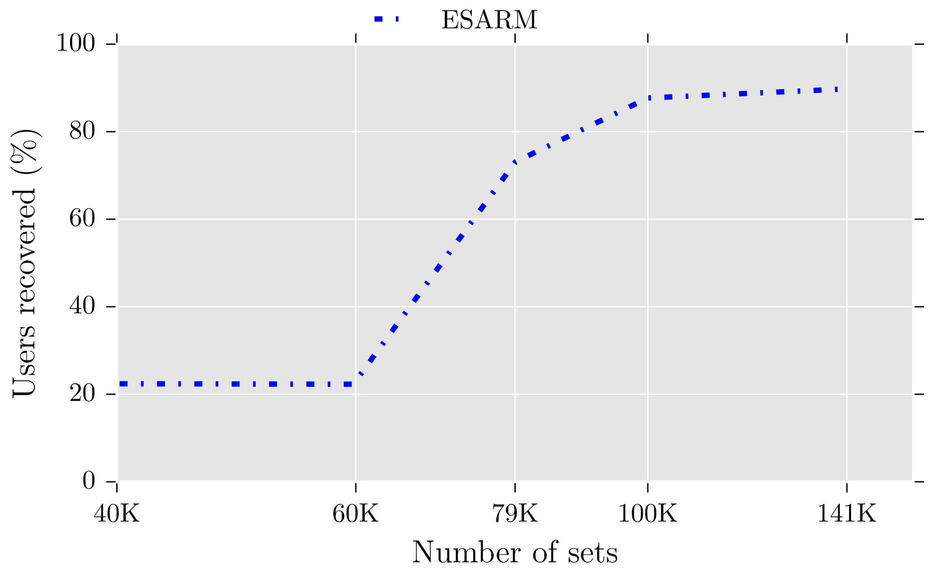


Figure 7.12: The percentage of users recovered by ESARM, i.e., the users for whom the original extremal subset had the highest estimated weight under these models.

sets that they rated. Additionally, we used the matrix factorization (MF) method to estimate the user and item latent factors by using only the added item-level ratings. In the second experiment, we selected 100, 250 and 500 additional users (beyond those that exist in the synthetically generated datasets) and added a random subset of 50 ratings per user from the items that belong to the existing users' sets. In the final experiment, we investigate if using set-level ratings from one set of users can improve the item-level predictions for another set of users for whom we have item-level ratings. We selected 500 additional users (U_b) and added a random subset of 50 ratings per user from the items that belong to the sets rated by existing users (U_a).

Figures 7.13 and 7.14 shows the performance of ESARM and VOARM on these datasets. As can be seen from Figure 7.13, as we continue to add item-level ratings for the same set of users who have provided ratings for the sets, there is an increase in accuracy of both the set- and item-level predictions for ESARM and VOARM. Both ESARM and VOARM outperform ARM with the availability of more item-level ratings. For the task of item-level rating prediction, ESARM and VOARM even outperform MF which is estimated only based on the additional item-level ratings. Figure 7.14 shows

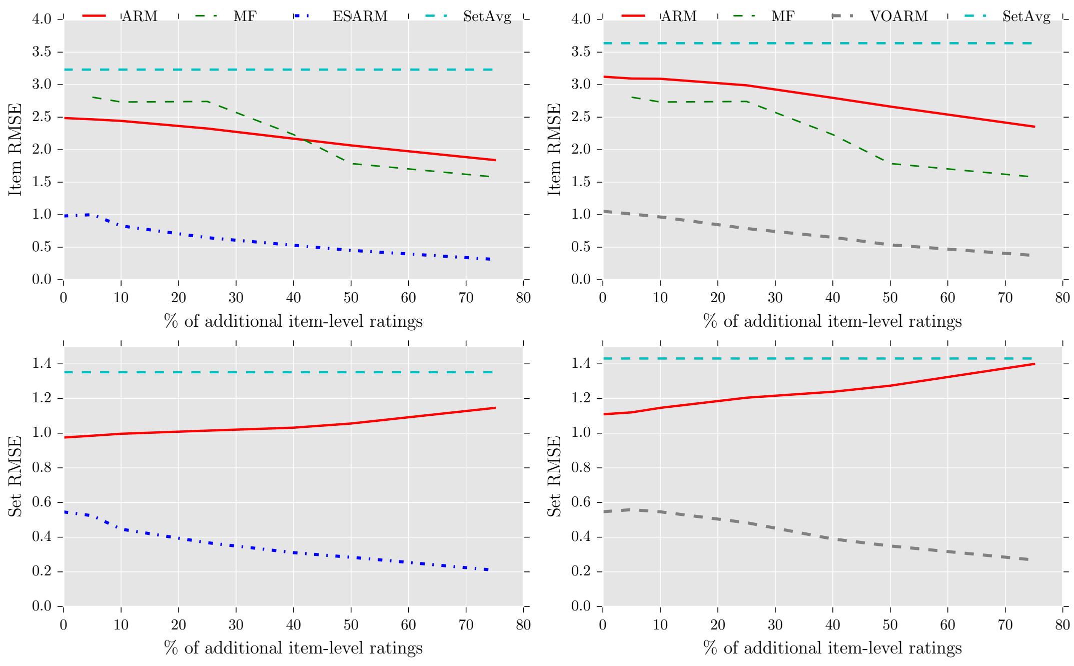


Figure 7.13: Effect of adding disjoint item-level ratings for the users in ESARM-based (left) and VOARM-based (right) datasets.

how the performance of the proposed methods changes when item-level ratings are available from another set of users. Similar to the addition of item-level ratings from the same set of users, ESARM and VOARM outperform ARM with the availability of item-level ratings from a different set of users.

Table 7.2 shows the performance of item-level predictions for additional users (U_b) after using set-level ratings from existing users (U_a). As can be seen from the table, using set-level ratings from users in U_a significantly improves the performance of item-level predictions for users in U_b. That is, using item-level ratings from the additional set of users and the set-level ratings from the existing users not only improves the performance for the latter but also for those additional users who have provided item-level ratings. The result that the performance of the proposed methods improve with the addition of item-level ratings suggests that using both item- and set-level ratings can lead to better item recommendations for the users.

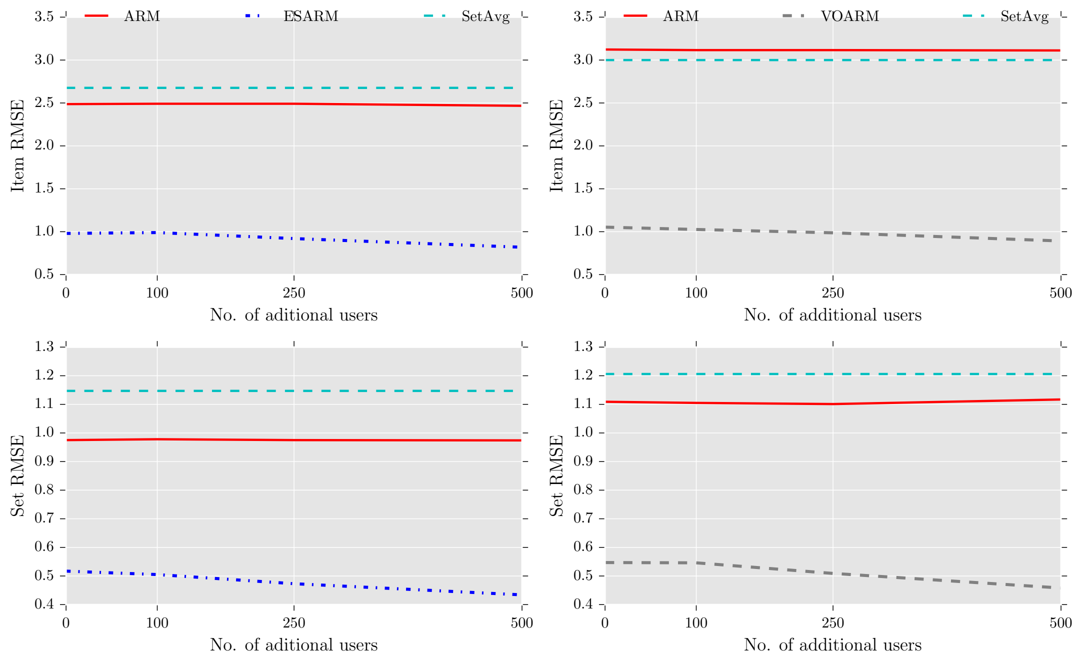


Figure 7.14: Effect of adding item-level ratings from additional users in ESARM-based (left) and VOARM-based (right) datasets.

7.5.3 Performance on the Movielens-based real dataset

Our final experiment used the proposed approaches (ARM, ESARM, and VOARM) to estimate both set- and item-level rating prediction models using the real set-level rating dataset that we obtained from Movielens users.

7.5.3.1 Accuracy of set- and item- level predictions

Table 7.3 shows results for the case when we have only set-level ratings. As can be seen in the table, ARM outperforms the remaining methods for item-level predictions. However, VOARM performs somewhat better than ARM for set-level predictions. The better performance of ARM for item-level predictions is not surprising as most of the sets in the dataset are rated close to the average of the ratings on items in sets. Also, as seen in our analysis in Section 7.5.2.2, ESARM needs a large number of sets in order to accurately recover the users' extremal subsets. The difference between the predictions

Table 7.2: Average RMSE performance of ESARM and VOARM for item-level predictions for additional users (U_b), that have provided only the item-level ratings.

	Item-level RMSE for U_b	
Type of ratings	ESARM	VOARM
Item-level (U_b)	2.860	2.860
Set-level (U_a) + item-level (U_b)	<u>1.811</u>	<u>1.866</u>

U_a represents the existing users that have provided ratings at the set-level. U_b represents the additional 500 users that have provided ratings only at item-level and item-level (U_b) denotes their item-level ratings. Set-level (U_a) refers to the set-level ratings from the users in U_a.

Table 7.3: The RMSE performance of the proposed methods with user- and item-biases on ML-RealSets dataset.

Method	Item	Set
SetAvg	0.976	0.630
ARM	<u>0.971</u>	0.624
ESARM	0.979	0.631
VOARM	0.973	<u>0.623</u>

of different models was found to be statistically significant (p-value ≤ 0.016 using t-test). Table 7.4 shows the percentage of the item-level predictions for whom a proposed approach performs better than the other approaches. As can be seen in the table, ARM and VOARM performs better than other methods for the majority of the item-level predictions. Also, to some extend VOARM performs better than ARM for the majority of the item-level predictions. The lower RMSE of ARM for item-level predictions and better performance of VOARM for the majority of the item-level predictions suggest that there are few item-level predictions where the error in VOARM is significantly higher than that of ARM, thereby leading to higher RMSE for item-level predictions for VOARM method. In Section 7.5.3.3, we will investigate the performance of the proposed methods independently for picky and non-picky users.

Table 7.4: Percentage of item-level predictions where method X performs better than method Y.

Method X \ Method Y	SetAvg	ARM	ESARM	VOARM
SetAvg	-	49.56	53.74	46.41
ARM	50.44	-	51.01	49.85
ESARM	46.26	48.99	-	45.54
VOARM	53.59	50.15	54.46	-

7.5.3.2 Effect of adding item-level ratings

In addition, we assessed how well the proposed methods can take advantage of additional item-level ratings. In the first experiment, we added $k\%$ of the users' set-level ratings, where $k \in [10, 75]$, as additional item-level ratings and the items that were added were disjoint from those that were part of the sets that they rated. In the second experiment, we added ratings from 100, 250 and 500 additional users (beyond those that have participated in the survey), and these users have provided on an average 20,000 ratings for the items that belong to the existing users' sets.

As can be seen from Figure 7.15, the performance for item-level predictions improves significantly after including item-level ratings. ARM and to some extend ESARM and VOARM even outperform MF for item-level predictions when fewer additional item-level ratings, i.e., $< 30\%$ of set-level ratings, are available. Figure 7.16 plots the estimated weights that model a user's level of pickiness in VOARM against the user's level of pickiness, i.e., β_u, computed from the data in Section 7.5.1. As can be seen in the figure, to some extend VOARM is able to recover the user' level of pickiness after addition of few item-level ratings.

Additionally, we examined the case when we have item-level ratings from the additional users. In addition to estimating ratings from the proposed methods, we estimated the ratings at item-level from the two non-personalized methods, i.e., Item average and UserMeanSub, as described in Section 7.4.2.1. Figure 7.17 shows the results for these non-personalized methods along with that of the proposed methods. As can be seen in

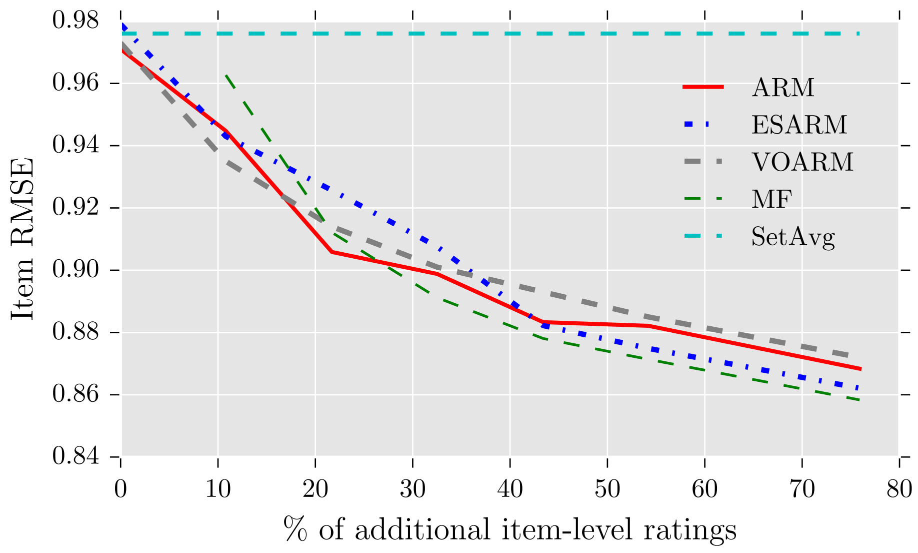


Figure 7.15: Effect of adding item-level ratings from the same set of users in the real dataset.

the figure, the proposed methods outperform the non-personalized methods and performance of the proposed methods continue to improve with the availability of more item-level ratings from additional users.

Further, we investigated if using set-level ratings from existing users can improve the item-level predictions for additional users who have provided ratings only at item-level. To this end, we selected 500 additional users and added a random subset of 10 ratings per user from the items that belong to the sets rated by existing users. Table 7.5

Table 7.5: RMSE for item-level predictions for additional users, that have provided only the item-level ratings.

Method	Item-level RMSE
MF	1.003
ARM	<u>0.978</u>
ESARM	1.043
VOARM	1.033

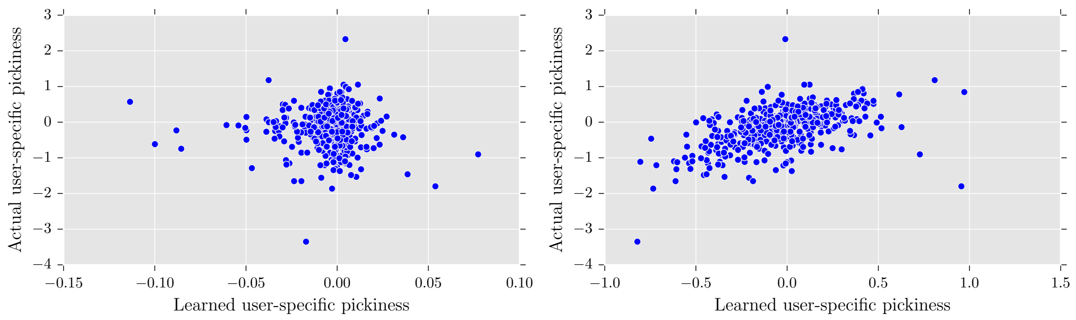


Figure 7.16: Scatter plots of the user's original level of pickiness computed from real data and the pickiness estimated by VOARM from set-level ratings (left), and after including 30% of item-level ratings (right).

shows the performance of item-level predictions for additional users after using set-level ratings from the existing users and also shows the performance of MF method after using only the additional item-level ratings. As can be seen in the table, ARM outperforms MF for item-level predictions after using set-level ratings from existing users. However, ESARM and VOARM do not perform better than MF for the additional users. Similar to our results on synthetic datasets, it is promising that using item-level ratings from the additional users and set-level ratings from the existing users improves the performance not only for latter but also for those additional users who have provided only item-level ratings.

7.5.3.3 Accuracy of item-level predictions for picky users

Even though ARM performs better than remaining methods for item-level predictions, we investigated how well do ARM, ESARM and VOARM perform for item-level predictions for the users who have rated at least 20 sets and have a high level of pickiness, i.e., $|\beta_u| > 0.5$. We found 374 users in the dataset that were non-picky ($U_{Non-picky}$) and 135 users that were having a higher level of pickiness (U_{Picky}). Table 7.6 shows the performance of the proposed methods for item-level predictions using set-level ratings and after including 30% of additional item-level ratings on both U_{Picky} and $U_{Non-picky}$. As can be seen in the table, for set-level ratings VOARM performs somewhat better than ARM on U_{picky} and after including additional item-level ratings both ESARM and

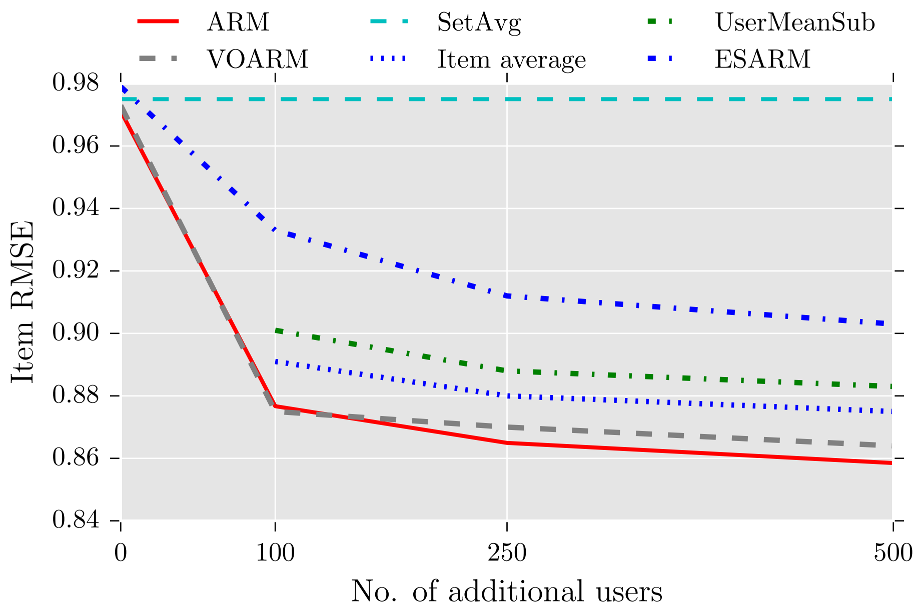


Figure 7.17: Effect of adding item-level ratings from disjoint set of users in the real dataset.

VOARM outperform ARM on U_{Picky}.

The overall consistency of the results between the synthetically generated and the real dataset suggests that VOARM and to some extend ESARM are able to capture the tendency that some users have to consistently under- or over-rate diverse sets of items.

7.6 Conclusion

In this work, we studied how users' ratings on sets of items relate to their ratings on the sets' individual items. We collected ratings from active users of Movielens on sets of movies and based on our analysis we developed collaborative filtering-based models that try to explicitly model the users' behavior in providing the ratings on sets of items. Through extensive experiments on synthetic and real data, we showed that the proposed methods can model the users' behavior as seen in the real data and predict the users' ratings on individual items.

Table 7.6: The item-level RMSE of the proposed methods on different subset of users using only set-level ratings and after including additional item-level ratings.

	Set only		+Items	
Method	$U_{Non-picky}$	U_{Picky}	$U_{Non-picky}$	U_{Picky}
ARM	$\underline{0.915}$	1.089	$\underline{0.879}$	0.975
ESARM	0.922	1.103	0.898	$\underline{0.923}$
VOARM	0.921	$\underline{1.085}$	0.892	0.932

The "Set only" column denotes the results of the models that were estimated using only set-level ratings. The "+Items" column show the results of the models that were estimated using the sets of "Set only" and also some additional ratings on a different set of items from the same users that provided the set-level ratings. U_{picky} refers to the users who have rated at least 20 sets and have a high level of pickiness, i.e., $|\beta_u| > 0.5$, in real dataset, and $U_{Non-picky}$ represents the remaining users.

Chapter 8

Conclusion

8.1 Thesis Summary

Recommender systems are widely used to recommend relevant products to the users. They help a user by identifying few relevant products from a catalog containing a large number of products and thus help the user by filtering information for the user. Recommendations are typically generated by using either content-based or collaborative filtering-based methods. Content-based methods rely on attributes of users or items to generate recommendations, and collaborative filtering-based methods rely on explicit or implicit preferences provided by the users over items. Furthermore, collaborative filtering-based methods are divided into two classes, i.e., neighborhood-based and matrix completion-based methods. Neighborhood-based methods identify the user and item neighborhoods based on co-rating data to generate recommendations. Matrix completion-based methods learn low-rank models, i.e., the user and the item latent factors, from the data to generate recommendations.

In this thesis, we have investigated how the accuracy and the ranking performance of the matrix completion-based approaches are affected by the skewed distribution of ratings in the user-item rating matrices. Furthermore, we have model user preferences in scenarios where standard recommendation methods can not be applied. We have developed methods to recommend new items, i.e., cold-start item recommendations, and to leverage user preferences over sets of items to generate item recommendations.

Accuracy of matrix completion methods

Matrix completion is the state-of-the-art collaborative filtering method and is widely used to generate recommendations. In this thesis, we investigated the effect of the skewed distribution of ratings, as found in real datasets, on the accuracy and the ranking performance of matrix completion. We showed that the skewed distribution affects the accuracy of matrix completion, and the item with high frequency are predicted more accurately than the others. Additionally, we found that the items predicted at the top by matrix completion miss a significant number of true high-rated items. Furthermore, the ranking based on the predicted ratings is not severely affected by false positives as the items that are predicted at the top for a user but are absent from the true high rated items are present close to the true high rated items by the user. Also, we saw that the infrequent items are predicted low by the matrix completion-based methods thereby appearing later in the ranking of the items for recommendations.

Truncated matrix factorization (TruncatedMF)

In practice, few item attributes determine a significant portion of the user rating and the leftover portion of the rating is determined by other attributes. The item attributes and the user weights over these attributes are known as the item latent factors and the user latent factors respectively. In this thesis, we showed that in real datasets, some users or items may not have sufficient ratings to estimate all the user and the item latent factors accurately. We developed TruncatedMF, a matrix completion-based method which considers the number of ratings that a user and an item has to estimate the user's rating on the item. The exhaustive experiments on real datasets illustrate that the TruncatedMF method outperforms the state-of-the-art MF method for the task of rating prediction for the users and the items with few ratings.

User-specific feature-based factorized bilinear similarity model

Since we do not have any prior preferences for new items, the standard collaborative filtering methods can not be used to recommend the new or *cold-start* items. The non-collaborative methods that rely on similarities between the new item and the items preferred by a user in the past can be used for cold-start item recommendations. However, these non-collaborative methods ignore the interaction among features and consider them independently while computing similarities. In this thesis, we presented User-specific Feature-based factorized Bilinear Similarity Model (UFBSM) for cold-start item recommendations. UFBSM captures the interaction among the item features and leverages the information available from all the users to recommend new items. The extensive experiments on real dataset show that UFBSM can perform better than other methods in terms of recommendation quality, especially in datasets that have relatively small number of features and a considerable number of ratings for existing items.

Learning from sets of items in recommender systems

An additional source of information in recommender systems can be the ratings provided by the users on sets of items. For example, the users can provide ratings on music albums, song playlists, and reading lists. A preference provided by a user on a set of items indicates some information about the user's preference for individual items in the set. Additionally, due to privacy concerns, users may not be willing to indicate their preferences for individual items but may provide a rating to a set of items as it provides some level of information hiding. In this thesis, we have investigated how a user's rating on a set of items relates to individual item-level ratings and developed collaborative filtering methods that can use the set-level ratings to generate item recommendations. The experiments on the real and the synthetic datasets show that the developed methods can recover the characteristics of underlying data and can be used for item recommendations.

8.2 Future research directions

The problems explored and methods presented in this thesis can be further extended in multiple future directions. It will be interesting to investigate the effect of different properties of ratings, e.g., diversity of ratings, in the user-item rating matrix on the performance of the matrix completion-based recommendation methods. We can also leverage the derived insight in Section 5.2, i.e., only fewer dimensions of latent factors are estimated accurately for users or items with few ratings, to modify existing locality-based matrix completion methods [17–19] by using lower ranks for the sparse part and higher ranks for the dense part of the user-item rating matrix. Similar to TruncatedMF, we may be able to improve other latent factor-based methods that may suffer from inaccuracy due to insufficient data, e.g., Factorization Machines [70, 77] and Word2Vec [78].

Furthermore, we can improve the usage of preferences over sets of items by modeling temporal effects and by using side-information like genres or other movie metadata. Also, it will be interesting to investigate if similar to the diversity of ratings in the set there exists other properties at the item- or set-level that can affect a user's ratings on sets of items. Moreover, a user may rate the set of items independent of what is his preference for an individual item and instead rate the set depending on how does he perceive the set as a whole. In this scenario, the items in a set can complement each other and thereby receive a more favorable rating from the user. On the contrary, it could be possible that items in a set compete with each other and thus receive a more critical rating on the set. Thus, modeling the synergy and the competition among items in a set can further improve the estimation of the user preferences over sets and items.

References

[1] David C. Anastasiu, Evangelia Christakopoulou, Shaden Smith, Mohit Sharma, and George Karypis. Big data and recommender systems. Technical Report 16-034, University of Minnesota, Department of Computer Science & Engineering, 2016.

[2] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. *Recommender Systems Handbook*. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition, 2010.

[3] Pasquale Lops, Marco De Gemmis, and Giovanni Semeraro. Content-based recommender systems: State of the art and trends. In *Recommender systems handbook*, pages 73–105. Springer, 2011.

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

[5] Jonathan L Herlocker, Joseph A Konstan, Al Borchers, and John Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 230–237. ACM, 1999.

[6] Upendra Shardanand and Pattie Maes. Social information filtering: algorithms for automating word of mouth. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 210–217. ACM Press/Addison-Wesley Publishing Co., 1995.

[7] Joseph A Konstan, Bradley N Miller, David Maltz, Jonathan L Herlocker, Lee R Gordon, and John Riedl. Grouplens: applying collaborative filtering to usenet news. *Communications of the ACM*, 40(3):77–87, 1997.

[8] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, New York, NY, USA, 2001. ACM.

[9] Mukund Deshpande and George Karypis. Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems (TOIS)*, 22(1):143–177, 2004.

[10] 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.

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

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

[13] Emmanuel J. Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Foundations of Computational Mathematics*, 9(6):717–772, 2009.

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

[15] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[16] Emmanuel J. Candès and Terence Tao. The power of convex relaxation: Near-optimal matrix completion. *IEEE Trans. Inf. Theor.*, 56(5):2053–2080, May 2010.

[17] Joonseok Lee, Seungyeon Kim, Guy Lebanon, and Yoram Singer. Local low-rank matrix approximation. In *International Conference on Machine Learning*, pages 82–90, 2013.

[18] Joonseok Lee, Samy Bengio, Seungyeon Kim, Guy Lebanon, and Yoram Singer. Local collaborative ranking. In *Proceedings of the 23rd international conference on World wide web*, pages 85–96. ACM, 2014.

[19] Chao Chen, Dongsheng Li, Yingying Zhao, Qin Lv, and Li Shang. Wemarec: Accurate and scalable recommendation through weighted and ensemble matrix approximation. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*, pages 303–312. ACM, 2015.

[20] Daniel Billsus and Michael J. Pazzani. A hybrid user model for news story classification. In *Proceedings of the seventh international conference on User modeling*, pages 99–108, 1999.

[21] Manuel de Buenaga Rodríguez, Manuel J. Maña López, Alberto Díaz Esteban, and Pablo Gervás Gómez-Navarro. A user model based on content analysis for the intelligent personalization of a news service. In *Proceedings of the 8th International Conference on User Modeling 2001*, UM '01, pages 216–218, London, UK, UK, 2001. Springer-Verlag.

[22] E. Banos, I. Katakis, N. Bassiliades, G. Tsoumakas, and I. Vlahavas. Personews: a personalized news reader enhanced by machine learning and semantic filtering. In *Proceedings of the 2006 Confederated international conference on On the Move to Meaningful Internet Systems: CoopIS, DOA, GADA, and ODBASE - Volume Part I*, ODBASE'06/OTM'06, pages 975–982, Berlin, Heidelberg, 2006. Springer-Verlag.

[23] Deepak Agarwal and Bee-Chung Chen. Regression-based latent factor models. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 19–28. ACM, 2009.

[24] Liang Zhang, Deepak Agarwal, and Bee-Chung Chen. Generalizing matrix factorization through flexible regression priors. In *Proceedings of the fifth ACM conference on Recommender systems*, RecSys '11, pages 13–20, New York, NY, USA, 2011. ACM.

[25] Zeno Gantner, Lucas Drumond, Christoph Freudenthaler, Steffen Rendle, and Lars Schmidt-Thieme. Learning attribute-to-feature mappings for cold-start recommendations. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 176–185. IEEE, 2010.

[26] Asmaa Elbadrawy and George Karypis. User-specific feature-based similarity models for top-n recommendation of new items. *ACM Trans. Intell. Syst. Technol.*, 6(3):33:1–33:20, April 2015.

[27] Wei Chu and Seung-Taek Park. Personalized recommendation on dynamic content using predictive bilinear models. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 691–700, New York, NY, USA, 2009. ACM.

[28] LedyardR Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966.

[29] Joshua B. Tenenbaum and William T. Freeman. Separating style and content with bilinear models. *Neural Comput.*, 12(6):1247–1283, June 2000.

[30] Wei Wu, Zhengdong Lu, and Hang Li. Learning bilinear model for matching queries and documents. *J. Mach. Learn. Res.*, 14(1):2519–2548, January 2013.

[31] Jimeng Sun, Dacheng Tao, and Christos Faloutsos. Beyond streams and graphs: Dynamic tensor analysis. In *In KDD*, pages 374–383, 2006.

[32] Seung-Taek Park and Wei Chu. Pairwise preference regression for cold-start recommendation. In *Proceedings of the Third ACM Conference on Recommender Systems*, RecSys '09, pages 21–28, New York, NY, USA, 2009. ACM.

[33] Shuo Chang, F. Maxwell Harper, and Loren Terveen. Using groups of items for preference elicitation in recommender systems. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work &#38; Social Computing*, CSCW '15, pages 1258–1269, New York, NY, USA, 2015. ACM.

[34] Shuo Chen, Josh L. Moore, Douglas Turnbull, and Thorsten Joachims. Playlist prediction via metric embedding. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 714–722, New York, NY, USA, 2012. ACM.

[35] Joshua L Moore, Shuo Chen, Thorsten Joachims, and Douglas Turnbull. Learning to embed songs and tags for playlist prediction. In *ISMIR*, pages 349–354, 2012.

[36] Natalie Aizenberg, Yehuda Koren, and Oren Somekh. Build your own music recommender by modeling internet radio streams. In *Proceedings of the 21st international conference on World Wide Web*, pages 1–10. ACM, 2012.

[37] Roberto Interdonato, Salvatore Romeo, Andrea Tagarelli, and George Karypis. A versatile graph-based approach to package recommendation. In *2013 IEEE 25th International Conference On Tools with Artificial Intelligence*, pages 857–864. IEEE, 2013.

[38] Q. Liu, E. Chen, H. Xiong, Y. Ge, Z. Li, and X. Wu. A cocktail approach for travel package recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 26(2):278–293, Feb 2014.

[39] Qi Liu, Yong Ge, Zhongmou Li, Enhong Chen, and Hui Xiong. Personalized travel package recommendation. In *2011 IEEE 11th International Conference on Data Mining*, pages 407–416. IEEE, 2011.

[40] Min Xie, Laks VS Lakshmanan, and Peter T Wood. Comprec-trip: A composite recommendation system for travel planning. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 1352–1355. IEEE, 2011.

[41] Yidan Liu, Min Xie, and Laks V.S. Lakshmanan. Recommending user generated item lists. In *Proceedings of the 8th ACM Conference on Recommender Systems*, RecSys '14, pages 185–192, New York, NY, USA, 2014. ACM.

[42] Min Xie, Laks VS Lakshmanan, and Peter T Wood. Breaking out of the box of recommendations: from items to packages. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 151–158. ACM, 2010.

[43] Idir Benouaret and Dominique Lenne. A package recommendation framework for trip planning activities. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pages 203–206, New York, NY, USA, 2016. ACM.

[44] George William Hart. Nonintrusive appliance load monitoring. *Proceedings of the IEEE*, 80(12):1870–1891, 1992.

[45] Jon Froehlich, Eric Larson, Sidhant Gupta, Gabe Cohn, Matthew Reynolds, and Shwetak Patel. Disaggregated end-use energy sensing for the smart grid. *IEEE Pervasive Computing*, 10(1):28–39, 2011.

[46] Sarah Darby et al. The effectiveness of feedback on energy consumption. *A Review for DEFRA of the Literature on Metering, Billing and direct Displays*, 486(2006), 2006.

[47] Dan Cosley, Shyong K Lam, Istvan Albert, Joseph A Konstan, and John Riedl. Is seeing believing?: how recommender system interfaces affect users' opinions. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 585–592. ACM, 2003.

[48] Tien T Nguyen, Daniel Kluver, Ting-Yu Wang, Pik-Mai Hui, Michael D Ekstrand, Martijn C Willemsen, and John Riedl. Rating support interfaces to improve user experience and recommender accuracy. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 149–156. ACM, 2013.

[49] Katja Hofmann, Anne Schuth, Alejandro Bellogin, and Maarten De Rijke. Effects of position bias on click-based recommender evaluation. In *ECIR*, volume 14, pages 624–630. Springer, 2014.

[50] Yisong Yue, Rajan Patel, and Hein Roehrig. Beyond position bias: Examining result attractiveness as a source of presentation bias in clickthrough data. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 1011–1018, New York, NY, USA, 2010. ACM.

[51] Alejandro Bellogín, Alan Said, and Arjen P de Vries. The magic barrier of recommender systems–no magic, just ratings. In *International Conference on User Modeling, Adaptation, and Personalization*, pages 25–36. Springer, 2014.

[52] Katja Hofmann, Fritz Behr, and Filip Radlinski. On caption bias in interleaving experiments. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, pages 115–124, New York, NY, USA, 2012. ACM.

[53] Gretchen B Chapman and Eric J Johnson. The limits of anchoring. *Journal of Behavioral Decision Making*, 7(4):223–242, 1994.

[54] Zimo Yang, Zi-Ke Zhang, and Tao Zhou. Anchoring bias in online voting. *EPL (Europhysics Letters)*, 100(6):68002, 2013.

[55] Amos Tversky and Daniel Kahneman. Advances in prospect theory: Cumulative representation of uncertainty. *Journal of Risk and uncertainty*, 5(4):297–323, 1992.

[56] Amos Tversky and Daniel Kahneman. Judgment under uncertainty: Heuristics and biases. *Science*, 185(4157):1124–1131, 1974.

[57] Gediminas Adomavicius, Jesse Bockstedt, Shawn Curley, and Jingjing Zhang. Recommender systems, consumer preferences, and anchoring effects. In *RecSys 2011 Workshop on Human Decision Making in Recommender Systems*, pages 35–42, 2011.

[58] Pinata Winoto and Tiffany Y. Tang. The role of user mood in movie recommendations. *Expert Syst. Appl.*, 37(8):6086–6092, August 2010.

[59] Andreas Herrmann, Frank Huber, and Robin Higie Coulter. Product and service bundling decisions and their effects on purchase intention. *Pricing Strategy and Practice*, 5(3):99–107, 1997.

[60] Dorothy Paun. When to bundle or unbundle products. *Industrial Marketing Management*, 22(1):29–34, 1993.

[61] Manjit S Yadav. How buyers evaluate product bundles: A model of anchoring and adjustment. *Journal of Consumer Research*, 21(2):342–353, 1994.

[62] Bari A Harlam, Aradhna Krishna, Donald R Lehmann, and Carl Mela. Impact of bundle type, price framing and familiarity on purchase intention for the bundle. *journal of Business Research*, 33(1):57–66, 1995.

[63] Gary J Gaeth, Irwin P Levin, Goutam Chakraborty, and Aron M Levin. Consumer evaluation of multi-product bundles: An information integration analysis. *Marketing letters*, 2(1):47–57, 1991.

[64] Prateek Jain, Praneeth Netrapalli, and Sujay Sanghavi. Low-rank matrix completion using alternating minimization. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 665–674, New York, NY, USA, 2013. ACM.

[65] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[66] Mohit Sharma, Jiayu Zhou, Junling Hu, and George Karypis. Feature-based factorized bilinear similarity model for cold-start top-n item recommendation. In *SIAM International Conference on Data Mining, 2015.*, SDM '15, 2015.

[67] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, pages 452–461. AUAI Press, 2009.

[68] Iván Cantador, Peter Brusilovsky, and Tsvi Kuflik. 2nd workshop on information heterogeneity and fusion in recommender systems (hetrec 2011). In *Proceedings of the 5th ACM conference on Recommender systems*, RecSys 2011, New York, NY, USA, 2011. ACM.

[69] Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. Improving recommendation lists through topic diversification. In *Proceedings of the 14th International Conference on World Wide Web*, WWW '05, pages 22–32, New York, NY, USA, 2005. ACM.

[70] Steffen Rendle. Factorization machines with libFM. *ACM Trans. Intell. Syst. Technol.*, 3(3):57:1–57:22, May 2012.

[71] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, New York, NY, USA, 2001. ACM.

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

[73] Dirk Bollen, Mark Graus, and Martijn C Willemsen. Remembering the stars?: effect of time on preference retrieval from memory. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 217–220. ACM, 2012.

[74] Léon Bottou. Online algorithms and stochastic approximations. In David Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998.

[75] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[76] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 5(4):19, 2016.

[77] Steffen Rendle. Factorization machines. In *2010 IEEE International Conference on Data Mining*, pages 995–1000. IEEE, 2010.

[78] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.