Machine Learning Methods for Recommender Systems

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

Santosh Kabbur

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

Dr. George Karypis, Advisor

February, 2015

Acknowledgements

First, I would like to thank my advisor Professor George Karypis for his invaluable support and guidance throughout my graduate study. He has been a great mentor and the main source of inspiration during my Ph.D. studies. Most of what I have learned in data mining and machine learning, I owe it to him. I am truly grateful for his encouragement and patience over the years.

My deepest gratitude goes to my parents, Krishna Kabbur and Vidyavati Kabbur, for their love, support and encouragement. I thank them for all their sacrifices and unconditional support. I couldn't have achieved any of it, without them. Special thanks to my brother Prasad Kabbur, for being a true inspiration and a guide from my childhood. My heartfelt thanks to my wife, Shami, for all the love and support.

I would like to express my gratitude to Professors Joseph Konstan, Gediminas Adomavicius, Rui Kuang and Jaideep Srivastava for serving on my thesis committee and providing valuable feedback, criticism and advice.

I was fortunate enough to be among the intelligent and highly motivated students of the Karypis lab: Asmaa El-Budrawy, Agoritsa Polyzou, Chris Kauffman, David Anastasiu, Dominique Lasalle, Evangelia Christakopoulou, Fan Yang, Fuzhen Zhuang, Jeremy Iverson, Kevin DeRonne, Mohit Sharma, Rezwan Ahmed, Sara Morsy, Shaden Smith, Xia Ning, Yevgeniy Podolyan and Zhonghua Jiang. I thank them all for their friendship, advice, help and support over the years. I have learned a lot from all of them and it was a great experience being able to interact with them.

I would also like to thank the co-operative staff at the Department of Computer Science, the Digital Technology Center, and the Minnesota Supercomputing Institute at the University of Minnesota for providing state of-the-art research facilities and for assisting in my research.

I am grateful to all my mentors and colleagues during my internships at nXn, Google, Paypal and Twitter. Their guidance has helped me immensely in my graduate studies. Special thanks to Eui-Hong Han, Huong Le, Wei-Shou Hu and Andrea Tagarelli for collaborating with me. Interactions with them helped me learn a lot.

Last, but not the least, I would like to thank all my wonderful friends and the rest of my family members, for their love, support and encouragement throughout my life.

Dedication

To my parents

Abstract

This thesis focuses on machine learning and data mining methods for problems in the area of recommender systems. The presented methods represent a set of computational techniques that produce recommendation of items which are interesting to the target users. These recommendations are made from a large collection of such items by learning preferences from their interactions with the users.

We have addressed the two primary tasks in recommender systems, that is top-N recommendation and rating prediction. We have developed, (i) an item-based method (FISM) for generating top-N recommendations that learns the item-item similarity matrix as the product of two low dimensional latent factor matrices. These matrices are learned using a structural equation modeling approach, wherein the value being estimated is not used for its own estimation. Since, the effectiveness of existing top-N recommendation methods decreases as the sparsity of the datasets increases, FISM is developed to alleviate the problem of data sparsity, (ii) a new user modeling approach (MPCF), that models the user's preference as a combination of global preference and local preferences components. Using this user modeling approach, we propose two different methods based on the manner in which the global preference and local preferences components interact. In the first approach, the global component models the user's strong preferences on a subset of item features, while the local preferences component models the tradeoffs the users are willing to take on the rest of the item features. In the second approach, the global preference component models the user's overall preferences on all the item features and the local preferences component models the different tradeoffs the users have on all the item features, thereby helping to fine tune the global preferences. An additional advantage of MPCF is that, the user's global preferences are estimated by taking into account all the observations, thus it can handle sparse data effectively, (iii) a new method called ClustMF which is designed to combine the benefits of the neighborhood models and the latent factor models. The benefits of latent factor models are utilized by modeling the users and items similar to the standard MF based methods and the benefit of neighborhood models are brought into the model, by introducing biases at the cluster level. That is, the biases for users are modeled at the item

cluster level and the biases for items are modeled at the user cluster level. For an item to be part of the top-N list of the user, along with the latent factors component producing a high score, the corresponding user cluster bias and item cluster bias must also be high. That is, to have a high user cluster bias, the item must be in the neighborhood of the items that the user has liked in the past and to have a high item cluster bias, the user must be in the neighborhood of the users who have liked the item.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis focuses on the development of new machine learning methods that arise primarily in the area of Recommender Systems. Recommender Systems are prevalent and are widely used in many applications. In particular, recommender systems have gained popularity via their usage in e-commerce applications to recommend items so as to help the users in identifying the items that best fit their personal tastes. Recommender Systems helps the users to evaluate the potentially overwhelming number of options (termed as *items*) that a commercial service has to offer. Thus recommender systems have emerged as a key enabling technology for e-commerce. They perform the role of virtual experts who are keenly aware of users' preferences and tastes, and correspondingly filter out vast amount of irrelevant data in order to identify and recommend the most relevant products. An example application is a video recommender system that helps users to select a movie to watch from a video catalogue. Popular online video streaming service like Netflix employs a video recommendation system to personalize the experience for each user. The recommendations are served based on the watch history of the user. Thus, the recommendations served for different users will be diverse.

Over the years, many algorithms have been developed for recommender systems. These algorithms make use of the user feedback (purchase, rating or review) to compute the recommendations. However, the state-of-the-art algorithms developed suffer from various issues like data sparsity, inability to effectively model users' diverse interests, scalability, etc. The focus in this thesis is three fold. First, the focus is in dealing with the issue of data sparsity in building an efficient recommender systems. New

methods are developed to address the lack of feedback data for the users. Second, the focus is on modeling users' multiple preferences. Users tend to have multiple and diverse preferences and the existing models are not sufficiently capable to capture these preferences. The problem of modeling users' diverse interests is addressed in this thesis and a set of effective methods are presented. Third, the focus is on utilizing the benefits of both neighborhood models and latent factors model, which capture different and complementary characterisitics of the data. This is addressed by developing a new unified method which combines the benefits from both the models.

1.1 Key Contributions

There are two main problems associated with recommender systems. First, rating prediction problem, which aims to predict the rating for a user-item pair. Second, top-N recommendation problem, which aims to identify a short list of items that most fit a user's personal preferences. In the recent years, top-N recommendation has gained popularity due to the increase of e-commerce services that serve various user needs. Rating prediction is still relevant in many application areas where the predicted recommendation score is used. Thus, development of efficient top-N and rating prediction recommendation methods to generate high quality recommendations is highly desired.

1.1.1 Factored Item Similarity Methods (FISM)

In a real world scenario, the users provide feedback or purchase only a handful of items, from the possible set of thousands or millions of items. Thus the available user-item relational data is sparse. One class of existing state-of-the-art top-N methods provide recommendations by learning relations between items. However, they rely on co-purchase/co-rating information i.e., for the methods to learn meaningful relations and provide high quality recommendations, the training data must have enough users who have co-purchased many items. Thus, they suffer from data sparsity issue and fails to capture meaningful relations between users who do not have enough co-rated items. To effectively handle the real world sparse datasets, there is a need for methods which can effectively handle such sparse data.

In this thesis (Chapter 5), a new Factored Item Similarity Method (FISM) [1] for top-N recommendation is presented. There are multiple contributions from FISM methods. First, it extends the factored item-based methods to the top-N problem, which allow them to effectively handle sparse datasets. Second it estimates the factored item-based top-N models using a structural equation modeling approach, and third it estimates the factored item-based top-N models using both squared error and a ranking loss.

1.1.2 Modeling Global and Local Preferences of Users in Collaborative Filtering (MPCF)

Many existing state-of-the-art rating prediction and top-N collaborative filtering methods model user preferences as a vector in a latent space. These methods assume that the user preference is consistent across all the items that the user has rated and thus model the user using a single user preference vector. However, user preferences are typically much more complicated than that. Many users can have multiple tastes and their preferences can vary with each such taste. To address this, a recently proposed method extended the latent representation models to include multiple preference vectors for each user, in order to capture the user's preferences for each of the preferences separately. In this thesis (Chapter 6), we propose a different user modeling approach (MPCF)[2] that models the user's preference as a combination of global preference and local preferences components. Using this user modeling approach, we propose two different methods based on the manner in which the global preference and local preferences components interact. In the first approach, the global component models the user's strong preferences on a subset of item features, while the local preferences component models the tradeoffs the users are willing to take on the rest of the item features. In the second approach, the global preference component models the user's overall preferences on all the item features and the local preferences component models the different tradeoffs the users have on all the item features, thereby helping to fine tune the global preferences. An additional advantage of MPCF is that, the user's global preferences are estimated by taking into account all the observations, thus it can handle sparse data effectively. A comprehensive set of experiments on multiple datasets show that the proposed model outperforms other state-of-the-art recommendation methods for both rating prediction and top-N recommendation tasks.

1.1.3 Cluster Based Matrix Factorization Methods

Neighborhood models like UserKNN and ItemKNN are intuitive and simple to implement. They are most effective at detecting localized relationships and thus helps to better explain the recommendations provided to the user. The recommendations provided by neighborhood methods are backed by the observed data (in terms of co-rated users/items), and thus are somewhat "familiar" to the user, as they can be shown to be explicitly related to an item they have consumed in the past. However, the main limitation of neighborhood models is that the top recommended similar items are computed using only a small fraction of the user's preferences. Thus, they fail to capture the sum total of the weak signals provided by all of the user's preferences. On the other hand, latent factor models utilize all the user's preferences in learning the user and item latent factors to produce the recommendations. Thus, they are generally effective in capturing the overall relations that exist among the users and the items. Although individually, latent factor models have been shown to produce superior top-N recommendations compared to neighborhood models, the recommendations provided by latent factor models cannot be easily explained to the user; the notion of observed item neighborhood is not present and thus the user familiarity is absent. For top-N recommendation task, the absence of familiarity might affect the choice a user makes from the computed list of top-N recommended items, since there is a potential for the user to lose the context on why a particular item was recommended to him/her, in particular for items which lie "far away" in the neighborhood of the items rated by the user. Hence, there is a need for a combined model, which can capitalize on both the benefits of the neighborhood models and the latent factor models. That is, a model which can capture the localized relationships like neighborhood models to bring in the familiarity aspect to the users and also capture the global relations between users and items like the latent factor models to provide better recommendations. In this thesis (Chapter 7), we propose a new method called ClustMF which is designed to combine the benefits of both the neighborhood and latent factor models. The benefits of latent factors models are utilized by modeling the users and items similar to the standard MF based methods. That is, the users and items are represented with latent vectors, where the item latent vectors correspond to the latent features associated with them and the user latent vectors correspond to the user preferences on the item features. The benefit of neighborhood models are brought

into the model, by introducing biases at the cluster level. These cluster level biases are introduced to capture the localized relationships present in the user and/or item neighborhoods. For an item to be part of the top-N list of the user, along with the latent factors component producing a high score, the corresponding user cluster bias and item cluster bias must also be high. That is, to have a high user cluster bias, the item must be in the neighborhood of the items that the user has liked in the past and to have a high item cluster bias, the user must be in the neighborhood of the users who have liked the item. A comprehensive set of experiments show that the proposed model outperforms the rest of the state-of-the-art methods for top-N recommendation task.

1.2 Outline

This thesis is organized as follows:

- Chapter 2 provides definitions and 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 presents the evaluation methodology employed, the different datasets used and the different state-of-the art algorithms that we compare the performance for various methods presented in this thesis.
- Chapter 5 presents a new Factored Item Similarities Method (FISM) for top-N recommendation problem is presented.
- Chapter 6 presents a new set of Non Linear Factorization (MPCF) methods which better models the users' multiple interest preferences.
- Chapter 7 presents a new Combined Neighborhood and Latent Factors Models (ClustMF) for top-N recommender system is presented.
- Chapter 8 summarizes the conclusions of the research presented in this thesis and some future research directions.

1.3 Related Publications

The work presented in this thesis and the related work has been published in leading conferences and journals in data mining and knowledge discovery. The related publications are listed as follows:

- **Santosh Kabbur**, Eui-Hong Han and George Karypis. Content-based methods for predicting web-site demographic attributes. In *Proceedings of the 10th IEEE ICDM International Conference on Data Mining 2010*, pages 863–868, 2010.

- **Santosh Kabbur**, Huong Le, Luciano Pollastrini, Ziran Sun, Keri Mills, Kevin Johnson, George Karypis, Wei-Shou Hu. Multivariate analysis of cell culture bioprocess datalactate consumption as process indicator. In *Journal of biotechnology*, 2012.

- **Santosh Kabbur**, Xia Ning and George Karypis. FISM: factored item similarity models for top-N recommender systems. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 659–667, 2013.

- **Santosh Kabbur** and George Karypis. NLMF: Non-Linear Matrix Factorization Methods for Top-N Recommender Systems. In *Proceedings of IEEE Workshop on Domain Driven Data Mining*, ICDMW, 2014.

- **Santosh Kabbur** and George Karypis. MPCF: Modeling Global and Local Preferences of Users in Collaborative Filtering In *Proceedings of the 38th Annual ACM Special Interest Group On Information Retrieval*, SIGIR, 2015 (under review).

- **Santosh Kabbur** and George Karypis. ClustMF: Combined Neighborhood and Latent Factor Models for Top-N Recommender Systems *Ready for submission.*

Chapter 2

Definitions and Notations

All vectors are represented by bold lower case letters and they are row vectors (e.g., $\mathbf{p}, \mathbf{q}$). All matrices are represented by bold upper case letters (e.g., $\mathbf{R}$, $\mathbf{W}$). The ith row of a matrix $\mathbf{A}$ is represented by $\mathbf{a_i}$. We use calligraphic letters to denote sets (e.g., $\mathcal{C}$, $\mathcal{D}$). A predicted value is denoted by having a ˜(tilde) over it (e.g., $\tilde{r}$) and an estimated value is denoted by having a ˆ(hat) over it (e.g., $\hat{r}$).

$\mathcal{C}$ and $\mathcal{D}$ are used to denote the sets of users and items, respectively, whose respective cardinalities are n and m (i.e., $|\mathcal{C}| = n$ and $|\mathcal{D}| = m$). Matrix $\mathbf{R}$ will be used to represent the user-item feedback/rating matrix of size $n \times m$, i.e., $\mathbf{R} \in \mathbb{R}^{n \times m}$. Symbols u and i are used to denote individual users and items, respectively. An entry (u, i) in $\mathbf{R}$, denoted by r_{ui}, is used to represent the rating on item i by user u. For implicit feedback, $\mathbf{R}$ is converted to a binary matrix. If the user has provided feedback for a particular item, then the corresponding entry in $\mathbf{R}$ is 1, otherwise it is 0. We will refer to the entries for which the user has provided feedback as *rated* items and those for which the user has not provided feedback as *unrated* items.

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
$\mathcal{C}$	Set of users.
$\mathcal{D}$	Set of items.
u, i	Individual user u, item i.
n, m	Number of users and items, $n = \|\mathcal{C}\|, m = \|\mathcal{D}\|$.
k	Number of latent factors.
l	Number of latent factors for local preferences component in MPCFi.
T	Number of user local preferences.
l_u, l_i	Number of user and item clusters.
$\mathbf{ci}$	Item cluster assignment
$\mathbf{cu}$	User cluster assignment
$\mathbf{R}$	User-Item Feedback/Rating Matrix, $\mathbf{R} \in \mathbb{R}^{n \times m}$.
$\mathcal{R}_u^+$	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.
$\mathbf{b_u}$	User Bias vector, $\mathbf{b_u} \in \mathbb{R}^{1 \times n}$.
$\mathbf{b_i}$	Item Bias vector, $\mathbf{b_i} \in \mathbb{R}^{1 \times m}$.
$\mathbf{B_u}$	User-Interest Bias matrix, $\mathbf{B_u} \in \mathbb{R}^{n \times T}$.
$\mathbf{BU}$	User Cluster Bias matrix, $\mathbf{BU} \in \mathbb{R}^{n \times l_i}$.
$\mathbf{BI}$	Item Cluster Bias matrix, $\mathbf{B_u} \in \mathbb{R}^{m \times l_u}$.
$\mathbf{P}$	User Latent Factor Matrix, $\mathbf{P} \in \mathbb{R}^{n \times k}$.
$\mathbf{W}$	User Latent Factor Tensor, $\mathbf{W} \in \mathbb{R}^{n \times k \times T}$.
$\mathbf{Q}$	Item Latent Factor Matrix, $\mathbf{Q} \in \mathbb{R}^{m \times k}$.
$\mathbf{Y}$	Item Latent Factor Matrix, for local preferences component in MPCFi, $\mathbf{Y} \in \mathbb{R}^{m \times l}$.
$\mathbf{S}$	Item-Item Similarity matrix, $\mathbf{S} \in \mathbb{R}^{m \times m}$
λ	ℓ_F regularization weight.
ρ	Sampling factor for learning algorithm.
η	Learning Rate for learning algorithm.
n_u^+	Number of items for which user u has provided feedback
λ, γ	ℓ_F regularization weights
α	User Normalization constant

Chapter 3

Background and Related Work

Over the years, many algorithms and methods have been developed to address the rating prediction and top-N recommendation problem [3, 4, 5, 6, 7] in recommender systems. These algorithms make use of the user feedback data available in the form of purchase, rating or review. Typically these algorithms represent the feedback information as a user-purchase matrix and act on it. The existing methods can be broadly classified into two groups: collaborative filtering (CF) [8] based methods and content based methods. User/Item co-rating is used in collaborative filtering methods to build models. One class of CF methods, referred to as nearest-neighborhood-based methods, compute the similarities between the users/items using the co-rating information and new items are recommended based on these similarity values. Another class of CF methods, referred to as model-based methods, employ a machine learning algorithm to build a model (in terms of similarities or latent factors), which is then used to perform the recommendation task. These methods learn representation for users and items in common latent space and the recommendation score for a given user and item pair is computed as the dot product of the corresponding user and item latent vectors. In content based methods, users/items features are used to build models [9, 10]. In this thesis work, the focus is limited only to CF based methods. In the next section we discuss some of the state-of-the-art CF methods in detail.

3.1 Existing Methods

User (UserKNN) and Item (ItemKNN) k-nearest neighbors UserKNN [11, 12] is a classical user based CF method, which computes k-nearest neighbors for each user, based on their rating profiles. These nearest neighbors are then used to predict the rating for a user on an unrated item as the weighted average of the rating of the nearest neighbors of the user.

Similar to UserKNN method, ItemKNN [13, 14, 15] is a CF based method which identifies the k-nearest neighbors in the items space, based on their co-rating information. These item neighbors are then used to recommend new items to the users, based on the items which are similar to the items rated by the user.

The user/item nearest neighbors are calculated from user-item rating matrix in a collaborative way using a vector similarity measure like cosine similarity, pearson correlation etc., These methods are simple and easier to implement and are nonlinear in terms of the preferences of the user, which are implicitly captured via the nearest neighbors. However, they rely on the co-rating information between the users to compute the similarity. Thus, it suffers from data sparsity issue and fails to capture relations between users who do not have enough co-rated items.

Matrix Factorization (MF) Methods In recent years, approaches based on matrix factorization of the user-item rating matrix have emerged as a very powerful technique for rating prediction and top-N recommendation tasks [16, 17, 18, 19, 3, 20, 21, 22, 23]. MF based methods are known to outperform [24] other models including Bayesian models URP [25] and PLSA [26]. In these methods, the rating matrix $\mathbf{R}$ is approximated as a product of two low-rank matrices $\mathbf{P}$ and $\mathbf{Q}$, where $\mathbf{P} \in \mathbb{R}^{n \times k}$ is the users latent vector matrix, $\mathbf{Q} \in \mathbb{R}^{m \times k}$ is the items latent vector matrix, k is the number of latent factors and $k \ll n, m$. The recommendation score of a user u for item i is then predicted as,

$$\hat{r}_{ui} = \mathbf{p}_u \mathbf{q}_i^{\mathsf{T}}, \tag{3.1}$$

where $\mathbf{p}_u$ is the latent vector associated with user u and $\mathbf{q}_i$ is the latent vector associated with item i.

In [27, 28], classical dimensionality reduction technique, Singular Value Decomposition (SVD) is applied on the rating matrix $\mathbf{R}$. The resulting factorized matrices are used to represent the users and items latent factors. While applying the SVD algorithm, they impute the missing values of $\mathbf{R}$ with zeros. Thus, the user item rating matrix $\mathbf{R}$ is estimated by the factorization:

$$\hat{\mathbf{R}} = \mathbf{U} \cdot \Sigma \cdot \mathbf{V} \ , \tag{3.2}$$

where, $\mathbf{U}$ is a $n \times k$ orthonormal matrix, $\mathbf{V}$ is a $k \times m$ matrix and Σ is a $k \times k$ diagonal matrix containing the first k singular values. The product $\mathbf{U} \cdot \Sigma$ can be used to represent the user factors and $\mathbf{V}$ the item factors. The predicted recommendation score can then be computed similar to the MF method. Most of the MF based methods employ either Alternating Least Squares (ALS) [21] or Stochastic Gradient Descent (SGD) [29] based methods to learn the low dimensional embeddings of the users and items. One potential limitation of these approaches is that, although the user and item latent factors are learnt in a collaborative fashion, unlike neighborhood methods it learns the user and item latent factors by incorporating all the ratings in the data, and thus it does not explicitly take the neighborhood of the user/item into account while making the predictions.

Many extensions and variations to MF based methods are also proposed in the recent years. A probabilistic based approach for matrix factorization is proposed in [30]. Temporal dynamics is incorporated into matrix factorization in [31] to better model the users' time sensitive preferences. Blending of various CF models was proposed in [32, 33] to achieve the best rating prediction performance for the Netflix Prize [34]. In [6, 7], a Weighted Regularized Matrix Factorization (WRMF) method is proposed. This method is formulated as a regularized Least-Squares (LS) problem, in which a weighting matrix is used to differentiate the contributions from observed purchase/rating activities and unobserved ones. A Max-Margin Matrix Factorization (MMMF) method is proposed [24, 35, 36], which requires a low-norm factorization of the user-item matrix and allows unbounded dimensionality for the latent space. This is implemented by minimizing the tracenorm of the reconstructed user-item matrix from the factors. Sindhwani et al [37] proposed a Weighted Non-Negative Matrix Factorization (WNNMF) method, in which they enforce nonnegativity on the user and item factors so as to lend part-based

interpretability to the model. Probabilistic Latent Semantic Analysis (PLSA) technique was applied for collaborative filtering by Hofmann in [26]. It was also shown to be equivalent to non-negative matrix factorization (NNMF). Sparse matrix factorization algorithms are used in [38, 39, 40] to reduce the computational complexity in the context of rating prediction problem. In [41], Yu et. al. formulate the collaborative filtering as a maximum-margin matrix approximation problem with regularization. Agarwal et. al. in [42], proposed a regression based latent factor model for rating prediction problem.

NSVD & SVD++ One other popular extension of MF based methods is called NSVD and was developed by Paterek in [43]. This is a factored item-item collaborative filtering method developed for the rating prediction problem. In this method, an item-item similarity was learned as a product of two low-rank matrices, $\mathbf{P}$ and $\mathbf{Q}$, where $\mathbf{P} \in \mathbb{R}^{m \times k}$, $\mathbf{Q} \in \mathbb{R}^{m \times k}$, and $k \ll m$. This approach extends the traditional item-based neighborhood methods by learning the similarity between items as a product of their corresponding latent factors. Given two items i and j, the similarity $sim(i, j)$ between them is computed as the dot product between the corresponding factors from $\mathbf{P}$ and $\mathbf{Q}$ i.e., $sim(i, j) = \mathbf{p}_i \cdot \mathbf{q}_j^\mathsf{T}$. The rating for a given user u on item j is both predicted and estimated as,

$$\hat{r}_{ui} = \tilde{r}_{ui} = b_u + b_i + \sum_{j \in \mathcal{R}_u^+} \mathbf{p}_j \mathbf{q}_i^\mathsf{T}, \tag{3.3}$$

where b_u and b_i are the user and item biases and $\mathcal{R}_u^+$ is the set of items rated by u. The parameters of this model are estimated as the minimizer to the following optimization problem:

$$\underset{\mathbf{P},\mathbf{Q}}{\text{minimize}} \ \frac{1}{2} \sum_{u \in \mathcal{C}} \sum_{j \in \mathcal{R}_u^+} \|r_{ui} - \hat{r}_{ui}\|_F^2 + \frac{\beta}{2}(\|\mathbf{P}\|_F^2 + \|\mathbf{Q}\|_F^2), \tag{3.4}$$

where $\hat{r}_{ui}$ is the estimated value for user u and item j (as in Equation 3.3).

In another set of methods [44, 21], the latent factor models and neighborhood based methods are combined. In a popular based on NSVD, Koren proposed a hybrid approach called SVD++ [5]. This method smoothly merges the latent factor and neighborhood based models. The proposed model is extended to include both implicit [45] and explicit user feedback data. The latent factors part of the model consists of the standard user and item latent factors along with the second set of item factors which are used to model the

asymmetric relations between the items in the latent space. For the neighborhood part of the model, the relations between the items are explicitly learned in the form of weights between the items. The proposed model was shown to outperform rest of the state-of-the-art methods. One of the limitations of this method is w.r.t. learning large number of parameters. Along with modeling each user with a latent vector and each item with two latent vectors, it also learns a weight vector for each item to learn the item similarities in the neighborhood component. To reduce the number of parameters learned in the neighborhood component, only top-k nearest neighbors based on the items corating information is used to learn the item relations. Even then, hundereds to thousands of additional parameters needs to be learned for each of the items. Learning these parameters is computationally expensive and large number of parameters can potentially lead to an overfitted model. Also, both these models (i.e., NSVD and SVD++) were evaluated by computing the root mean square error (RMSE) on the test ratings in the Netflix competition data set. Hence the goal of these models was to minimize the RMSE and only the non-zero entries of the rating matrix were used in training.

Sparse Linear Methods (SLIM) Recently, a novel top-N recommendation method has been developed, called SLIM [46], which improves upon the traditional item-based nearest neighbor collaborative filtering approaches by learning the item relationships from the data, a sparse matrix of aggregation coefficients that are analogous to the traditional item-item similarities. SLIM predicts the recommendation scores of a user u for all items as,

$$\tilde{\mathbf{r}}_u = \mathbf{r}_u \mathbf{S}, \tag{3.5}$$

where $\mathbf{r}_u$ is the rating vector of u on all items and $\mathbf{S}$ is a $m \times m$ sparse matrix of aggregation coefficients.

Matrix $\mathbf{S}$ can be considered as an item-item similarity matrix, and as such the recommendation strategy employed by SLIM is similar in nature to that of the traditional item-based nearest-neighbor top-N recommendation approaches [14]. However, unlike these methods, SLIM directly estimates the similarity values from the data using a simultaneous regression approach, which is similar to structural equation modeling with no exogenous variables [47]. Specifically, SLIM estimates the sparse matrix $\mathbf{S}$ as the

minimizer for the following regularized optimization problem:

$$\begin{aligned} \underset{\mathbf{S}}{\text{minimize}} \;\; & \frac{1}{2}\|\mathbf{R} - \mathbf{RS}\|_F^2 + \frac{\beta}{2}\|\mathbf{S}\|_F^2 + \lambda\|\mathbf{S}\|_1 \\ \text{subject to} \;\; & \mathbf{S} \geq 0 \;, \;\; \text{diag}(\mathbf{S}) = 0, \end{aligned} \tag{3.6}$$

where $\|\mathbf{S}\|_F$ is the matrix Frobenius norm of $\mathbf{S}$ and $\|\mathbf{S}\|_1$ is the entry-wise ℓ_1-norm of $\mathbf{S}$. In Equation 3.6, $\mathbf{RS}$ is the estimated matrix of recommendation scores (i.e., $\tilde{\mathbf{R}}$). The constraint $\text{diag}(\mathbf{S}) = 0$ conforming to the structural equation modeling is also applied to ensure that r_{ui} is not used to compute r_{ui}. The non-negativity constraint is applied on $\mathbf{S}$ so that the learned $\mathbf{S}$ corresponds to positive aggregations over items. In order to learn a sparse $\mathbf{S}$, SLIM introduces the ℓ_1-norm of $\mathbf{S}$ as a regularizer in Equation 3.6 [48]. Along with ℓ_1-norm, ℓ_F-norm is also used as another regularizer, which leads the optimization problem to an elastic net problem [49]. The ℓ_F-norm is used to prevent overfitting of the model to the training data. The matrix $\mathbf{S}$ learned by SLIM is referred to as SLIM's aggregation coefficient matrix. Extensive experiments in [46] have shown that SLIM outperforms the rest of the state-of-the-art top-N recommendation methods.

SLIM has been shown to achieve good performance on a wide variety of datasets and to outperform other state-of-the-art approaches. However, an inherent limitation of SLIM is that it can only model relations between items that have been co-purchased/co-rated by at least some users. As a result, it cannot capture transitive relations between items that are essential for good performance of item-based approaches in sparse datasets.

Nonlinear latent factorization (MaxMF) One of the recently developed methods for top-N recommendation called MaxMF [50], extends the matrix factorization based approaches by representing the user with multiple latent vectors, each corresponding to a different "taste" associated with the user. These different tastes associated with each user representation are termed as *interests*. The assumption behind this approach is that, by letting the users to have multiple interests, it helps to capture user preferences better, especially when the itemsets or user's interests are diverse. Thus, the user is represented with T different interests and a max function based nonlinear model was proposed. The model takes the maximum scoring interest as the final recommendation score for a given user item pair, i.e., the interest which matches the best with the given

item is captured in this model. In other words, the set of items is partitioned into T partitions for each user, and this partitioning process is personalized on the user. For each such item partition a different scoring function is used to estimate the rating. The user factors are thus represented by a tensor $\mathbf{P}$, where $\mathbf{P} \in \mathbb{R}^{n \times k \times T}$. The items factors, $\mathbf{Q}$ remains similar to MF based approaches. Thus, each user u is represented by $\mathbf{p}_u$, where $\mathbf{p}_u \in \mathbb{R}^{k \times T}$. For a given user u and item i pair, the predicted recommendation score is calculated by computing T dot products between each of the T user vectors and the corresponding item vector. The highest scoring dot product is taken as the estimated/predicted rating. That is,

$$\hat{r}_{ui} = \max_{t=1,\ldots,T} \mathbf{p}_{ut} \mathbf{q}_i^{\mathsf{T}}, \tag{3.7}$$

where the max function computes the maximum of the set of dot products between each of $\mathbf{p}_{ut}$ and $\mathbf{q}_i$.

It was shown that MaxMF achieves better top-N recommendation performance compared to other state-of-the-art methods. However, a limitation of this method is that it assumes that the interest-specific preferences of the users are completely different. Another limitation of MaxMF is w.r.t. data sparsity. When the data gets sparse i.e., when less preference data is available for each user, this approach can potentially dilute the learnt latent factors for users who do not have enough diversity in their itemsets due to lack of support (in terms of number of rated items) for each of the interests. This problem is magnified, as the data gets sparser.

In [51], a nonlinear matrix factorization approach with Gaussian processes is proposed. This method uses a kernelized form for the model. Salakhutdinov et. al. in [52] applied Restricted Boltzmann machines for collaborative filtering, which is a form of neural networks that introduces nonlinearities via Gaussian hidden units.

Bayesian Personalized Ranking (BPR) Methods In one of the recent methods, Top-N recommendation has also been formulated as a ranking problem. Rendle et al [53] proposed a Bayesian Personalized Ranking (BPR) criterion, which is the maximum posterior estimator from a Bayesian analysis and measures the difference between the rankings of items which are purchased and not purchased by the user. A differentiable loss function is used to optimize the AUC. BPR was adopted for both item knn method (BPRkNN) and MF methods (BPRMF) as a general objective function.

Clustering Based Methods In another class of methods, clustering is employed to improve the scalability and accuracy of the CF methods. In [54, 55, 56, 57, 58], clustering techniques are used to partition the users or items into clusters and a memory-based collaborative algorithm like UserKNN or ItemKNN is applied to make predictions for each of the cluster. Ungar et. al. [59] use variations of k-means and Gibbs sampling to cluster users based on the items they rated and clustering items based on the users who rated them. Users are then reclustered based on the number of items they rated and similarly the items are reclustered based on the number of users who ratem them. This alternating process is repeated till convergence. In [60], flexible mixture model is applied to cluster both the users and item at the same time, allowing the users and items to be part of multiple clusters. A membership score is assigned to each of the clusters that the users and items belong to. Many other clustering based approaches have been proposed to improve the scalability of the existing CF approaches [61, 62, 63, 64]. A good survey on the clustering based CF methods can be found in [65].

Other methods Several other methods has been proposed in the recent years that formulate the top-N problem as a ranking problem. For a given user, pairwise optimization methods [66] like Ordinal Regression [67], WSABIE [68], COFI-RANK [69], EigenRank [70] and CLiMF [71] optimize a ranking loss with the goal of ranking higher valued items above lower scoring items. Many of these methods rely on Learning to Rank optimization algorithms used in information retrieval domain [72]. A different approach was proposed in [73], in which the rating prediction problem was formulated as a binary classification problem and a one-class classifier was built for each user, and all the classifiers were learned and boosted together.

A review on early works of traditional collaborative filtering is available in [19] and a review on recent collaborative filtering methods can be found in [3, 65]. Matrix Factorization has gained popularity in the recent years and has also achieved the state-of-the-art recommendation performance particularly on large-scale rating prediction problems. A review on such MF based methods for recommender systems is available in [16]. In recent years, many methods and frameworks [74, 75, 76, 77, 78, 79] have been proposed to improve the scalability of the collaborative filtering algorithms to large-scale problems.

3.2 Loss Functions

In this thesis, we employ two different loss functions to estimate the parameters of the proposed models. First, is the squared loss, which computes the total loss as the sum of the square of the difference between the actual value and the estimated value. Given a set of data points X, the squared error loss function is given by,

$$\mathcal{L}(\cdot) = \sum_{x \in X} (x - \hat{x})^2, \tag{3.8}$$

where x is the actual value and $\hat{x}$ is the estimated value of the data.

Second loss function used is the BPR criterion function. BPR employs a pairwise ranking criterion between the data points. Unlike squared loss which computes loss on the ground truth value of the data point, BPR computes loss only the estimated values of the data points. It measures the difference between the estimated values of the data points which have a higher value with the ones which have a lower value. Given a set of data points X, the BPR criterion is given by,

$$\mathcal{L}(\cdot) \ = \sum_{(x_i, x_j) \in X} \ln \sigma(\hat{x}_i - \hat{x}_j) \tag{3.9}$$

where x_i is a higher valued data point compared to x_j, $\sigma(x)$ is the sigmoid function, i.e., $\sigma(x) = 1/(1 + e^{-x})$. The sigmoid and log functions are used to make the criterion function differentiable, which is helpful in using gradient based methods to optimize the objective function corresponding to the ranking criterion.

3.3 Optimization Algorithms

In this thesis we utilize the Stochastic Gradient Descent (SGD) based algorithm to optimize (minimize/maximize) the objective function associated with the different proposed methods. First the general formulation of the SGD algorithm is presented. Then the SGD algorihm in the context of squared loss and BPR criterion is presented.

Stochastic Gradient Descent (SGD) is a gradient descent optimization method for minimizing an objective function that is expressed as a sum of differentiable functions. SGD based approximations are known to perform poorly for optimization tasks, while their performance is extermely good for machine learning tasks [29]. Historically,

Stochastic gradient algorithms have been associated with back-propagation learning methods in multilayer neural networks, which can be very challenging non-convex problems.

Given a objective function $Q(w)$, which is a sum of differentiable functions, i.e., $Q(w) = \sum_{i=1}^{n} Q_i(w)$, where $Q_i(w)$ is the function associated with i-th sample in the data set and n is the number of data samples, a standard gradient descent (also known as Steepest Gradient Descent) based iterative optimization technique updates the weights w on of the basis of the gradients $\nabla Q_i(w)$ in each iteration. That is,

$$w_t = w_{t-1} - \eta \sum_{i=1}^{n} \nabla Q_i(w), \tag{3.10}$$

where η is the learning rate and t is the current iteration number.

The SGD is a simplification of the steepest gradient descent. Instead of computing the gradient of $Q_i(w)$ exactly, each iteration estimates this gradient on the basis of a single randomly selected data sample. That is, given a sample i, the SGD update rule is given by,

$$w_t = w_{t-1} - \eta \nabla Q_i(w). \tag{3.11}$$

This update is repeated for each of the randomly picked training sample of the data set. Several passes over the training data is made until the algorithm converges. The generalized pseudocode for the SGD algorithm is given in Algorithm 1.

SGD is shown to converge to a local minimum and have a convergance rate which is independent of the size of the data. Thus SGD is highly suitable for large scale datasets. Another advantage of SGD is that, since the SGD algorithm does not need to remember which examples were visited during the previous iterations, it can process data samples on the fly in an online setting.

Squared Loss

In case of squared loss, the parameters of the model, Θ are learned by minimizing the following regularized objective function,

$$Squared - OPT \ = \sum_{x \in X} (x - \hat{x})^2 + \lambda_\Theta \|\Theta\|^2 \tag{3.12}$$

where λ_Θ is the regularization constant corresponding to parameter Θ. The details of the corresponding SGD based learning algorithm is provided in Algorithm 2.

Algorithm 1 SGD:Opt.

```
1: procedure SGD_OPT
2:     w ← weight vector to learn
3:     η ← learning rate
4:     iter ← 0
5:     Init w with random values
6:
7:     while iter < maxIter or error on validation set decreases do
8:         for all i = 1, 2, 3, ..., n do
9:             w_t ← w_{t−1} − η∇Q_i(w)
10:        end for
11:        iter ← iter + 1
12:    end while
13:
14:    return w
15: end procedure
```

Bayesian Personalised Ranking (BPR)

Bayesian Personalised Ranking (BPR) [53] is a generic method for solving the personalized ranking task. The approach mainly consists of a general optimization criterion (known as $BPR-OPT$) for personalized ranking. BPR measures the difference between the rankings of items which are purchased and not purchased by the user. For a given user u, purchased item i and unpurchased item j, $\hat{x}_{uij}(\Theta)$ represents an arbitrary real valued function which approximates the bayesian formulation of the personalized ranking, where Θ captures the relationship between the triplet (u,i,j). The generic framework of BPR lets the underlying method like matrix factorization or k-nearest neighbors to model these relationships between the users and items. A differentiable loss function is used to optimize the criterion which is shown to optimize the AUC metrix. Given D_S, a set of sampled triplets of (u, i, j), the objective function of the BPR is given by,

$$BPR-OPT \;=\; \sum_{(u,i,j)\in D_S} \ln \sigma(\hat{x}_{uij}) - \lambda_\Theta \|\Theta\|^2 \tag{3.13}$$

where λ_θ are model specific regularization parameters. Parameters of the $BPR-OPT$ can be learnt using a Stochastic Gradient Descent (SGD) based method. The algorithm for learning the model parameters using a SGD based method is outlined in Algorithm 3.

Algorithm 2 SquaredLoss:Opt.

procedure SQUAREDLOSS_OPT
  $w \leftarrow$ parameter vector to learn
  $\eta \leftarrow$ learning rate
  $iter \leftarrow 0$
  Init w with random values

  while $iter < maxIter$ or error on validation set decreases **do**
    for all $x_i \in X$ **do**
      $e_i = x_i - \hat{x_i}$
      $w_t \leftarrow w_{t-1} + \eta \cdot (e_i - \lambda_\Theta \cdot \Theta)$
    end for
    $iter \leftarrow iter + 1$
  end while

  return w
end procedure

Algorithm 3 BPR:Opt.

procedure BPR_OPT
  $D_S \leftarrow$ sampled triplets (u, i, j) from $\mathbf{R}$
  $\eta \leftarrow$ learning rate
  $\Theta \leftarrow$ parameters to learn
  Init Θ with random values

  while not converged **do**
    draw (u, i, j) from D_S randomly
    $\Theta \leftarrow \Theta + \eta \, (\frac{e^{-\hat{x}_{uij}}}{1+e^{-\hat{x}_{uij}}} \cdot \frac{\partial}{\partial \Theta} \hat{x}_{uij} + \lambda_\Theta \cdot \Theta)$
  end while

  return Θ
end procedure

Chapter 4

Datasets & Evaluation Methodology

4.1 Datasets

In this thesis, the performance of proposed methods is evaluated on different real datasets, namely ML100K, ML1M, Netflix, Grades, Flixster and Yahoo Music. ML100K and ML1M are the subsets of data obtained from the MovieLens[1] research project, Netflix is a subset of data extracted from Netflix Prize dataset[2] , Grades is the dataset collected from a student database at an academic institution and it consists of students as users and courses as items and the grades obtained by the students in courses represents the rating values, Flixster is a subset of data extracted from publicly available data set collected from Flixster and finally Yahoo Music is the subset of data obtained from Yahoo! Research Alliance Webscope program[3] . For each of the ML100K, Netflix, Yahoo and Flixster datasets, different versions at different sparsity levels are created. This was done to specifically evaluate the performance of the proposed method on sparse datasets. For each dataset, a random subset of users and items are selected from the main dataset. These datasets are represented with a *'-1'* suffix. Keeping the same set of users and items, the first sparser version of the datasets with the *'-2'* suffix are created

[1] http://www.grouplens.org/node/12
[2] http://www.netflixprize.com/
[3] http://research.yahoo.com/academic_relations

by randomly removing entries from the first datasets' user-item matrices. The second sparser version of the datasets with the *'-3'* suffix are similarly created by randomly removing entries from second datasets' user-item matrices. Note that all these datasets have rating values and we converted them into implicit feedback by setting the positive entries to 1 to evaluate the proposed methods in the context of the top-N recommendation task. The characteristics of all the datasets is summarized in Table 4.1. For the different methods presented in this thesis, we use a subset of the listed datasets.

Table 4.1: Datasets

Dataset	#Users	#Items	#Ratings	Density
ML100K	943	1,349	99,287	7.80%
ML100K-1	943	1,178	59,763	5.43%
ML100K-2	943	1,178	39,763	3.61%
ML100K-3	943	1,178	19,763	1.80%
ML1M	6,040	3,952	1,000,209	4.19%
Netflix	5,403	2,933	2,197,096	13.86%
Netflix-1	6,079	5,641	429,339	1.25%
Netflix-2	6,079	5,641	221,304	0.65%
Netflix-3	6,079	5,641	110,000	0.32%
Netflix-4	5,403	2,933	1,649,174	10.41%
Grades	19,319	9,562	422,203	0.23%
Flixster-1	4,627	3,295	1,184,817	7.77%
Flixster-2	4,627	3,295	889,842	5.84%
Yahoo	5,824	15,869	1,440,212	1.56%
Yahoo-1	7,558	3,951	282,075	0.94%
Yahoo-2	7,558	3,951	149,050	0.50%
Yahoo-3	7,558	3,951	75,000	0.25%

The "#Users", "#Items" and "#Ratings" columns are the number of users, items and ratings respectively, in each of the datasets. The "Rating Scale" column is the range of the ratings present. The "Density" column is the density of each dataset (i.e., density = #Ratings/(#Users × #Items)).

4.2 Evaluation Methodology

The performance of the methods presented in this thesis are evaluated using a 5-fold Leave-One-Out-Cross-Validation (LOOCV) method similar to the one employed in [46]. Training and test set is created by randomly selecting one rated item per user from the dataset and placing it in the test set. The rest of the data is used as the training set. This process is repeated to create five different folds. Only for the Grades dataset, the test set is created by using the grades of the last semester for each student. The rest of the grades from previous semesters is used as the training set. The training set is used to build the model and the trained model is then evaluated on the test set. For rating prediction task, the recommendation scores from the trained model is used as the predicted ratings. For the top-N recommendation task, the trained model is then used to generate a ranked list of size-N items for each user. The model is then evaluated by comparing the ranked list of recommended items with the item in the test set. Unless specified, for all the results presented in this thesis, N is equal to 10.

4.3 Performance Metrics

The recommendation quality of the rating prediction task, for each user the trained model is used to predict the rating for the unrated items in the test set. The model is then evaluated by computing the test metric for each of the test user-item rating pair. The rating prediction quality is measured using Root Mean Square Error(RMSE). RMSE is defined as,

$$RMSE = \frac{1}{|Test|}\sqrt{\sum_{r_{ui} \in Test} (r_{ui} - \hat{r}_{ui})^2},$$

where r_{ui} is the ground truth value and $\hat{r}_{ui}$ is the predicted rating value for a given user u and item i and $Test$ is the test data consisting of test user-item pairs.

For the top-N recommendation task, the trained model is evaluated by comparing the ranked list of recommended items with the item in the test set. The recommendation quality for top-N recommendation task is measured using Hit Rate (HR) and Average Reciprocal Hit Rank (ARHR) [14]. HR is defined as,

$$HR = \frac{\#hits}{\#users},$$

where $\#hits$ is the number of users for which the model was successfully able to recall the test item in the size-N recommendation list and $\#users$ is the total number of test users. The ARHR is defined as,

$$ARHR = \frac{1}{\#users} \sum_{i=1}^{\#hits} \frac{1}{pos_i},$$

where pos_i is the position of the test item in the ranked recommendation list for the i^{th} hit. ARHR represents the weighted version of HR, as it measures the inverse of the position of the recommended item in the ranked list.

We chose RMSE, HR and ARHR as evaluation metrics since they directly measure the performance of the model on the ground truth data i.e., what users have already provided feedback for.

4.4 Comparison Algorithms

The performance of the different methods proposed in this thesis are compared against to that achieved by a number of the current state-of-the-art methods. These methods include UserKNN [11], ItemKNN [14], ItemKNN (cprob) [14], ItemKNN (log)[4] , PureSVD [28], MF [28], MaxMF [50], BPRMF & BPRkNN [53] and SLIM [46]. The details of these methods are provided in the Section 3.1. This set of methods constitute the current state-of-the-art for rating prediction and top-N recommendation task. Hence they form a good set of methods to compare and evaluate our proposed approach against.

The following parameter space was explored for each of these methods and the best performing model in that parameter space in terms of RMSE/HR is reported. For UserKNN, PureSVD, MF, BPRMF and MaxMF, parameter k was selected from the range 2 to 800. For ItemKNN (cprob), the α parameter was selected from the range 0 to 1. Learning rate and regularization constants for MF and BPRMF was selected from the range 10^{-5} to 1.0. For BPRkNN, the learning rate and λ were selected from the range 10^{-5} to 1.0 For SLIM and MaxMF, the regularization constants were selected from the range 10^{-5} to 20. For MaxMF the number of local preferences T was selected from the range 1 to 6.

[4] Part of Mahout library (http://mahout.apache.org/)

Chapter 5

FISM: Factored Item Similarity Methods for Top-N Recommender Systems

This chapter focuses on developing an effective algorithm for top-N recommender systems. A novel Factored Item Similarities based method FISM is proposed, which learns the item-item similarity matrix as a product of two low-dimensional latent factor matrices. This factored representation of the item-item similarity matrix allows FISM to capture and model relations between items even on very sparse datasets. Our experimental evaluation on multiple datasets and at different sparsity levels confirms that and shows that FISM performs better than SLIM and other state-of-the-art methods. Moreover, the relative performance gains increase with the sparsity of the datasets.

5.1 Introduction

In real world scenarios, users typically provide feedback (purchase, rating or review) to only a handful of items out of possibly thousands or millions of items. This results in the user-item rating matrix becoming very sparse. Methods like SLIM (as well as traditional methods like ItemKNN), which rely on learning similarities between items, fail to capture the dependencies between items that have not been co-rated by at least

one user. It can be shown that the minimizer in SLIM will have $s_{ij} = 0$, if i and j have not been co-rated by at least one user. But two such items can be similar to each other by virtue of another item which is similar to both of them (transitive relation). Methods based on matrix factorization, alleviate this problem by projecting the data onto a low dimensional space, thereby implicitly learning better relationships between the users and items (including items which are not co-rated). However, such methods are consistently out-performed by SLIM.

To overcome this problem, the proposed item-oriented FISM method uses a factored item similarity model similar in spirit to that used by NSVD and SVD++. Learning the similarity matrix by projecting the values in a latent space of much smaller dimensionality, implicitly helps to learn transitive relations between items. Hence, this model is expected to perform better even on sparse data, as it can learn relationships between items which are not co-rated.

Comparing FISM with NSVD, besides the fact that these two methods are designed to solve different problems (top-N vs rating prediction), their key difference lies in how the factored matrices are estimated. FISM employs a regression approach based on structural equation modeling in which, unlike NSVD (and SVD++), the known rating information for a particular user-item pair (r_{ui}) is not used when the rating for that item is being estimated. This impacts how the diagonal entries of the item-item similarity matrix corresponding to $\mathbf{S} = \mathbf{P}\mathbf{Q}^{\mathsf{T}}$ influence the estimation of the recommendation score. Diagonal entries in the item similarities matrix correspond to including an item's own value while computing the prediction for that item. NSVD does not exclude the diagonal entries while estimating the ratings during learning and prediction phases, while FISM explicitly excludes the diagonal entries while estimating. This shortcoming of NSVD impacts the quality of the estimated factors when the number of factors becomes large. In this case it can lead to rather trivial estimates, in which an item ends up recommending itself. This is illustrated in our experimental results (Section 5.3), which show that for a small number of factors, the two estimation approaches produce similar results, whereas as the number of factors increases moderately, FISM's estimation approach consistently and significantly outperforms the approach used by NSVD.

The key contributions of the FISM method presented in this chapter are the following,

(i) extends the factored item-based methods to the top-N problem, which allow them to effectively handle sparse datasets;

(ii) estimates the factored item-based top-N models using a structural equation modeling approach;

(iii) estimates the factored item-based top-N models using both squared error and a ranking loss; and

(iv) investigates the impact of various parameters as they relate to biases and model's induced sparsity.

5.2 FISM - Factored Item Similarity Methods

In FISM, the recommendation score for a user u on an unrated item j (denoted by $\tilde{r}_{ui}$) is calculated as an aggregation of the items that have been rated by u with the corresponding product of $\mathbf{p}_j$ latent vectors from $\mathbf{P}$ and the $\mathbf{q}_i$ latent vector from $\mathbf{Q}$. That is,

$$\tilde{r}_{ui} = b_u + b_i + (n_u^+)^{-\alpha} \sum_{j \in \mathcal{R}_u^+} \mathbf{p}_j \mathbf{q}_i^\mathsf{T}, \tag{5.1}$$

where $\mathcal{R}_u^+$ is the set of items rated by user u, $\mathbf{p}_j$ and $\mathbf{q}_i$ are the learned item latent factors, n_u^+ is the number of items rated by u, and α is a user specified parameter between 0 and 1.

The term $(n_u^+)^{-\alpha}$ in Equation 5.1 is used to control the degree of agreement between the items rated by the user with respect to their similarity to the item whose rating is being estimated (i.e., item j). To better understand this, consider the case in which $\alpha = 1$. In this case (excluding the bias), the predicted rating is the average similarities between the items rated by the user (i.e., $\mathcal{R}_u^+$) and item j. Item j will get a high rating if nearly all of the items in $\mathcal{R}_u^+$ are similar to j. On the other hand, if $\alpha = 0$, then the predicted rating is the aggregate similarity between j and the items in $\mathcal{R}_u^+$. Thus, j can be rated high, even if only one (or few) of the items in $\mathcal{R}_u^+$ are similar to j. These two settings represent different extremes and we believe that in most cases the right choice will be somewhere in between. That is, the item for which the rating is being predicted needs to be similar to a substantial number of items to get a high rating. To

capture this difference, we have introduced the parameter α, to control the number of neighborhood items that need to be similar for an item to get the high rating. The value of α is expected to be dependent on the characteristics of the dataset and its best performing value is determined empirically.

We developed two different types of FISM models that use different loss functions and associated optimization methods, which are described in the next two sections.

5.2.1 FISMrmse

In FISMrmse, the loss is copmuted using the squared error loss function, given by

$$\mathcal{L}(\cdot) = \sum_{i \in \mathcal{D}} \sum_{u \in \mathcal{C}} (r_{ui} - \hat{r}_{ui})^2, \tag{5.2}$$

where r_{ui} is the ground truth value and $\hat{r}_{ui}$ is the estimated value. The estimated value $\hat{r}_{ui}$, for a given user u and item j is computed as

$$\hat{r}_{ui} = b_u + b_i + (n_u^+ - 1)^{-\alpha} \sum_{j \in \mathcal{R}_u^+ \setminus \{i\}} \mathbf{p}_j \mathbf{q}_i^\mathsf{T}, \tag{5.3}$$

where $\mathcal{R}_u^+ \setminus \{i\}$ is the set of items rated by user u, excluding the current item j, whose value is being estimated. This exclusion is done to conform to regression models based on structural equation modeling. This is also one of the important differences between FISM and other factored item similarities model (like NSVD and SVD++) as discussed in Section 5.1.

In FISMrmse, the matrices $\mathbf{P}$ and $\mathbf{Q}$ are learned by minimizing the following regularized optimization problem:

$$\underset{\mathbf{P},\mathbf{Q}}{\text{minimize}} \quad \frac{1}{2} \sum_{u,i \in R} \|r_{ui} - \hat{r}_{ui}\|_F^2 + \frac{\beta}{2}(\|\mathbf{P}\|_F^2 + \|\mathbf{Q}\|_F^2) + \frac{\lambda}{2}\|\mathbf{b_u}\|_2^2 + \frac{\gamma}{2}\|\mathbf{b_i}\|_2^2, \tag{5.4}$$

where the vectors $\mathbf{b_u}$ and $\mathbf{b_i}$ correspond to the vector of user and item biases, respectively. The regularization terms are used to prevent overfitting and β, λ and γ are the regularization weights for latent factor matrices, user bias vector and item bias vector respectively.

Following the common practices for top-N recommendation [28, 46], note that the loss function in Equation 5.2 is computed over all entries of $\mathbf{R}$ (i.e., both rated and

unrated). This is in contrast with rating prediction methods, which compute the loss over only the rated items. However, in order to reduce the computational requirements for optimization, the zero entries are sampled and used along with all the non-zero values of $\mathbf{R}$. During each iteration of learning, $\rho \cdot nnz(R)$ zeros are sampled and used for optimization. Here ρ is a constant and $nnz(R)$ is the number of non-zero entries in $\mathbf{R}$. Our experimental results indicate that a small value of ρ (in the range $3-15$) is sufficient to produce the best model. This sampling strategy makes FISMrmse computationally efficient.

The optimization problem of Equation 5.4 is solved using a Stochastic Gradient Descent (SGD) algorithm [80]. Algorithm 4 provides the detailed procedure and gradient update rules. $\mathbf{P}$ and $\mathbf{Q}$ are initialized with small random values as the initial estimate (line 6). In each iteration of SGD (Lines 8 – 26), based on the sampling factor (ρ), a different set of zeros are sampled and used for training along with the non-zero entries of $\mathbf{R}$. This process is repeated until the error on the validation set does not decrease further or the number of iterations has reached a predefined threshold.

5.2.2 FISMauc

As a second loss function, a ranking error based loss function is considered. This is motivated by the fact that the Top-N recommendation problem deals with ranking the items in the right order, unlike the rating prediction problem where minimizing the RMSE is the goal. We used a ranking loss function based on Bayesian Personalized Ranking (BPR) [53], which optimizes the area under the curve (AUC). Given user's rated items in $\mathcal{R}_u^+$ and unrated items in $\mathcal{R}_u^-$, the overall ranking loss is given by

$$\mathcal{L}(\cdot) = \sum_{u \in \mathcal{C}} \sum_{i \in \mathcal{R}_u^+, j \in \mathcal{R}_u^-} ((r_{ui} - r_{uj}) - (\hat{r}_{ui} - \hat{r}_{uj}))^2, \tag{5.5}$$

where the estimates $\hat{r}_{ui}$ and $\hat{r}_{uj}$ are computed as in Equation 5.3. As we can see in Equation 5.5, the error is computed as the relative difference between the actual non-zero and zero entries and the difference between their corresponding estimated values. Thus, this loss function focuses not on estimating the right value, but on the ordering of the zero and non-zero values.

Algorithm 4 FISMrmse:Learn.

procedure FISMrmse_LEARN
  $\eta \leftarrow$ learning rate
  $\beta \leftarrow \ell_F$ regularization weight
  $\rho \leftarrow$ sample factor
  $iter \leftarrow 0$
  Init $\mathbf{P}$ and $\mathbf{Q}$ with random values in (-0.001, 0.001)

  while $iter < maxIter$ or error on validation set decreases **do**
    $\mathcal{R}' \leftarrow \mathbf{R} \cup$ *SampleZeros(*$\mathbf{R}$*,* ρ*)*
    $\mathcal{R}' \leftarrow$ *RandomShuffle(*$\mathcal{R}'$*)*

    for all $r_{ui} \in \mathcal{R}'$ **do**
      $\mathbf{x} \leftarrow (n_u^+ - 1)^{-\alpha} \sum_{j \in \mathcal{R}_u^+ \setminus \{i\}} \mathbf{p}_j$

      $\tilde{r}_{ui} \leftarrow b_u + b_i + \mathbf{q}_i^\mathsf{T} \mathbf{x}$
      $e_{ui} \leftarrow r_{ui} - \tilde{r}_{ui}$
      $b_u \leftarrow b_u + \eta \cdot (e_{ui} - \lambda \cdot b_u)$
      $b_i \leftarrow b_i + \eta \cdot (e_{ui} - \gamma \cdot b_i)$
      $\mathbf{q}_i \leftarrow \mathbf{q}_i + \eta \cdot (e_{ui} \cdot \mathbf{x} - \beta \cdot \mathbf{q}_i)$

      for all $j \in \mathcal{R}_u^+ \setminus \{i\}$ **do**
        $\mathbf{p}_j \leftarrow \mathbf{p}_j + \eta \cdot (e_{ui} \cdot (n_u^+ - 1)^{-\alpha} \cdot \mathbf{q}_i - \beta \cdot \mathbf{p}_j)$
      end for
    end for
    $iter \leftarrow iter + 1$
  end while

  return $\mathbf{P}, \mathbf{Q}$
end procedure

In FISMauc, the matrices $\mathbf{P}$ and $\mathbf{Q}$ are learned by minimizing the following regularized optimization problem:

$$\underset{\mathbf{P},\mathbf{Q}}{\text{minimize}} \ \frac{1}{2}\sum_{u\in\mathcal{C}}\sum_{i\in\mathcal{R}_u^+,j\in\mathcal{R}_u^-}\|(r_{ui}-r_{uj})-(\hat{r}_{ui}-\hat{r}_{uj})\|_F^2 + \frac{\beta}{2}(\|\mathbf{P}\|_F^2+\|\mathbf{Q}\|_F^2)+\frac{\gamma}{2}(\|\mathbf{b_i}\|_2^2), \quad (5.6)$$

where the terms mean the same as in Equation 5.4. Note that there are no user bias terms (i.e., b_u), since the terms cancel out when taking the difference of the ratings. For each user, FISMauc computes loss over all possible pairs of entries in $\mathcal{R}_u^+$ and $\mathcal{R}_u^-$. Similar to FISMrmse, to reduce the computational requirements, zero entries for each user are sampled from $\mathcal{R}_u^-$ based on sample factor (ρ).

The optimization problem in Equation 5.6 is solved using a Stochastic Gradient Descent (SGD) based algorithm. Algorithm 5 provides the detailed procedure.

5.2.3 Scalability

The scalability of these methods consists of two aspects. First, the training phase needs to be scalable, so that these methods can be used with larger datasets. Second, the time taken to compute the recommendations needs to be reduced and ideally made independent of the total number of recommendable items. Regarding the first aspect, the training for both FISMrmse and FISMauc is done using SGD algorithm. The gradient computations and updates of SGD can be parallelized and hence these algorithms can be easily applied to larger datasets. In [76], a distributed SGD is proposed. A similar algorithm with modifications can be used to scale the FISM methods to larger datasets. The main difference is in computing the rows of $\mathbf{P}$ that can be updated independently in parallel. There are also software packages like Spark[1] which can be used to implement SGD based algorithms on a large cluster of processing nodes.

For computing the recommendations efficiently during run time, methods like SLIM enforce sparsity constraint on $\mathbf{S}$ while learning and utilizes this sparsity structure to reduce the number of computations during run time. However, in FISM, the factored matrices learned are usually dense and as such, the predicted vector $\tilde{\mathbf{r}}_u$ will be dense

[1] http://spark-project.org/

Algorithm 5 FISMauc:Learn.

procedure FISMauc_LEARN
  $\eta \leftarrow$ learning rate
  $\beta \leftarrow \ell_F$ regularization weight
  $\rho \leftarrow$ number of sampled zeros
  $iter \leftarrow 0$
  Init $\mathbf{P}$ and $\mathbf{Q}$ with random values in (-0.001, 0.001)

  while $iter < maxIter$ or error on validation set decreases **do**
    for all $u \in \mathcal{C}$ **do**
      for all $i \in \mathcal{R}_u^+$ **do**
        $\mathbf{x} \leftarrow 0$
        $\mathbf{t} \leftarrow (n_u^+ - 1)^{-\alpha} \sum_{j \in \mathcal{R}_u^+ \setminus \{i\}} \mathbf{p}_j$
        $\mathcal{Z} \leftarrow$*SampleZeros(ρ)*

        for all $j \in \mathcal{Z}$ **do**
          $\tilde{r}_{ui} \leftarrow b_i + \mathbf{t} \cdot \mathbf{q}_i^\mathsf{T}$
          $\tilde{r}_{uj} \leftarrow b_j + \mathbf{t} \cdot \mathbf{q}_j^\mathsf{T}$
          $r_{uj} \leftarrow 0$
          $e \leftarrow (r_{ui} - r_{uj}) - (\tilde{r}_{ui} - \tilde{r}_{uj})$
          $b_i \leftarrow b_i + \eta \cdot (e - \gamma \cdot b_i)$
          $b_j \leftarrow b_j - \eta \cdot (e - \gamma \cdot b_j)$
          $\mathbf{q}_i \leftarrow \mathbf{q}_i + \eta \cdot (e \cdot \mathbf{t} - \beta \cdot \mathbf{q}_i)$
          $\mathbf{q}_j \leftarrow \mathbf{q}_j - \eta \cdot (e \cdot \mathbf{t} - \beta \cdot \mathbf{q}_j)$
          $\mathbf{x} \leftarrow \mathbf{x} + e \cdot (\mathbf{q}_i - \mathbf{q}_j)$
        end for
      end for

      for all $j \in \mathcal{R}_u^+ \setminus \{i\}$ **do**
        $\mathbf{p}_j \leftarrow \mathbf{p}_j + \eta \cdot (\frac{1}{\rho} \cdot (n_u^+ - 1)^{-\alpha} \cdot \mathbf{x} - \beta \cdot \mathbf{p}_j)$
      end for
    end for

    $iter \leftarrow iter + 1$
  end while

  return $\mathbf{P}, \mathbf{Q}$
end procedure

(because $\mathbf{PQ}^\mathsf{T}$ is dense). Sparsity in $\tilde{\mathbf{r}}_u$ can be introduced by computing $\mathbf{S} = \mathbf{PQ}^\mathsf{T}$ and then setting the smaller values to zero. One systematic way of doing this is to selectively retain only those non-zero entries which contribute the most to the length of the item similarities vector represented by the column in $\mathbf{S}$ to which the entry belongs. The impact of this sparsification is further explored in the experimental results.

5.3 Results

The experimental evaluation consists of two parts. First, the effect of various model parameters of FISM on the recommendation performance is studied. Specifically, how bias, induced sparsity, estimation approach and non-negativity affects the top-N performance is studied. These studies are presented only on the ML100K-3 (represented as ML100K), Yahoo-2 (represented as Yahoo) and Netflix-3 (represented as Netflix) datasets. However the same results and conclusions carry over to the rest of the datasets as well. These datasets are chosen to represent the datasets from different sources and at different sparsity levels. Unless specified all results in the first set of experiments are based on FISMrmse. Second, the comparison results with other competing methods (Section 4.4) on all the datasets is presented. The performance of FISM is also compared for different values of N (as in top-N) and finally the performance of FISM is compared with respect to data sparsity.

5.3.1 Effect of Bias

In FISM's model, the user and item biases are learned as part of the model. In this study we compare the influence of user and item biases on the overall performance of the model. We compare the following four different schemes, $NoBias$ - where no user or item bias is learned as part of the model, $UserBias$ - only the user bias is learned, $ItemBias$ - only the item is learned and $User\&ItemBias$ - where both user and item biases are learned. The results are presented in Table 5.1. The results indicate that the biases affect the overall performance, with item bias leading to the greatest gains in performance.

Table 5.1: Performance of different bias schemes.

Scheme	ML100K				Yahoo			
	Beta	**Lambda**	**Gamma**	**HR**	**Beta**	**Lambda**	**Gamma**	**HR**
No Bias	8e-4	-	-	0.1281	2e-5	-	-	0.0974
User Bias	6e-4	0.1	-	0.1336	4e-5	0.1	-	0.1012
Item Bias	2e-4	-	0.01	0.1401	4e-5	-	1e-4	0.1007
User & Item Bias	6e-4	0.1	1e-4	0.1090	4e-5	0.1	1e-4	0.0977

5.3.2 Performance of Induced Sparsity on S

Figure 5.1 shows FISM's performance on the sparsified **S** matrix. The x-axis represents the density of **S** after sparsifying the matrix as explained in Section 5.2.3. We can see that there is only a minimal reduction in the recommendation performance up to a density in the range 0.1 to 0.15. At the same time, the average time required to compute the recommendations for each user reduces drastically. This gain in recommendation efficiency comes at a very small cost in terms of recommendation performance, which may justify it's use in applications in which high-throughput recommendation rates are required.

5.3.3 Effect of Estimation Approach

To study the effect of FISM's estimation approach, which excludes the item's own rating during estimation, we compare the performance of FISM with an approach which is the same as FISM except that it includes the rating's own value during estimation. We call this method FISM(F), where F corresponds to similar approaches used in factorization (F) based schemes for rating prediction (NSVD and SVD++).

Keeping the rest of the parameters constant, the number of latent factors k is varied and the performance of FISM and FISM(F) is compared. Figure 5.4 shows the results for different datasets. We can see that, for smaller values of k, the performance of both the schemes is very similar. However, when the value of k is increased, FISM starts to perform better than FISM(F) and the gap between the performance of the methods increases as the value of k increases. This confirms the fact that the estimation approach

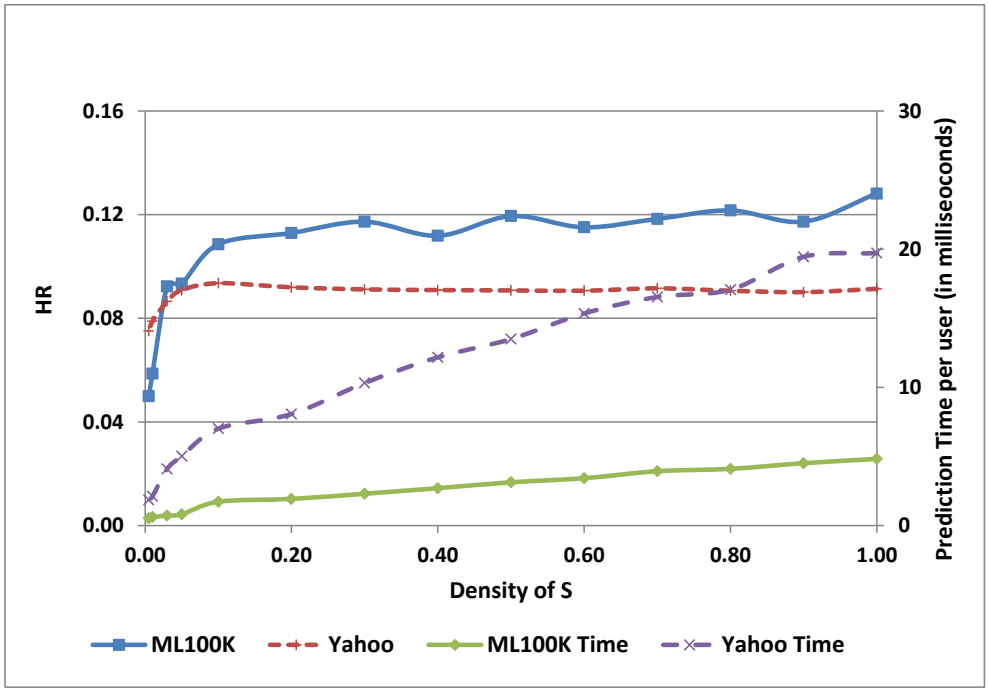

Figure 5.1: Performance of induced Sparsity on **S**.

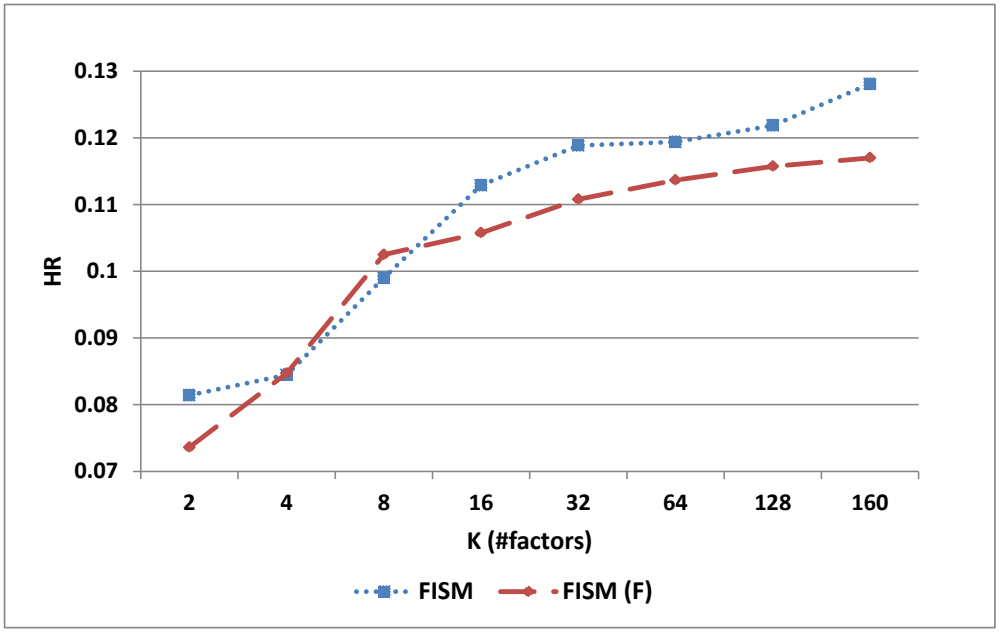

Figure 5.2: Effect of estimation approach on performance on ML100K dataset.

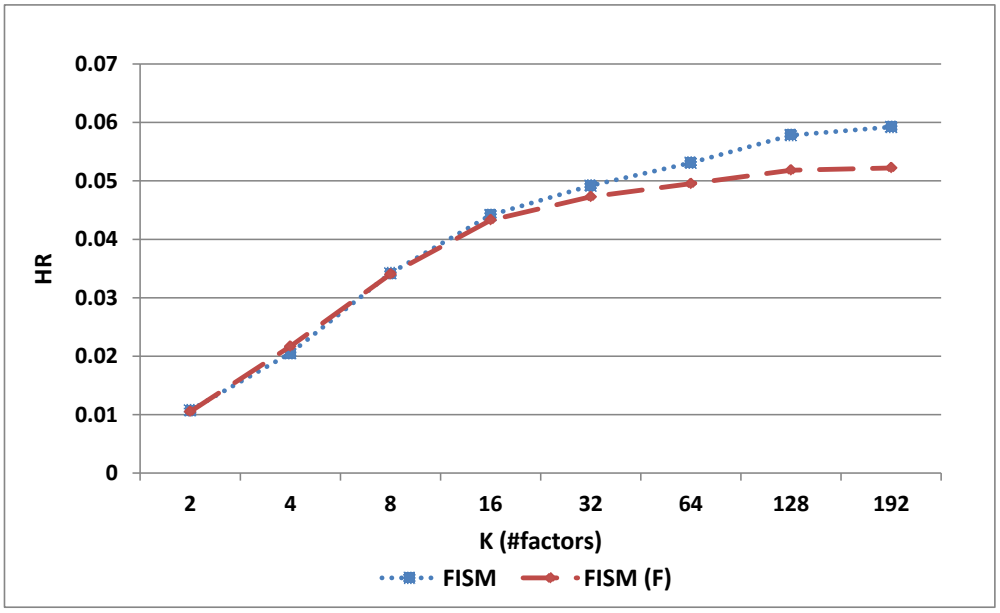

Figure 5.3: Effect of estimation approach on performance on Netflix dataset.

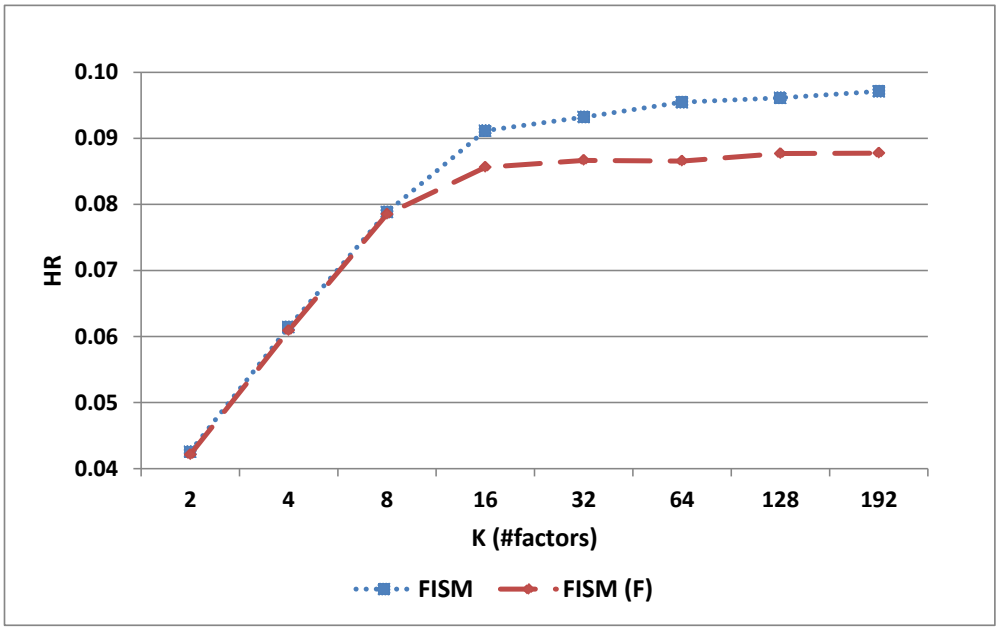

Figure 5.4: Effect of estimation approach on performance on Yahoo dataset.

used by FISM is superior to that used by approaches like NSVD and SVD++ and helps to avoid trivial solutions when the number of factors becomes large.

5.3.4 Effect of Non-Negativity

SLIM enforces non-negativity constraint to ensure that the learned item similarities correspond to positive relationships. SLIM has also shown that the adding such a constraint helps to improve the recommendation performance. In FISM, no such explicit constraint is enforced. We implemented the FISMrmse and FISMauc algorithms with non-negativity constraints and, to our surprise there was no improvement in the performance. In fact, the performance dropped considerably (HR of 0.0933 for ML100K and 0.0848 for Yahoo, compared to 0.1281 and 0.0974 without the constraints).

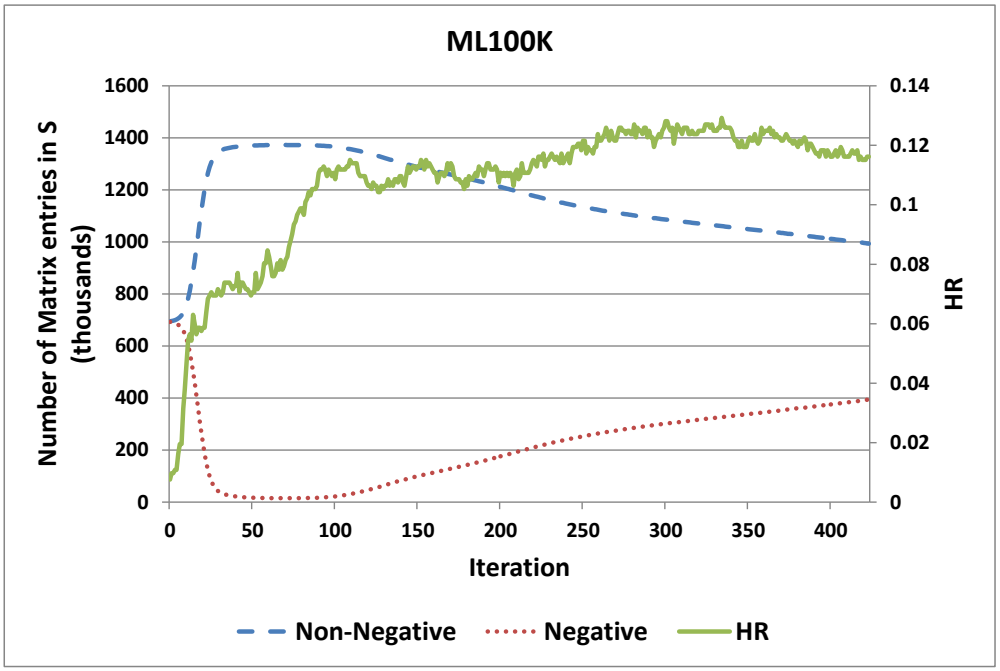

Figure 5.5: Non-negative and negative entries in $\mathbf{S}$.

To gain some insights on this issue, we observed the properties of $\mathbf{S} = \mathbf{P}\mathbf{Q}^{\top}$ during the learning process. In particular, we observed the number of negative and non-negative entries in $\mathbf{S}$ during each iteration of the learning process. The observations are plotted in Figure 5.5. We can see that initially the number of negative and non-negative entries

is similar, but as the model starts to learn, the number of negative entries decreases drastically. The best performance is obtained when the number of negative entries is significantly smaller compared to the number of non-negative entries (of the order 1:3). This shows that, even though the non-negativity constraint is not explicitly enforced in FISM, the model still learns the majority of the similarity values as non-negative entries.

5.3.5 Comparison With Other Approaches

Table 5.2, Table 5.3 and Table 5.4 shows the overall performance of FISM in comparison to other state-of-the-art algorithms (Section 4.4) for the top-N recommendation task.

Columns corresponding to "params" indicate the model parameters for the corresponding method. For ItemKNN (cos) and ItemKNN (log) methods, the parameter is the number of neighbors. For ItemKNN (cprob), the parameters are the number of neighbors and α. For PureSVD method, the parameter is the number of singular values used. For BPRkNN method, the parameters are the learning rate and λ. For BPRMF method, the parameters are the number of latent factors and the learning rate. For SLIM, the parameters are β and λ and for FISM the parameters are number of latent factors (k), regularization weight (β) and learning rate (η). The columns corresponding to HR and ARHR represent the hit rate and average reciprocal hit-rank metrics, respectively. Underlined numbers represent the best performing model measured in terms of HR for each dataset.

The results in Table 5.2,Table 5.3 and Table 5.4 show that FISM performs better than all the other methods across all the datasets. For many of these datasets, the improvements achieved by FISM against the next best performing schemes are quite substantial. In terms of the two loss functions, quite surprisingly, the RMSE loss (FISMrmse) achieved better performance than the AUC loss (FISMauc). This is contrary to the results reported by other studies and we are currently investigating it.

Note that for all the results presented so far, the number of top-N items chosen is 10 (i.e., $N = 10$). Figure 5.6 and Figure 5.7 shows the performance achieved by the various schemes for different values of N. These results are fairly consistent with those presented in Table 5.2, Table 5.3 and Table 5.4, with FISM performing the best.

To better illustrate the gains achieved by FISM over the other competing approaches as the sparsity of the datasets increases, Figure 5.8 shows the percentage improvement

Table 5.2: Comparison of performance of *top-N* recommendation algorithms with FISM for ML100K Dataset.

Method	**ML100K-1**				
	Params			HR	ARHR
ItemKNN (cos)	100	-	-	0.1604	0.0578
ItemKNN (log)	100	-	-	0.1047	0.0336
ItemKNN (cprob)	500	0.6	-	0.1711	0.0581
PureSVD	10	-	-	0.1700	0.0594
BPRkNN	1e-4	0.01	-	0.1621	0.0564
BPRMF	400	0.1	-	0.1610	0.0512
SLIM	0.1	20	-	0.1782	0.0620
FISMrmse	96	2e-5	0.001	0.1908	0.0641
FISMauc	64	0.001	1e-4	0.1518	0.0504
Method	**ML100K-2**				
	Params			HR	ARHR
ItemKNN (cos)	100	-	-	0.1214	0.0393
ItemKNN (log)	100	-	-	0.0809	0.0250
ItemKNN (cprob)	500	0.3	-	0.1308	0.0440
PureSVD	10	-	-	0.1362	0.0438
BPRkNN	1e-5	0.01	-	0.1272	0.0447
BPRMF	700	0.1	-	0.1224	0.0407
SLIM	0.01	18	-	0.1283	0.0448
FISMrmse	64	8e-4	0.01	0.1482	0.0462
FISMauc	144	2e-5	5e-5	0.1304	0.0424
Method	**ML100K-3**				
	Params			HR	ARHR
ItemKNN (cos)	100	-	-	0.0602	0.0193
ItemKNN (log)	100	-	-	0.0424	0.0116
ItemKNN (cprob)	400	0.1	-	0.0938	0.0293
PureSVD	5	-	-	0.0438	0.0316
BPRkNN	1e-5	14	-	0.1006	0.0319
BPRMF	700	0.25	-	0.0943	0.0305
SLIM	1e-4	14	-	0.0919	0.0303
FISMrmse	96	8e-4	0.001	0.1260	0.0384
FISMauc	144	8e-5	1e-5	0.1140	0.0340

Table 5.3: Comparison of performance of *top-N* recommendation algorithms with FISM for Netflix Dataset.

Method	**Netflix-1**				
	Params			HR	ARHR
ItemKNN (cos)	100	-	-	0.1516	0.0689
ItemKNN (log)	100	-	-	0.0630	0.0240
ItemKNN (cprob)	20	0.5	-	0.1555	0.0678
PureSVD	600	-	-	0.1783	0.0865
BPRkNN	1e-3	1e-4	-	0.1678	0.0781
BPRMF	800	0.1	-	0.1638	0.0719
SLIM	1e-3	8	-	0.2025	0.1008
FISMrmse	192	2e-5	0.001	0.2118	0.1107
FISMauc	192	1e-5	1e-4	0.2095	0.1016
Method	**Netflix-2**				
	Params			HR	ARHR
ItemKNN (cos)	100	-	-	0.0849	0.0316
ItemKNN (log)	100	-	-	0.0838	0.0303
ItemKNN (cprob)	500	0.5	-	0.0879	0.0326
PureSVD	400	-	-	0.0807	0.0297
BPRkNN	1e-4	1	-	0.0889	0.0329
BPRMF	700	0.1	-	0.0862	0.0318
SLIM	0.1	8	-	0.0947	0.0374
FISMrmse	192	6e-5	0.001	0.1041	0.0386
FISMauc	240	2e-5	1e-4	0.0979	0.0341
Method	**Netflix-3**				
	Params			HR	ARHR
ItemKNN (cos)	100	-	-	0.0374	0.0123
ItemKNN (log)	100	-	-	0.0188	0.0062
ItemKNN (cprob)	200	0.1	-	0.0461	0.0162
PureSVD	400	-	-	0.0382	0.0131
BPRkNN	0.01	1e-3	-	0.0439	0.0148
BPRMF	5	0.01	-	0.0454	0.0153
SLIM	1e-4	12	-	0.0422	0.0149
FISMrmse	128	6e-5	0.001	0.0578	0.0185
FISMauc	160	4e-4	5e-4	0.0548	0.0177

Table 5.4: Comparison of performance of *top-N* recommendation algorithms with FISM for Yahoo Dataset.

Method	**Yahoo-1**				
		Params		HR	ARHR
ItemKNN (cos)	100	-	-	0.1344	0.0502
ItemKNN (log)	100	-	-	0.1046	0.0358
ItemKNN (cprob)	500	0.6	-	0.1387	0.0510
PureSVD	50	-	-	0.1229	0.0459
BPRkNN	1e-3	1e-4	-	0.1432	0.0528
BPRMF	700	0.1	-	0.1337	0.0473
SLIM	0.1	12	-	0.1454	0.0542
FISMrmse	192	1e-4	0.001	0.1522	0.0542
FISMauc	144	8e-5	1e-4	0.1426	0.0488
Method	**Yahoo-2**				
		Params		HR	ARHR
ItemKNN (cos)	100	-	-	0.0890	0.0295
ItemKNN (log)	100	-	-	0.0820	0.0261
ItemKNN (cprob)	200	0.4	-	0.0908	0.0313
PureSVD	20	-	-	0.0769	0.0257
BPRkNN	1e-3	1e-4	-	0.0894	0.0304
BPRMF	700	0.1	-	0.0869	0.0288
SLIM	1e-3	12	-	0.0904	0.0304
FISMrmse	192	2e-5	5e-4	0.0971	0.0371
FISMauc	160	2e-5	5e-4	0.0974	0.0315
Method	**Yahoo-3**				
		Params		HR	ARHR
ItemKNN (cos)	100	-	-	0.0366	0.0116
ItemKNN (log)	100	-	-	0.0489	0.0153
ItemKNN (cprob)	20	0.1	-	0.0571	0.0187
PureSVD	20	-	-	0.0494	0.0154
BPRkNN	0.1	0.01	-	0.0549	0.0183
BPRMF	10	0.01	-	0.0530	0.0169
SLIM	0.1	2	-	0.0491	0.0159
FISMrmse	160	0.002	0.001	0.0740	0.0230
FISMauc	176	2e-4	0.001	0.0722	0.0228

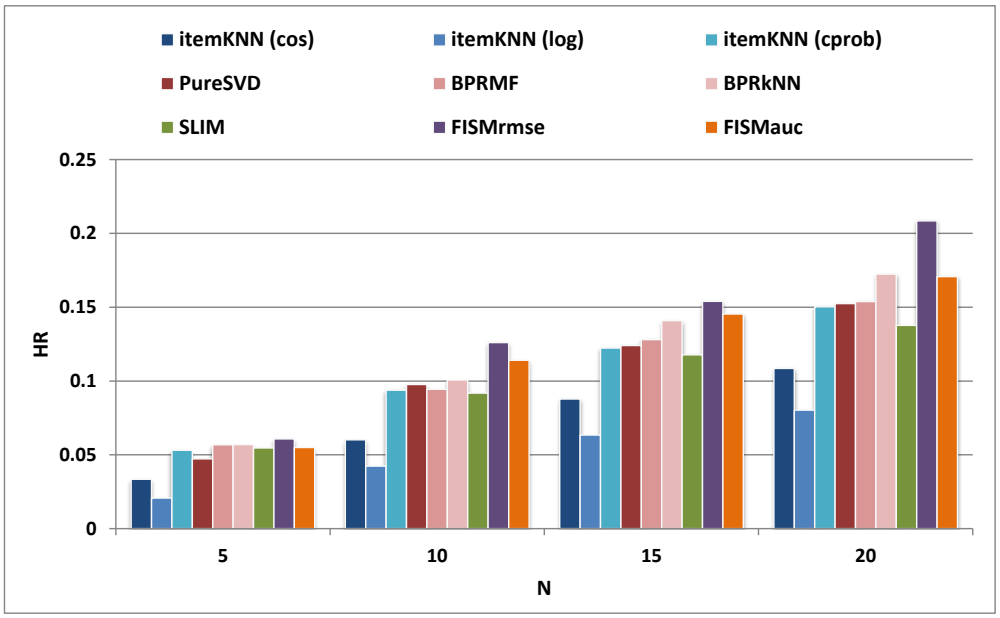

Figure 5.6: Performance for different values of N for ML100K dataset.

achieved by FISM against the next best performing scheme for each dataset across the three sparsity levels. These results show that, as the datasets become sparser, the relative performance of FISM (in terms of HR) increases and, on the sparsest datasets, outperforms the next best scheme by at least 24%.

5.4 Conclusion

In this chapter, we presented a factored item similarity based method (FISM) for the top-N recommendation problem. FISM learns the item similarities as the product of two matrices, allowing it to generate high quality recommendations even on sparse datasets. The factored representation is estimated using a structural equation modeling approach, which leads to better estimators as the number of factors increases. We conducted a comprehensive set of experiments on multiple datasets at different sparsity levels and compared FISM's performance against that of other state-of-the-art top-N recommendation algorithms. The results showed that FISM outperforms the rest of the methods and the performance gaps increases as the datasets become sparser. For faster recommendation, we showed that sparsity can be induced in the resulting item

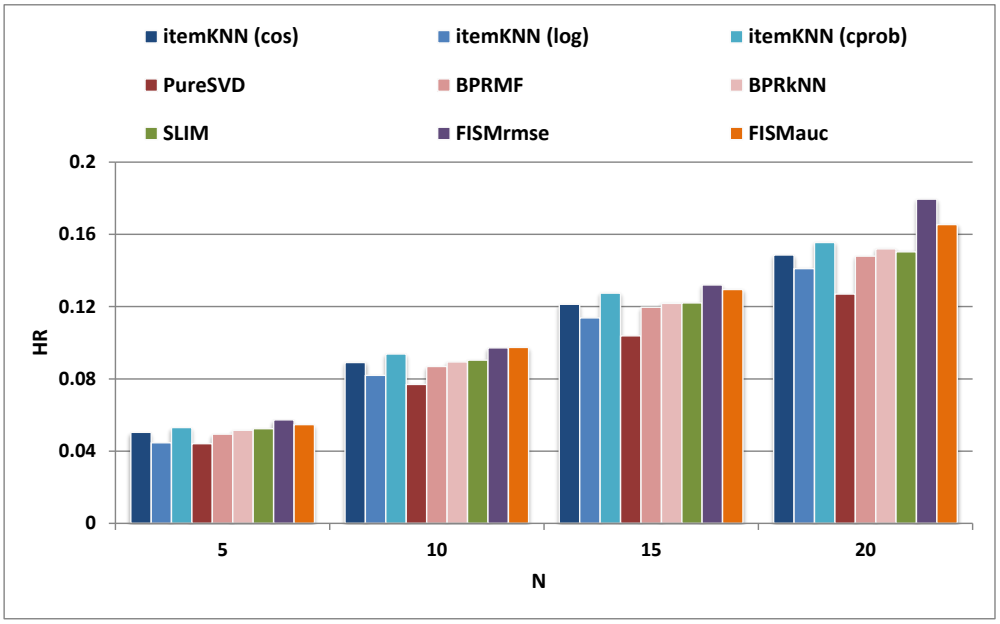

Figure 5.7: Performance for different values of N for Yahoo dataset.

similarity matrix with minimal reduction in the recommendation quality.

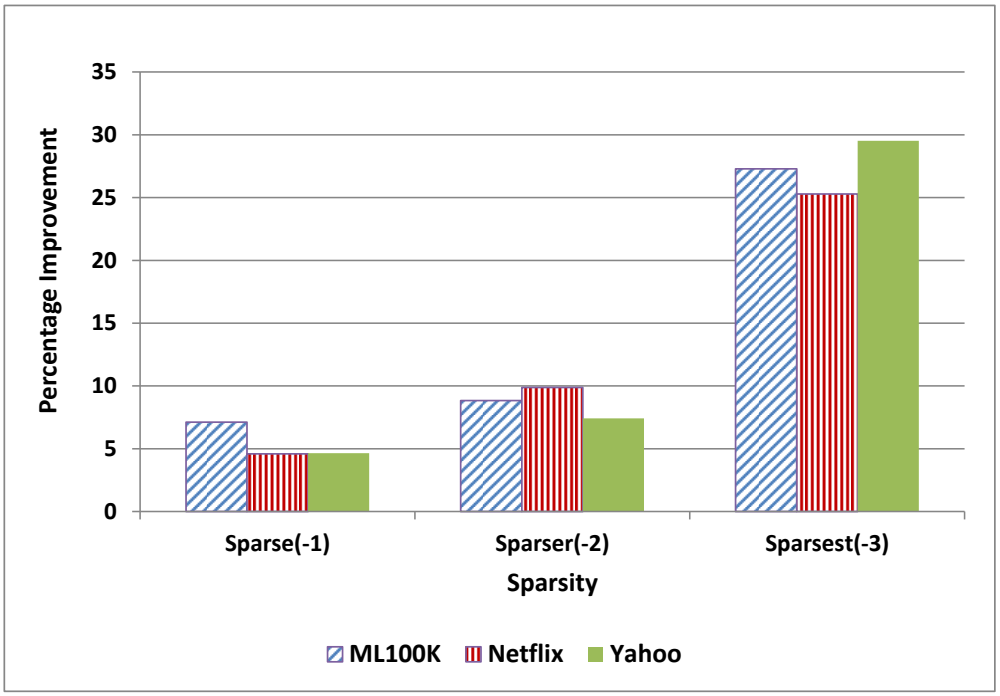

Figure 5.8: Effect of sparsity on performance for various datasets.

Chapter 6

MPCF: Modeling Global and Local Preferences of Users in Collaborative Filtering

This chapter focuses on developing a new method called MPCF (NonLinear Matrix Factorization), which better model the users with multiple preferences. MPCF models the user as a combination of global and interest-specific preference components. This modeling approach helps to capture both of user's high level preferences and finer preferences in terms of different priorities that the user might have and the tradeoffs that the user is willing to make, corresponding to the different features of the items. An additional advantage of MPCF is that, the user's global preferences are estimated by taking into account all the observations, which as our experimental results will show, allows MPCF to perform well on sparser datasets.

6.1 Introduction

Consider an example of modeling user preferences for restaurants. Restaurants can be represented using various features such as ambience, location, family-friendliness, cuisine, food choices, freshness, spice level etc. Users can have different preferences for these different features of restaurants and the food they serve. For example, a user

might like restaurants which are family-friendly and serve chinese food, or which are family-friendly and serve Indian food with low spice levels. These different preferences can be captured by representing users with multiple preference vectors, with the vectors corresponding to each of the preferences. The weights on these preference vectors corresponding to the features that the user likes, will have a higher value. Methods like MaxMF do a good job in capturing these different preferences across different item features. However, the item features for which the users have multiple preferences are not necessarily disjoint and can have many common features, for which the users have similar preferences. For example, a user with a family would most probably have family-friendliness as a common feature preference for restaurants across his different preference vectors. Similarly, a vegetarian user, will mostly have vegetarian food choices and freshness of the veggies as a common feature preference. Hence, even though users at a high level have different preferences corresponding to restaurant features, at a finer level, they can have similar preferences for many features which are common across the different set of preferences. Thus, there is a need to better model the users multiple preferences to capture the finer priorities associated with different subsets of item features. We discuss two different user behaviors in detail which depicts the need to better model the users multiple preferences.

Continuing with the restaurants example, in the first user behavior, the users may feel strong about some of the these restaurant features and may not be willing to make any tradeoffs on them. For the rest of the features, based on the different preferences they have, the users might be ready to tradeoff certain subset of features for a different subset of features. For example, users might have strong preference and not willing to make any tradeoffs w.r.t. features like family-friendliness, location and freshness of the food, while they are ready to make a tradeoff on other features like ambience, food choices and the type of cuisine. That is, in this case, the features of restaurants are partitioned into two disjoint sets, one corresponds to the set of features for which the users have a strong common preference across the different preference vectors. These feature preferences are common across multiple preferences that they have. The second set contains feature preferences for which the users are willing to make a tradeoff.

In the second user behavior, the users at a high level can have certain preferences for all the features of restaurants. These preferences represent the common overall

preferences for all the restaurant features. While at a finer level they might be ready to tradeoff preferences for certain features in favor of preferences for other features. For example, consider a user who in general prefers a Indian restaurant which serves spicy curry and is located close to his/her house. Depending on the different restaurant choices available, the same user might be ready to tradeoff the location of the restaurant for a restaurant which serves a tasty vegetarian curry of his/her choice or tradeoff the Indian cuisine type for a thai restaurant which serves his/her favorite curry. Thus, in this case, the features of restaurants are not partitioned and the users have an overall preference on all the features and they also have a set of different preferences each representing a tradeoff on the different features of the restaurants.

In this thesis chapter, we propose a method called MPCF that is designed to model the user to effectively capture both the high level common preferences and the finer preferences on the item features. In MPCF, each user is modeled as a combination of a global preference component and a local preferences component. For the first behavior case, the global preference component is used to model the common strong preferences of the user and the local preferences component is used to model the different user preferences corresponding to the different set of item features for which the users are willing to make a tradeoff. The global preference component consists of a single user preference vector and the local preferences component consists of a set of user preference vectors, each one corresponding to the different local preferences. For the second user behavior case, the global preference component of MPCF models the basic overall preferences of the user on all the item features and the local preferences component models the different tradeoffs the users are willing to make on the item features corresponding to the different preferences they have.

Our discussion so far was motivated using restaurants as an example; however, we believe that the users exhibit a similar behavior in other domains as well. For example, consider the problem of modeling user preferences for watching movies. Users in this case can have different preferences for different features of movies like language, genre, rating, actors, running time etc. In the first case, users can have strong preferences w.r.t. features like language, genre and rating, while they are willing to make a tradeoff on other features like running time and actors. And in the second case, users can have a general preference for watching action or thriller movies in English, while at a finer

level they are ready to tradeoff certain features like English language for a critically accliamed Spanish action movie with english subtitles or tradeoff the action genre for a drama movie featuring his/her favorite actors which is two hours or less in running time.

Methods like MaxMF, model the user's different preferences only at the higher level, i.e., they represent the user with multiple latent vectors, each corresponding to his/her different preferences. Potentially MaxMF can include these finer preferences as part of each of the multiple latent dimensions, but it becomes computationally expensive to learn the corresponding multiple latent vectors for all the users. Thus, there is a need to model the users in a way which can explicitly capture these finer preferences, thereby helping in providing better recommendations.

The key contributions of the work presented in this chapter are the following:

(i) Proposes a new method to model user's multiple preferences as a combination of user global preference and multiple local preference components. In addition, the proposed method can handle sparse data effectively.

(ii) Models the user bias at the local preference level to better capture the user biases at a finer granularity.

(iii) Proposes two different approaches to model the users based on shared and independent item features between the global preference and local preferences components.

(iv) Compares the performance of the proposed model with other state-of-the-art methods in the context of rating prediction and top-N recommendation tasks, and investigates the impact of various parameters as they relate to number of local preferences, biases and data sparsity.

Figure 6.1 illustrates how the users preferences are modeled in the proposed MPCF method.

6.2 MPCF - Nonlinear Methods for CF

In MPCF, given a user u, item i and T user local preferences, the estimated rating $\hat{r}_{ui}$ is given by the sum of the estimations from global preference and local preference

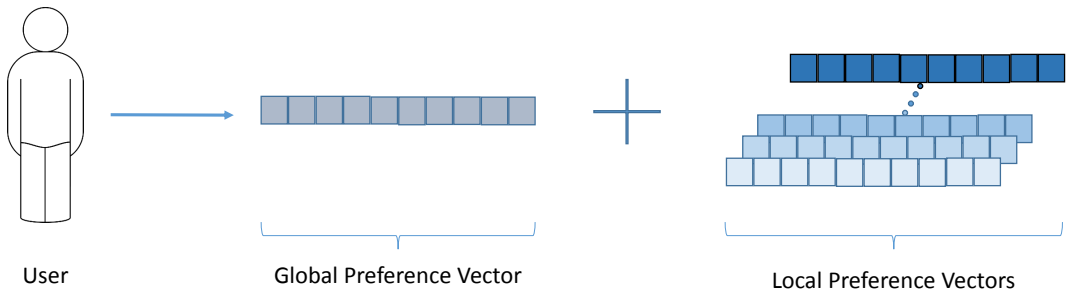

Figure 6.1: Modeling user preferences in MPCF.

components. That is,

$$\hat{r}_{ui} = \mu + b_i + \mathbf{p}_u \mathbf{q}_i^{\mathsf{T}} + \max_{t=1,\ldots,T}(b_{ut} + f(u,i,t)), \tag{6.1}$$

where μ is the global bias i.e., the average rating value of the entire data set, b_i is the item bias corresponding to item i, b_{ut} is the user-local preference bias corresponding to user u and local preference t, $\mathbf{p}_u$ is the latent vector associated with user u and $\mathbf{q}_i$ is the latent vector associated with item i. Thus, $\mathbf{p}_u \mathbf{q}_i^{\mathsf{T}}$ gives the prediction score from global preference component of the model and $b_{ut} + f(u,i,t)$ is the prediction score from local preferences component corresponding to the local preference t. The final prediction score is the sum of the predictions from global preference and local preferences components. Note that for each user, we model user biases at local preference level. This is added to capture the user bias for each local preference separately. However, we can also add global preference user bias. But our experimental analysis indicated that having user biases only for local preferences helps to achieve the best performance. Thus, we restricted the model to having only the local preference specific user biases.

The selection of the best local preference t^* is done by choosing the local preference which results in the maximum score from the local preferences component. The max function is used to compute the maximum recommendation score for the item amongst all the local preferences of the user in the local preferences function. For top-N recommendation task, the intuition behind this idea is that, for an item to be ranked higher in the top-N list of the user, at least one of the local preferences of the user must provide a high score for that item. In case of rating prediction task, the intuition is that, for a high rated item at least one of the local preferences must provide a high score for that item and for a low rated item, none of the local preferences should provide a high score

for that item.

An added advantage of modeling the users as a combination of global and multiple local preferences is the ability to deal with sparse data. Users typically provide ratings to only a handful of items out of possible thousands or millions of items. Due to limited preferences given out by users, the user-item rating matrix becomes sparse. Methods like MaxMF, models users with multiple preference vectors by implicitly partitioning the items rated by the user into multiple subsets and learning a separate user latent preference vector for each partition. This representation does not differentiate the global preference and local user preferences. If the users provide sufficient preference data, then the user's global preference can be potentially estimated as part of the local preference. However, as explained above, most of the real world datasets are sparse. In such cases, when the users have not provided sufficient ratings, it becomes hard to capture the global preference as part of local preferences. Another potential issue with modeling the users with only the local preferences component is that, in case of limited availability of preference data, there is lesser support (in terms of number of rated items) for each local preference. This can potentially affect the learning process and result in learning less meaningful (latent) factors for all the item partitions (local preferences) corresponding to that user. Thus, in case of sparse data, it is beneficial to model the global preference separately and estimate it explicitly. This makes the proposed method MPCF more robust in handling sparse data. By adding regularization, MPCF allows the model to be flexible, i.e., it implicitly allows the learning process to strike a balance between the two components. This helps to effectively capture the user's global and local preferences, including the ones who have not provided sufficient ratings corresponding to each of the local preferences.

Corresponding to the two different user behaviors cases discussed before, we propose two different MPCF methods. In the first approach named as MPCFi (*Independent Features Model*), the global component models the user's strong preferences corresponding to certain features of items, while the local preferences component models the preferences on item features which the user is willing to tradeoff. In the second approach named as MPCFs (*Shared Features Model*), the global component models the user's basic preferences for all the features of the items and the local preferences component helps to fine tune these preferences based on the item features for which the user is

willing to tradeoff.

6.2.1 MPCFi - Independent Features Model

MPCFi is designed to model user's strong preferences across the different item features in the global preference component and the preferences which are applicable only to specific item features are modeled as part of local preferences component. This is achieved by having independent item factors in local preferences component $f(u, i, t)$, compared to that of global preference component. Thus, for independent features model (MPCFi), the local preferences component $f(u, i, t)$ is given by,

$$f(u, i, t) = \mathbf{w}_{ut}\mathbf{y}_i^{\mathsf{T}}, \tag{6.2}$$

where $\mathbf{w}_{ut}$ is the user latent vector for u in the local preferences component corresponding to the local preference t and $\mathbf{y}_i$ is the item latent vector in the local preferences component. We can see that, for a given item i, MPCFi has two independent item factors ($\mathbf{q}_i$ and $\mathbf{y}_i$), each one corresponding to the global preference and local preferences components.

The recommendation score $\hat{r}_{ui}$ for a given user u and item i is computed as,

$$\hat{r}_{ui} = \mu + b_i + \mathbf{p}_u\mathbf{q}_i^{\mathsf{T}} + \max_{t=1,\ldots,T}(b_{ut} + \mathbf{w}_{ut}\mathbf{y}_i^{\mathsf{T}}) \tag{6.3}$$

where $\mathbf{p}_u$ and $\mathbf{q}_i$ are the user and item latent vectors in the global preference component, respectively. Thus, MPCFi is an additive model which independently learns two non-overlapping models corresponding to global preference and local preferences components and computes their sum as the final prediction score.

Note that the number of latent factors for the global preference component (i.e., $\mathbf{p}_u\mathbf{q}_i^{\mathsf{T}}$) and the local preferences component (i.e., $\mathbf{w}_{ut}\mathbf{y}_i^{\mathsf{T}}$) need not be the same. Thus, this model has the flexibility of having different number of latent factors for the two components. We use k to represent the number of latent factors for the global preference component (i.e., $\mathbf{p}_u, \mathbf{q}_i \in \mathbb{R}^{1\times k}$) and we use l to represent the number of latent factors for the local preferences component (i.e., $\mathbf{w}_{ut}, \mathbf{y}_i \in \mathbb{R}^{1\times l}$).

6.2.2 MPCFs - Shared Features Model

MPCFs is designed to model the user's basic overall preferences corresponding to all the features of items as part of the global component and the tradeoffs that the user is willing to take w.r.t. certain features are modeled as part of the local preferences component. This is represented in the model by sharing the item factors in the local preferences component $f(u, i, t)$ with the global preference component. Thus, the local preferences component $f(u, i, t)$ for MPCFs is given by,

$$f(u, i, t) = \mathbf{w}_{ut}\mathbf{q}_i^{\mathsf{T}}, \tag{6.4}$$

$\mathbf{w}_{ut}$ is the user latent vector for u in the local preferences component corresponding to the local preference t and $\mathbf{q}_i$ is the shared item latent vector between the global preference and the local preferences components. By using the shared item latent vectors, this model has the ability to transfer the learning between the two components. Contrast this with MPCFi model, which has independent item factors ($\mathbf{q}_i$ and $\mathbf{y}_i$) for both global preference and local preferences components.

The recommendation score $\hat{r}_{ui}$ for a given user u and item i is computed as,

$$\tilde{r}_{ui} = \mu + b_i + \mathbf{p}_u\mathbf{q}_i^{\mathsf{T}} + \max_{t=1,\ldots,T}(b_{ut} + \mathbf{w}_{ut}\mathbf{q}_i^{\mathsf{T}}) \tag{6.5}$$

where the different variables mean the same as in Equation 6.3.

6.2.3 Model Estimation

To estimate the model parameters of MPCF, we use squared error loss function to compute and minimize the loss. Following regularized optimization problem is minimized to learn the $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{W}$ and $\mathbf{Y}$ matrices:

$$\underset{\mathbf{P},\mathbf{Q},\mathbf{W},\mathbf{Y}}{\text{minimize}} \ \frac{1}{2}\sum_{u,i\in R}\|r_{ui} - \hat{r}_{ui}\|_F^2 + \frac{\lambda}{2}(\|\mathbf{P}\|_F^2 + \|\mathbf{Q}\|_F^2 + \|\mathbf{W}\|_F^2 + \|\mathbf{Y}\|_F^2 + \|\mathbf{B_u}\|_F^2 + \|\mathbf{b_i}\|_2^2), \tag{6.6}$$

where r_{ui} is the ground truth value, $\hat{r}_{ui}$ is the estimated value, $\mathbf{b_i}$ is the vector corresponding to the item biases, $\mathbf{B_u}$ is the matrix corresponding to the local preference user biases and λ is the l_2-regularization constant for latent factor matrices. l_2 regularization is used to prevent overfitting.

The optimization problem in Equation 6.6 is solved using a Stochastic Gradient Descent (SGD) algorithm [80]. Algorithm 6 and Algorithm 7 provides the detailed procedure for learning the model parameters of MPCFi and MPCFs respectively. Initially the vector $\mathbf{b_i}$, matrices $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{Y}$ and $\mathbf{B_u}$, and tensor $\mathbf{W}$ are initialized with small random values as the initial estimate. Then, in each iteration the parameter values are updated based on the gradients computed w.r.t. the parameter being updated. This process is repeated until the error on validation set does not decrease further or the number of iterations has reached a predefined threshold. The main difference in the algorithms for minimizing objective functions of MPCFi and MPCFs is in the computation of $\hat{r}_{ui}$ and the corresponding gradient update rules for the parameters involved in the model. In case of MPCFi, the matrix $\mathbf{Y}$ is learnt along with $\mathbf{P}$, $\mathbf{Q}$ and $\mathbf{W}$, whereas in MPCFs, only $\mathbf{P}$, $\mathbf{Q}$ and $\mathbf{W}$ are involved.

For the top-N recommendation task, the gradient updates for model parameters are computed for both rated and non-rated entries of $\mathbf{R}$. This is in accordance with the common practice followed for the top-N recommendation task [28, 46, 1]. This is in contrast with the rating prediction task, where only the rated items are typically used for computing gradient updates. In order to reduce the computational complexity of the learning process, the zero entries corresponding to non-rated items are sampled and used along with all the non-zero entries (corresponding to rated items) of $\mathbf{R}$. Given a sampling constant ρ and $nnz(\mathbf{R})$, the number of non-zeros in $\mathbf{R}$, $\rho \cdot nnz(\mathbf{R})$ zeros are sampled and used for optimization in each iteration of the learning algorithm. Our experimental results indicate that a small value of ρ (in the range $3 - 5$ is sufficient to produce the best model. This sampling strategy makes MPCF methods computationally efficient and scalable.

The gradient computations and updates for SGD can be parallelized. Hence, these algorithms can be efficiently applied to larger datasets. In [76], a distributed SGD is proposed. A similar algorithm with modifications can be used to scale the MPCF methods to larger datasets. Software packages like Spark[1] can be used to execute SGD based algorithms on a large cluster of processing nodes.

Note that for top-N recommendation task, instead of squared loss function, a ranking based loss function [53, 1] can be used. We plan to investigate this as part of the future

[1] http://spark.apache.org/

Algorithm 6 MPCFi:Learn.

procedure MPCFi _LEARN
  $\eta \leftarrow$ learning rate
  $\lambda \leftarrow \ell_F$ regularization weight
  $\rho \leftarrow$ sample factor
  $\mu \leftarrow$ average training data rating
  $iter \leftarrow 0$
  Init $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{W}$, $\mathbf{Y}$, $\mathbf{b_i}$ and $\mathbf{B_u}$ with random values in (-0.001, 0.001)

  while $iter < maxIter$ or error on validation set decreases **do**
    $\mathcal{R}' \leftarrow \mathbf{R} \cup$ *SampleZeros(*$\mathbf{R}$*,* ρ*)* //For top-N recommendation task
    $\mathcal{R}' \leftarrow \mathbf{R}$ //For rating prediction task
    $\mathcal{R}' \leftarrow$ *RandomShuffle(*$\mathcal{R}'$*)*

    for all $r_{ui} \in \mathcal{R}'$ **do**
      $\hat{r}_{ui} \leftarrow \mu + b_i + \mathbf{p}_u \mathbf{q}_i^\mathsf{T} + \max_{t=1,\ldots,T}(b_{ut} + \mathbf{w}_{ut}\mathbf{y}_i^\mathsf{T})$

      $t^* \leftarrow$ local preference corresponding to max score
      $e_{ui} \leftarrow r_{ui} - \hat{r}_{ui}$
      $b_i \leftarrow b_i + \eta \cdot (e_{ui} - \lambda \cdot b_i)$
      $b_{ut^*} \leftarrow b_{ut^*} + \eta \cdot (e_{ui} - \lambda \cdot b_{ut^*})$
      $\mathbf{p}_u \leftarrow \mathbf{p}_u + \eta \cdot (e_{ui} \cdot \mathbf{q}_i - \lambda \cdot \mathbf{p}_u)$
      $\mathbf{q}_i \leftarrow \mathbf{q}_i + \eta \cdot (e_{ui} \cdot \mathbf{p}_u - \lambda \cdot \mathbf{q}_i)$
      $\mathbf{w}_{ut^*} \leftarrow \mathbf{w}_{ut^*} + \eta \cdot (e_{ui} \cdot \mathbf{y}_i - \lambda \cdot \mathbf{w}_{ut^*})$
      $\mathbf{y}_i \leftarrow \mathbf{y}_i + \eta \cdot (e_{ui} \cdot \mathbf{w}_{ut} - \lambda \cdot \mathbf{y}_i)$
    end for

    $iter \leftarrow iter + 1$
  end while

  return $b_i, \mathbf{B_u}, \mathbf{P}, \mathbf{Q}, \mathbf{W}, \mathbf{Y}$
end procedure

Algorithm 7 MPCFs:Learn.

procedure MPCFs _LEARN
  $\eta \leftarrow$ learning rate
  $\lambda \leftarrow \ell_F$ regularization weight
  $\rho \leftarrow$ sample factor
  $\mu \leftarrow$ average training data rating
  $iter \leftarrow 0$
  Init $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{W}$, $\mathbf{b_i}$ and $\mathbf{B_u}$ with random values in (-0.001, 0.001)

  while $iter < maxIter$ or error on validation set decreases **do**
    $\mathcal{R}' \leftarrow \mathbf{R} \cup$ *SampleZeros(*$\mathbf{R}$*,* ρ*)* //For top-N recommendation task
    $\mathcal{R}' \leftarrow \mathbf{R}$ //For rating prediction task
    $\mathcal{R}' \leftarrow$ *RandomShuffle(*$\mathcal{R}'$*)*

    for all $r_{ui} \in \mathcal{R}'$ **do**
      $\hat{r}_{ui} \leftarrow \mu + b_i + \mathbf{p}_u \mathbf{q}_i^\mathsf{T} + \max_{t=1,\ldots,T} (b_{ut} + \mathbf{w}_{ut} \mathbf{q}_i^\mathsf{T})$

      $t^* \leftarrow$ local preference corresponding to max score
      $e_{ui} \leftarrow r_{ui} - \hat{r}_{ui}$
      $b_i \leftarrow b_i + \eta \cdot (e_{ui} - \lambda \cdot b_i)$
      $b_{ut^*} \leftarrow b_{ut^*} + \eta \cdot (e_{ui} - \lambda \cdot b_{ut^*})$
      $\mathbf{p}_u \leftarrow \mathbf{p}_u + \eta \cdot (e_{ui} \cdot \mathbf{q}_i - \lambda \cdot \mathbf{p}_u)$
      $\mathbf{q}_i \leftarrow \mathbf{q}_i + \eta \cdot (e_{ui} \cdot (\mathbf{p}_u + \mathbf{w}_{ut^*}) - \lambda \cdot \mathbf{q}_i)$
      $\mathbf{w}_{ut^*} \leftarrow \mathbf{w}_{ut^*} + \eta \cdot (e_{ui} \cdot \mathbf{q}_i - \lambda \cdot \mathbf{w}_{ut^*})$
    end for

    $iter \leftarrow iter + 1$
  end while

  return $b_i, \mathbf{B_u}, \mathbf{P}, \mathbf{Q}, \mathbf{W}$
end procedure

research.

6.3 Results

The experimental evaluation consists of four parts. First, we assess the effect of model parameters in terms of number of interests and biases on the recommendation performance. Second, we present the MPCF recommendation performance comparison with the MaxMF method, which is also a non-linear method based on modeling user with multiple interests. We present these studies only for a Netflix dataset, however, the same trend in results and conclusions carry over to the rest of the datasets as well. In the third part of the results, we present the comparison with other competing state-of-the-art methods (Section 4.4). Finally, we present the effect of data sparsity on the performance of the MPCF methods compared to the MaxMF.

6.3.1 Effect of Number of Local Preferences

In this study, we compare the effect of the number of local preferences (T) on the recommendation performance. Figures 6.2 and 6.3 show the results for both the MPCFs and MPCFi models for the rating prediction and the top-N recommendation tasks in terms of HR and RMSE, respectively. The value of k is kept constant at 128. For MPCFi model, we have set l to be same as k. We can see that in both models, the performance initially increases (i.e., HR increases and RMSE decreases) with increasing T and reaches a peak value when $T = 2$ for the rating prediction and $T = 3$ or $T = 4$ for the top-N recommendation. This indicates that, modeling the users with multiple local preferences provides the best recommendation performance. As we further increase the value of T, the performance starts to decrease (i.e., HR decreases and RMSE increases). This is possibly due to the fact that the support for each local preference in terms of number of items decreases; thus, leading to learning less meaningful user preferences for different local preferences.

6.3.2 Effect of Bias

In this study, we compare the effect of the various bias terms i.e., global bias(μ), user-local preference bias(b_{ut}) and item bias(b_i) on the rating prediction performance. Note

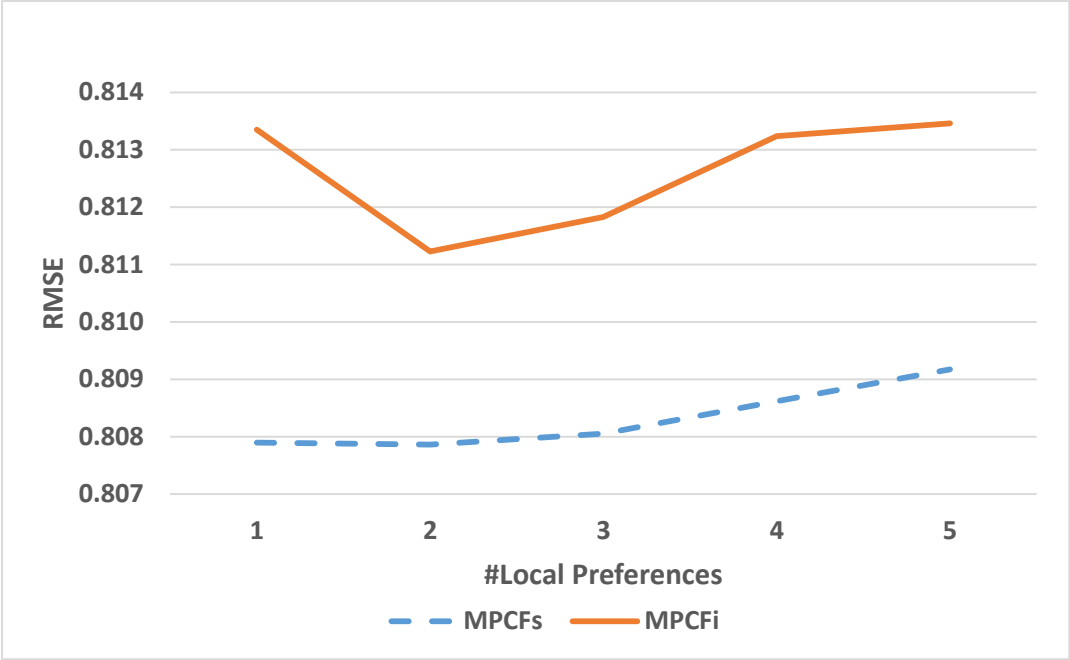

Figure 6.2: Effect of Number of Local Preferences for Rating Prediction.

that the use of these bias terms did not improve the performance in the case of the top-N recommendation task. Thus, we present the effect of bias terms only in the context of rating prediction task. The results of this study are presented in Table 6.1. We fixed the value of k to 128 and varied the bias terms. We can see that adding the item and user-local preference bias has positive impact on the performance and the addition of both item and user-local preference biases performs the best.

Table 6.1: Effect of Bias.

Bias	MPCFs	MPCFi
No Bias	0.8103	0.8109
Only Item Bias	0.8097	0.8099
Only User-Local Preference Bias	0.8096	0.8098
Item Bias + User-Local Preference Bias	0.8080	0.8097

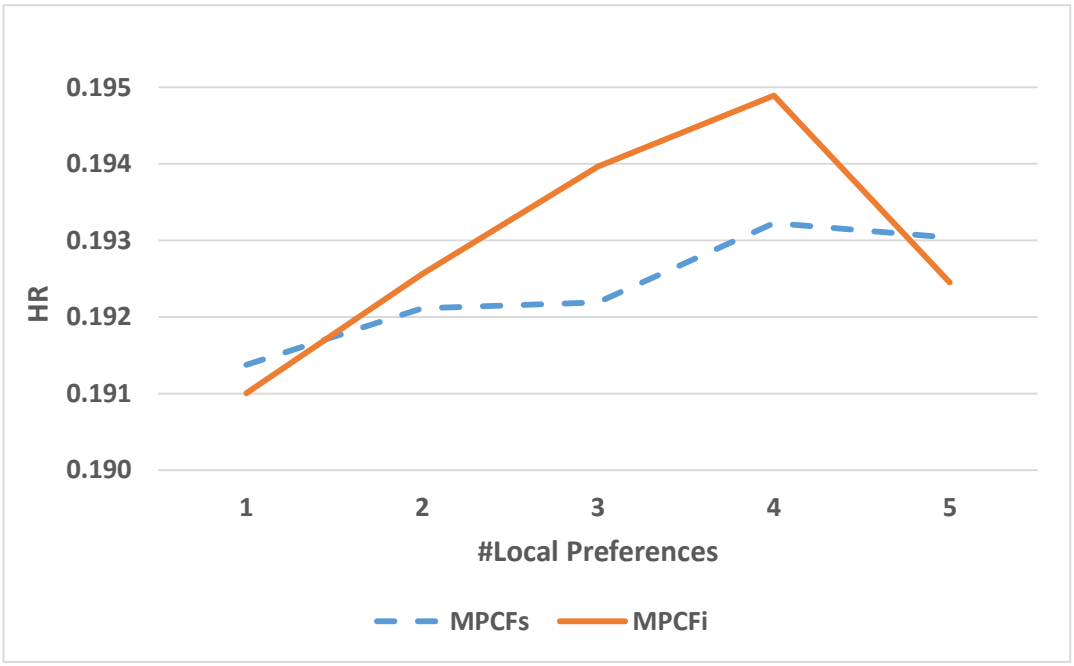

Figure 6.3: Effect of Number of Local Preferences for Top-N.

6.3.3 Comparison with MaxMF

In this study, we compare the performance of the MPCF methods with the MaxMF for different number of latent factors (k). For MPCFi, we pick the performance corresponding to the best l, for a given value of k. The results of this study are presented in Figures 6.4 and 6.5 for the rating prediction and the top-N recommendation tasks, respectively. We can see that, both of the MPCF methods outperform the MaxMF for different values of k for both the tasks. This result indicates the superiority of the MPCF methods over the MaxMF in better estimating the user preferences and thus achieving better recommendation performance.

6.3.4 Comparison with Other Approaches

Rating Prediction Task

The rating prediction performance of the MPCF methods in terms of RMSE in comparison to the other state-of-the-art methods (Section 4.4) is presented in Table 6.2.

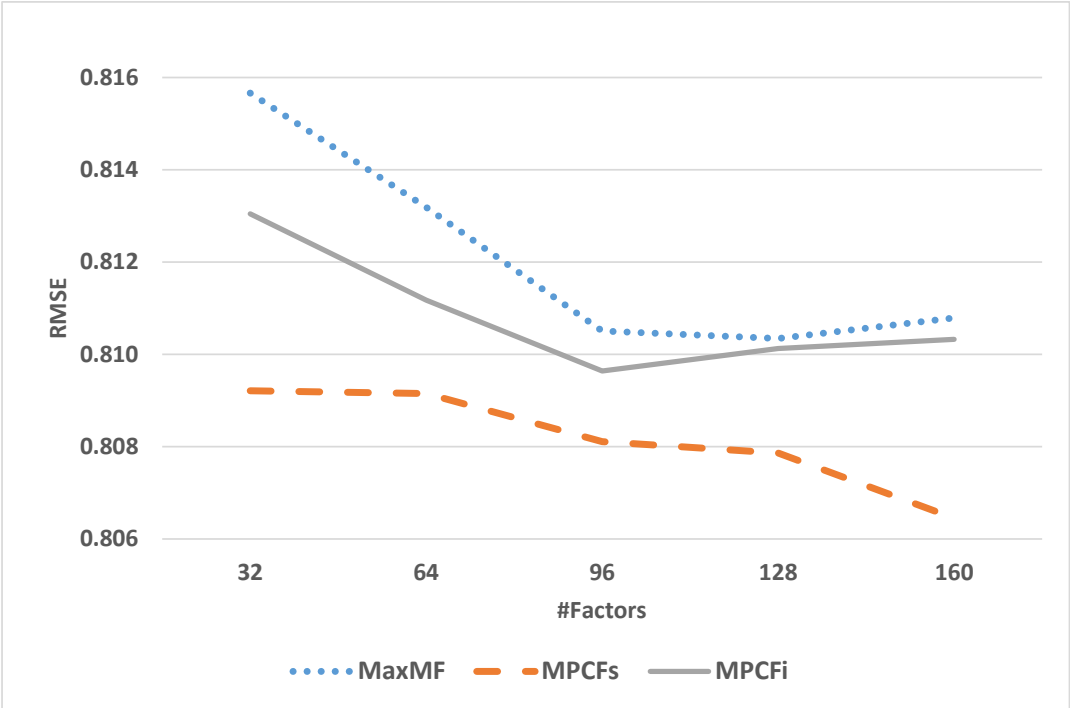

Figure 6.4: Comparison with MaxMF for Rating Prediction.

The results show that, the MPCF methods perform better than the competing methods for all the datasets. Note that except the Grades dataset, the MaxMF method does not outperform MF for the rest of the datasets. This means, in the context of the standard MF method, increasing the number of local preferences beyond one does not help to achieve better performance for these datasets. Thus, having a combination of global preference and local preferences components is beneficial to achieve better rating prediction performance.

Top-N Recommendation Task

Table 6.3 and Table 6.4 shows the overall recommendation performance of the MPCF methods for the top-N recommendation task in terms of HR and ARHR in comparison to the other state-of-the-art methods (Section 4.4). For all the results presented, the number of top-N items chosen is 10 (i.e., $N = 10$).

Table 6.2: Comparison of performance of Rating Prediction algorithms with MPCF

Method	ML1M					Grades				
	Params				RMSE	Params				RMSE
UserKNN	100	-	-	-	0.9188	50	-	-	-	4.5186
MF	96	0.01	0.0005	-	0.8712	96	0.1	0.0005	-	1.8598
MaxMF	96	1	0.01	0.0005	0.8712	128	2	0.0075	0.00075	1.8576
MPCFs	128	2	0.06	0.001	<u>0.8664</u>	192	2	0.01	0.0025	1.8550
MPCFi	128/32	4	0.025	0.005	0.8668	32/128	2	0.0025	0.001	<u>1.8514</u>

Method	Netflix					Flixster-1				
	Params				RMSE	Params				RMSE
UserKNN	50	-	-	-	0.8678	150	-	-	-	0.8749
MF	128	0.0075	0.005	-	0.8103	96	0.01	0.001	-	0.8680
MaxMF	128	1	0.0075	0.005	0.8103	96	1	0.01	0.001	0.8680
MPCFs	160	4	0.03	0.0005	<u>0.8064</u>	128	2	0.02	0.0005	<u>0.8671</u>
MPCFi	96/96	2	0.01	0.001	0.8096	128/96	2	0.01	0.001	0.8673

Method	Yahoo				
	Params				RMSE
UserKNN	50	-	-	-	1.2028
MF	128	0.025	0.005	-	1.1633
MaxMF	128	3	0.025	0.01	1.1572
MPCFs	384	2	0.07	0.005	<u>1.1546</u>
MPCFi	128/32	2	0.05	0.001	1.1557

Columns corresponding to "params" indicate the model parameters for the corresponding method. For UserKNN method, the parameter is the number of neighbors. For MF method, the parameter is the number of latent factors, regularization constant and the learning rate. For MaxMF and MPCFs methods, the parameters correspond to the number of latent factors, number of interests, regularization constant and learning rate. For MPCFi method, the parameters correspond to number of latent factors for global/interest-specific preference, number of interests, regularization constant and learning rate. <u>Underlined</u> entries represent the best performing model measured in terms of RMSE for each dataset.

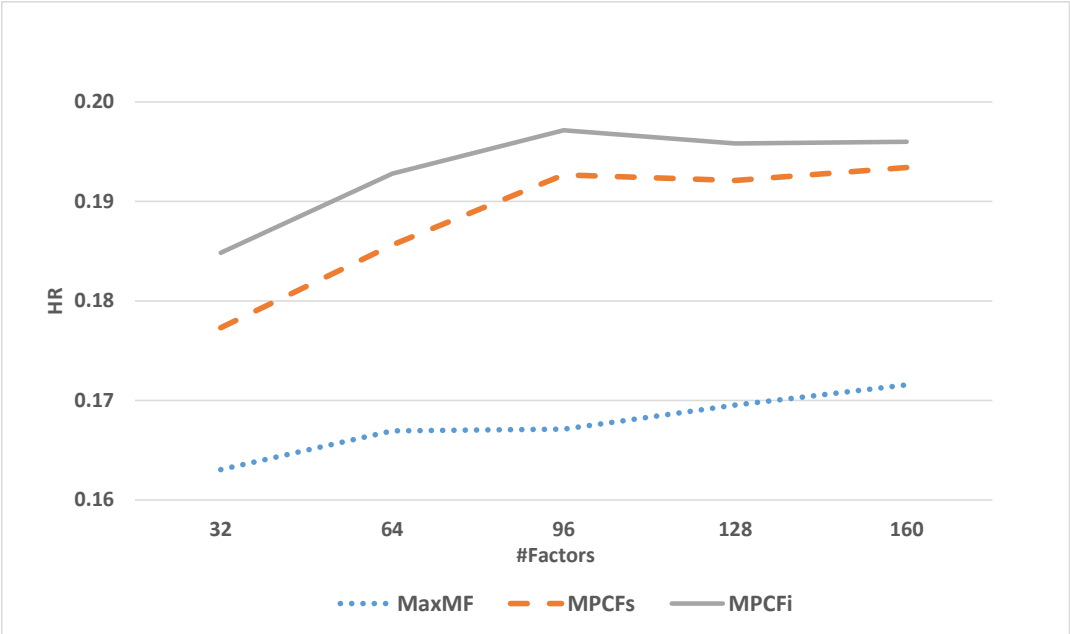

Figure 6.5: Comparison with MaxMF for Top-N.

Columns corresponding to "params" indicate the model parameters for the corresponding method. For UserKNN method, the parameter is the number of neighbors. For PureSVD method, the parameter is the number of latent factors. For BPRkNN method, the parameters are the number of latent factors used and the learning rate. For SLIM method, the parameters correspond to the ℓ_2 and ℓ_1 regularization constants. For MaxMF and MPCFs methods, the parameters correspond to the number of latent factors, number of interests, regularization constant and learning rate. For MPCFi method, the parameters correspond to number of latent factors for global/local components, number of interests, regularization constant and learning rate. Underlined numbers represent the best performing model measured in terms of HR for each dataset.

Similar to the results for the rating prediction task, this results show that, the MPCF methods perform better than the rest of the competing methods for all the datasets for the top-N recommendation task as well. The performance gains of the MPCF methods compared to the next best performing baseline method are of the order of 6% and 10% for the Netflix and Flixster datasets, respectively. Note that similar to results of the rating prediction task and contrary to the results presented in [50], the MaxMF model

does not outperform the PureSVD method for some of the datasets considered in this study.

Table 6.3: Comparison of performance of Top-N recommendation algorithms with MPCF for Netflix Dataset

Method	**Netflix**					
	Params				**HR**	**ARHR**
UserKNN	100	-	-	-	0.1412	0.0515
PureSVD	50	-	-	-	0.1821	0.0807
BPRMF	400	0.01	-	-	0.1890	0.0813
SLIM	0.001	0.1	-	-	0.1888	0.0872
MaxMF	192	2	0.0005	0.0005	0.1743	0.0704
MPCFs	192	2	0.01	0.0005	0.1975	0.0870
MPCFi	256/160	2	0.008	0.001	0.1999	0.0835
Method	**Netflix-4**					
	Params				**HR**	**ARHR**
UserKNN	500	-	-	-	0.0709	0.0186
PureSVD	50	-	-	-	0.0776	0.0234
BPRMF	200	0.001	-	-	0.0751	0.0231
SLIM	0.001	0.1	-	-	0.0805	0.0246
MaxMF	192	1	0.0001	0.0005	0.0788	0.0218
MPCFs	224	3	0.01	0.001	0.0837	0.0252
MPCFi	192/128	2	0.01	0.0025	0.0846	0.0259

In terms of the two proposed MPCF methods, unlike the rating prediction task, there is clearly a better performing scheme for top-N recommendation task. The independent item factors model (MPCFi) consistently achieved better performance than the shared item factors model (MPCFs). The reason for this could be that, MPCFi has the ability to learn the global preference and local preferences components independently, as the items factors are not shared, thereby resulting in learning better representation of users and items. This allows the model to strike a better balance between the two components compared to MPCFs, which shares the item factors during the learning process.

Table 6.4: Comparison of performance of Top-N recommendation algorithms with MPCF for Flixster Dataset

Method	Flixster-1					
	Params				HR	ARHR
UserKNN	100	-	-	-	0.1013	0.0295
PureSVD	100	-	-	-	0.1273	0.0494
BPRMF	200	0.01	-	-	0.1165	0.0437
SLIM	0.01	1.0	-	-	0.1303	0.0602
MaxMF	160	2	0.0001	0.0005	0.1245	0.0763
MPCFs	256	2	0.01	0.005	0.1401	0.0532
MPCFi	288/192	2	0.01	0.001	0.1441	0.0546
Method	**Flixster-2**					
	Params				**HR**	**ARHR**
UserKNN	400	-	-	-	0.0625	0.0125
PureSVD	20	-	-	-	0.0724	0.0224
BPRMF	200	0.001	-	-	0.0707	0.0202
SLIM	0.0001	1.0	-	-	0.0843	0.0241
MaxMF	192	1	0.0001	0.005	0.0836	0.0232
MPCFs	192	4	0.01	0.001	0.0900	0.0275
MPCFi	160/64	2	0.008	0.0025	0.0908	0.0276

6.3.5 Data Sparsity

Figure 6.6 shows the relative top-N performance of the MPCF methods against the MaxMF method for datasets across the two different sparsity levels. These results show that as the dataset gets sparser, the relative performance of MPCF methods (in terms of HR) increases. This is in accordance with our hypothesis that the MaxMF method might suffer from data sparsity problem due to its lack of ability to capture user's global and local preferences when the availability of user preference data decreases.

6.4 Conclusion

In this chapter, we presented a novel user modeling approach for collaborative filtering called MPCF that is applicable to both rating prediction and top-N recommendation tasks. MPCF models the users as a combination of global preference and local preferences

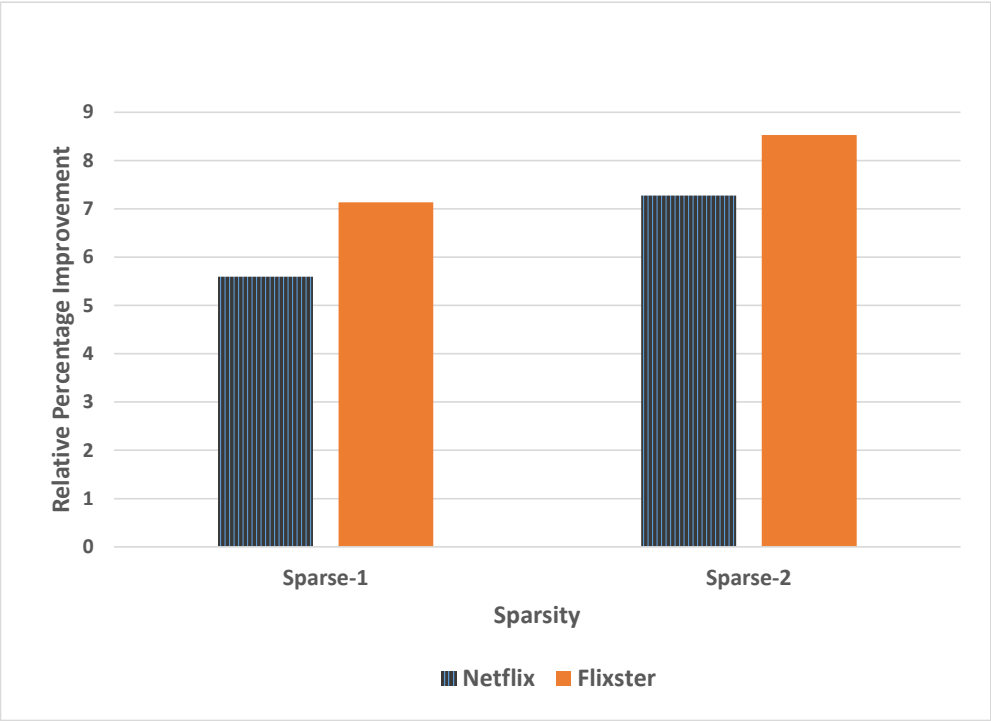

Figure 6.6: Effect of Sparsity on top-N Performance.

components to better capture the high level and finer preferences of users corresponding to their different preferences for item features. The recommendation score is computed as a sum of the scores from the components representing global preference and local preferences. We presented two different approaches to model the different use cases which affect the user preferences at a finer level. The first approach (MPCFi) models the strong preferences of users corresponding to a subset of item features along with a set of local preferences that correspond to the remaining set of item features on which the users are willing to make a tradeoff, whereas the second approach (MPCFs) models the users preferences on all items as part of the global preference component and each of the local preferences correspond to tradeoffs that the users are willing to take on the item features. The results showed that the proposed methods outperform rest of the state-of-the-art methods in terms of both the rating prediction and the top--N recommendation performance. As a future research work, we plan to investigate ranking loss based objective function for top-N recommendation task. Also, in terms of

the relative performance between the MPCFi and MPCFs, the results showed that even though in the case of top-N recommendations, MPCFi perform consistently better than MPCFs, their performance for the rating prediction task was not quite similar. As part of our future research we plan to investigate the reasons behind this inconsistency.

Chapter 7

ClustMF: Combined Neighborhood and Latent Factor Models for Top-N Recommender Systems

This chapter focuses on developing a new method called ClustMF, which is designed to capture the benefits of both the neighborhood based models and the latent factor models. Neighborhood methods are good in capturing the localized neighborhood of users/items, thereby providing direct context to users for the recommended items while the Latent Factor models are good in capturing the overall global relations between the users and items. Thus, there is a need to combine these two class of methods to gain both the benefits associated with them.

7.1 Introduction

Neighborhood models like UserKNN and ItemKNN are intuitive and simple to implement. They are most effective at detecting localized relationships, as they depend on only a few significant neighborhood relations to compute the ranked list of top-N recommendations. This property helps to better explain the recommendations provided to the user. For example, in case of ItemKNN method, the recommended items are the

items which are most similar to the items rated by the user. Thus, the recommendations provided by neighborhood methods are backed by the observed data (in terms of co-rated users), and thus are somewhat "familiar" to the user, as they can be shown to be explicitly related to an item they have consumed in the past. However, the main limitation of neighborhood models is that the top recommended similar items are computed using only a small fraction of the user's preferences. Thus, they fail to capture the sum total of the weak signals provided by all of the user's preferences.

On the other hand, latent factor models utilize all the user's preferences in learning the user and item latent factors to produce the recommendations. Thus, they are generally effective in capturing the overall relations that exist among the users and the items. Although individually, latent factor models have been shown to produce superior top-N recommendations compared to neighborhood models, the recommendations provided by latent factor models cannot be easily explained to the user; the notion of observed item neighborhood is not present and thus the user familiarity is absent. Even though the latent factor models implicitly captures the relations between the items utilizing all the rating data, unlike neighborhood models, the computed item relations are not explicitly backed by the user co-rating data. This aspect of latent factor models is helpful in bringing diversity and serendipity to the recommended list of items, but not the user familiarity. For top-N recommendation task, the absence of familiarity might affect the choice a user makes from the computed list of top-N recommended items, since there is a potential for the user to lose the context on why a particular item was recommended to him/her, in particular for items which lie "far away" in the neighborhood of the items rated by the user. Another disadvantage of these models is that, unlike neighborhood models, they are not capable of detecting the strong relations that exists between a small set of closely related items. Hence, there is a need for a combined model, which can capitalize on both the benefits of the neighborhood models and the latent factor models. That is, a model which can capture the localized relationships like neighborhood models to bring in the familiarity aspect to the users and also capture the global relations between users and items like the latent factor models to provide better recommendations.

Therefore, these two class of methods are complimentary to each other w.r.t. the different properties of the data and the different kinds of users/items relations that

they capture. Thus, there is a need for a combined model which can capture both the benefits of these models. In this thesis chapter, we propose a new method called ClustMF which is designed to combine the benefits of both the neighborhood and latent factor models. The benefits of latent factor models are utilized by modeling the users and items similar to the standard MF based methods. That is, the users and items are represented with latent vectors, where the item latent vectors correspond to the latent features associated with them and the user latent vectors correspond to the user preferences on the item features. The benefit of neighborhood models are brought into the model, by introducing biases at the cluster level. Unlike MF models, which model bias for users and items at a global level, i.e., they learn a single bias value for each of the users and items, in ClustMF, the biases for users are modeled at the item cluster level and the biases for items are modeled at the user cluster level. That is, user will have a different bias value for each of the item clusters and vice versa for the items. These cluster level biases are introduced to capture the localized relationships present in the user and/or item neighborhoods. For an item to be part of the top-N list of the user, along with the latent factors component producing a high score, the corresponding user cluster bias and item cluster bias must also be high. That is, to have a high user cluster bias, the item must be in the neighborhood of the items that the user has liked in the past and to have a high item cluster bias, the user must be in the neighborhood of the users who have liked the item.

The key contributions of the work presented in this thesis chapter are the following:

(i) Proposes a new method ClustMF which combines the benefits of latent factors model and neighborhood based models. In the proposed approach, user biases are modeled at item clusters level and item biases at user clusters level, and the model parameters are learned by using a ranking objective (BPR).

(ii) Proposes a cluster refinement technique to update the cluster assignments based on the ranking objective.

(iii) Proposes an alternating approach to learn both the model parameters (i.e., latent factors, user and item cluster biases) and the user and item cluster assignments together.

(iv) Compares the performance of the proposed model with other state-of-the-art

methods in the context of top-N recommendation tasks and investigates the impact of various parameters as they relate to number of latent factors, number of clusters and biases.

Our experimental evaluation on multiple datasets show that the ClustMF method performs better than the state-of-the-art methods for top-N recommendation tasks.

7.2 ClustMF Method

In ClustMF, given a user u, item i, user cluster assignments vector $\mathbf{cu}$ and item cluster assignments vector $\mathbf{ci}$, recommendation score is computed as,

$$\hat{r}_{ui} = \mathbf{BU}_{u,\mathbf{ci}[i]} + \mathbf{BI}_{i,\mathbf{cu}[u]} + \mathbf{p}_u \mathbf{q}_i^{\mathsf{T}}, \tag{7.1}$$

where $\mathbf{ci}[i]$ is the cluster of item i, $\mathbf{cu}[u]$ is the cluster of user u, $\mathbf{BU}$ is the user-cluser bias matrix and $\mathbf{BU}_{u,\mathbf{ci}[i]}$ is the cluster bias corresponding to the user u and item cluster $\mathbf{ci}[i]$, and $\mathbf{BI}$ is the item cluster bias matrix and $\mathbf{BI}_{i,\mathbf{cu}[u]}$ is the cluster bias corresponding to the item i and user cluster $\mathbf{cu}[u]$. Thus, the recommendation score is computed by taking the sum of the user and item cluster biases, and the dot product of the latent factors corresponding to the given user and item.

We propose an approach similar to Alternating Least Squares (ALS) [7] to learn both the model parameters (i.e., latent factors and cluster biases) and the user and item cluster assignments together in an alternating manner. In this approach, the cluster assignments are initially fixed and the rest of the model paramters are learnt. Next, by fixing the learnt model parameters, the clusters assignments are refined. These two steps are repeated until convergence. This alternating learning process is deemed as converged when the number of cluster changes falls below a pre-defined threshold. The overall alternating approach of learning the model parameters and the cluster assignments together is outlined in Algorithm 8. For the initial cluster assignments we use CLUTO [81] to compute the cluster labels for the users and items.

To estimate the latent factors and cluster biases of the ClustMF model, we use the BPR [53] criterion, which is a pairwise ranking based method. The parameters of the

Algorithm 8 ClustMF:Learn.

```
1: procedure ClustMF _Learn
2:     cu ← initial user cluster assignments from CLUTO
3:     ci ← initial item cluster assignments from CLUTO
4:     BU ← user-cluster bias matrix
5:     BI ← item-cluster bias matrix
6:     iter ← 0
7:     Init P, Q, BU and BI with random values in (-0.001, 0.001)
8:
9:     while iter < maxIter or error on validation set decreases do
10:        (P, Q, BU, BI) = Learn_Parameters(P, Q, BU, BI, cu, ci)
11:        ci = Refine_Item_Clusters(P, Q, BU, BI, cu, ci)
12:        cu = Refine_User_Clusters(P, Q, BU, BI, cu, ci)
13:        iter ← iter + 1
14:    end while
15: end procedure
```

model are learnt by maximizing the following regularized objective function,

$$\underset{\mathbf{BU},\mathbf{BI},\mathbf{P},\mathbf{Q}}{\text{maximize}} \sum_{(u,i,j)\in D_S} \ln \sigma(\hat{x}_{uij}) - (\frac{\gamma}{2}\,(\|\mathbf{BU}\|_F^2 + \|\mathbf{BI}\|_F^2) + \frac{\lambda}{2}\,(\|\mathbf{P}\|_F^2 + \|\mathbf{Q}\|_F^2)), \quad (7.2)$$

where D_S is the set of sampled triplets (u, i, j), i is a higher rated item than j for u, $\hat{x}_{uij}$ is the difference in the predicted ratings for items i and j, i.e., $\hat{x}_{uij} = \hat{r}_{ui} - \hat{r}_{uj}$, $\sigma(\hat{x}_{uij})$ is the sigmoid function, i.e., $\sigma(x) = 1/(1 + e^{-x})$, $\ln(x)$ is the natural logarithm function of x, and λ and γ are ℓ_2-regularization constants. We use Stochastic Gradient Descent (SGD) based algorithm to learn the paramters of the model. Algorithm 9 provides the detail of the learning algorithm.

In the cluster refinement stage, given a user (or item), the BPR objective is computed by assigning the user (or item) to each of the clusters. That is, if there are l_u (or l_i) user (or item) clusters, l_u (or l_i) different BPR objectives are computed for each of the users (or items). The user (or item) is finally assigned to the cluster which provides the maximum BPR objective. The main intuition behind this approach is that, the user (or item) is moved to a cluster which best optimizes the overall personalized ranking criterion based on the learned model parameters. A threshold is used to limit the movement of users (or items) to a different clusters for minor improvements in the ranking objective. The details of the cluster refinement algorithm for items and users

Algorithm 9 ClustMF:Learn_Factors.

procedure LEARN_FACTORS
  $\eta \leftarrow$ learning rate
  Init $\mathbf{BU}, \mathbf{BI}, \mathbf{P}, \mathbf{Q}$ with random values
  $iter \leftarrow 0$
  $D_S \leftarrow$ sampled triplets (u, i, j) from $\mathbf{R}$
  random shuffle D_S

  while not converged **do**
    draw (u, i, j) from D_S randomly
    $\hat{r}_{ui} \leftarrow \mathbf{BU}_{u,\mathbf{ci}}[i] + \mathbf{BI}_{i,\mathbf{cu}}[u] + \mathbf{p}_u \mathbf{q}_i^\mathsf{T}$
    $\hat{r}_{uj} \leftarrow \mathbf{BU}_{u,\mathbf{ci}}[j] + \mathbf{BI}_{j,\mathbf{cu}}[u] + \mathbf{p}_u \mathbf{q}_j^\mathsf{T}$
    $\hat{x}_{uij} \leftarrow \hat{r}_{ui} - \hat{r}_{uj}$
    $logit \leftarrow 1/(1 + e^{\hat{x}_{uij}})$

    $\mathbf{BU}_{u,\mathbf{ci}}[i] \leftarrow \mathbf{BU}_{u,\mathbf{ci}}[i] + \eta\,(logit - \gamma \cdot \mathbf{BU}_{u,\mathbf{ci}}[i])$
    $\mathbf{BU}_{u,\mathbf{ci}}[j] \leftarrow \mathbf{BU}_{u,\mathbf{ci}}[j] + \eta\,(-logit - \gamma \cdot \mathbf{BU}_{u,\mathbf{ci}}[j])$
    $\mathbf{BI}_{i,\mathbf{cu}}[u] \leftarrow \mathbf{BI}_{i,\mathbf{cu}}[u] + \eta\,(logit - \gamma \cdot \mathbf{BI}_{i,\mathbf{cu}}[u])$
    $\mathbf{BI}_{j,\mathbf{cu}}[u] \leftarrow \mathbf{BI}_{j,\mathbf{cu}}[u] + \eta\,(-logit - \gamma \cdot \mathbf{BI}_{j,\mathbf{cu}}[u])$
    $\mathbf{p}_u \leftarrow \mathbf{p}_u + \eta\,(logit \cdot (\mathbf{q}_i - \mathbf{q}_j) - \lambda \cdot \mathbf{BI}_{j,\mathbf{cu}}[u])$
    $\mathbf{q}_i \leftarrow \mathbf{q}_i + \eta\,(logit \cdot \mathbf{p}_u - \lambda \cdot \mathbf{q}_i)$
    $\mathbf{q}_j \leftarrow \mathbf{q}_j + \eta\,(-logit \cdot \mathbf{p}_u - \lambda \cdot \mathbf{q}_j)$

    $iter \leftarrow iter + 1$
  end while

  return $\mathbf{BU}, \mathbf{BI}, \mathbf{P}, \mathbf{Q}$
end procedure

is presented in Algorithm 10.

Algorithm 10 RefineClusters.

procedure REFINE_ITEM_CLUSTERS
  ci $\leftarrow$ input item cluster assignment
  $l_i \leftarrow$ number of item clusters
  for all Items $i \in 1, 2, \ldots, m$ **do**
    $D_{Si} \leftarrow$ sampled triplets containing i
    for all Clusters $c \in 1, 2, \ldots, l_i$ **do**
      assign item i to cluster c
      $clust\_obj[c] \leftarrow \sum_{(u,i,j) \in D_{Si}} \log \sigma(\hat{x}_{uij})$
    end for
    ci$[i] \leftarrow \arg\max_c clust\_obj[c]$
  end for
  return ci
end procedure

procedure REFINE_USER_CLUSTERS
  cu $\leftarrow$ input user cluster assignment
  $l_u \leftarrow$ number of user clusters
  for all Users $u \in 1, 2, \ldots, n$ **do**
    $D_{Su} \leftarrow$ sampled triplets containing u
    for all Clusters $c \in 1, 2, \ldots, l_u$ **do**
      assign user u to cluster c
      $clust\_obj[c] \leftarrow \sum_{(u,i,j) \in D_{Su}} \log \sigma(\hat{x}_{uij})$
    end for
    cu$[u] \leftarrow \arg\max_c clust\_obj[c]$
  end for
  return cu
end procedure

7.3 Results

7.3.1 Effect of Number of Clusters

In this study, we compare the effect of having different number of user and item clusters in ClustMF. Figure 7.1 presents the results of this study. The x-axis represents the number of user and item clusters used to build the model. We can see that the recommendation performance does not vary a lot with the number of user and item

clusters. The peak performance is achieved for ML100K with 10 clusters, Netflix for 50 clusters and Yahoo for 100 clusters. We believe, this behavior has to do with the inherent characteristics of the different datasets, since different datasets tends to have different number of natural clusters for users/items.

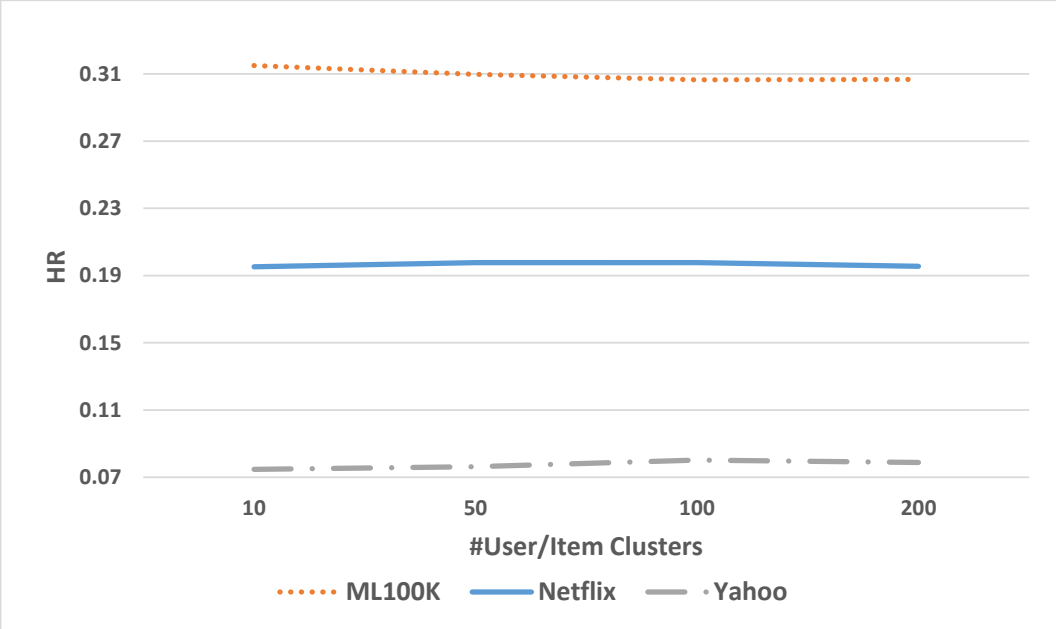

Figure 7.1: Effect of Number of Clusters on top-N Performance.

7.3.2 Effect of Cluster Refinement

In this study, we compare the effect of adding cluster refinement to the ClustMF. The results from this study are presented in Table 7.1. The column corresponding to ClustMF_NCR (No Cluster Refinement) is the ClustMF method which does not refine the clusters. We can see that adding the cluster refinement based on the ranking criterion further helps to improve the recommendation performance in terms of the hit-rate.

7.3.3 Comparison With Other Approaches

Table 7.2 shows the overall recommendation performance of the MPCF methods for the top-N recommendation task in terms of HR and ARHR in comparison to the other

Table 7.1: Effect of Cluster Refinement (HR).

Dataset	ClustMF_NCR	ClustMF
ML100K	0.3001	<u>0.3150</u>
Netflix	0.1923	<u>0.1984</u>
Yahoo	0.0780	<u>0.0802</u>

state-of-the-art methods (Section 4.4). For all the results presented, the number of top-N items chosen is 10 (i.e., $N = 10$). The presented results in Table 7.2 show that ClustMF performs better than the rest of the methods across all the datasets. For ML100K dataset, the performance improvement is quite substantial.

7.4 Conclusion

In this thesis chapter, we proposed a new method called ClustMF which combines the benefits of neighborhood models and latent factor models for top-N recommendation task. The user preferences and item characteristics were modeled using latent vectors, similar to the standard latent factors model. To bring in the benefits of the neighborhood based models, the user and item biases were modeled at the item and user clusters level respectively. Thus, for an item to be ranked higher in the top-N list, along with the dot product of the corresponding user and latent factors giving a high score, the corresponding user and item cluster biases also play a role. That is, the item must be similar to the other items rated by the user and the user must be similar to the other users who have rated the item. A comprehensive set of experiments on multiple datasets showed that, the proposed method outperforms rest of the state-of-the-art methods for top-N recommendation task.

Table 7.2: Comparison of performance of Top-N recommendation algorithms with ClustMF

Method	ML100K					
		Params			HR	ARHR
UserKNN	50	-	-	-	0.2704	0.0957
ItemKNN	20	-	-	-	0.2778	0.1037
PureSVD	10	-	-	-	0.2888	0.1298
BPRMF	32	0.01	0.001	-	0.2916	0.1287
ClustMF	64	10/10	0.01/2.0	0.005	0.3150	0.1317
Method	**Netflix**					
		Params			**HR**	**ARHR**
UserKNN	50	-	-	-	0.1360	0.0434
ItemKNN	20	-	-	-	0.1151	0.0328
PureSVD	50	-	-	-	0.1845	0.0796
BPRMF	400	0.01	0.001	-	0.1890	0.0872
ClustMF	320	100/50	0.001/2.0	0.001	0.1984	0.0896
Method	**Yahoo**					
		Params			**HR**	**ARHR**
UserKNN	200	-	-	-	0.0627	0.0219
ItemKNN	300	-	-	-	0.0633	0.0225
PureSVD	100	-	-	-	0.0774	0.0310
BPRMF	160	0.01	0.001	-	0.0760	0.0288
ClustMF	160	100/100	1e-4/0.001	0.005	0.0802	0.0310

Columns corresponding to "params" indicate the model parameters for the corresponding method. For UserKNN and ItemKNN methods, the parameter is the number of neighbors. For PureSVD method, the parameter is the number of latent factors. For BPRMF method, the parameters are the number of latent factors used, regularization constant and the learning rate. For ClustMF method, the parameters correspond to the number of latent factors, number of user/item clusters, regularization constant and learning rate. Underlined numbers represent the best performing model measured in terms of HR for each dataset.

Chapter 8

Conclusion

8.1 Thesis Summary

Recommender Systems are prevalent and are widely used in many applications. In particular, in the recent years recommender systems have gained popularity via their usage in e-commerce applications to recommend items so as to help the users in identifying the items that best fit their personal tastes. Computationally, recommender systems represent a set of methods that produce recommendations of relevant items (songs, movies, books etc.,) to the users based on the preferences captured from their consumption history. There are two main tasks associated with recommender system. The first is to predict the preference for a given user-item pair in the form of a numerical rating. This is known as *rating prediction* problem. The second is to generate a ranked list of items for users that best suits their personal preferences. This is known as *top-N recommendation* problem. These problems are typically solved using one of the two classes of methods. First class of methods are called content based methods, and they build models based on the intrinsic properties associated with the users and the items. The second class of methods called collaborative filtering utilizes the preference information available in the form of explicit ratings or implicit feedback. Specifically these models utilize the user/item co-rating information to learn relationships between the users and the items. At a high level, collaborative filtering based approaches can be further classified into two classes. The first class of methods known as neighborhood based methods, explicitly compute/learn the user and/or item neighborhood using the

co-rating data and then uses these neighborhoods to compute the recommendations. In the second class of methods, known as model based approaches learns an explicit model from the data and this model is then used to generate the recommendations. In this thesis, we have presented three different approaches that addresses the rating prediction and top-N recommendation tasks for recommender systems.

Factored Item Similarities Method for top-N Recommendation

The existing state-of-the-art top-N recommendation algorithms suffer from data sparsity issue associated with the user-item preference data and thus fails to capture meaningful relations between users who do not have enough co-rated items. In this thesis to address this, we presented a factored item similarity based method (FISM) for the top-N recommendation problem. FISM learns the item similarities as the product of two matrices, allowing it to generate high quality recommendations even on sparse datasets. The factored representation is estimated using a structural equation modeling approach, which leads to better estimators as the number of factors increases. We conducted a comprehensive set of experiments on multiple datasets at different sparsity levels and compared FISM's performance against that of other state-of-the-art top-N recommendation algorithms. The results showed that FISM outperforms the rest of the methods and the performance gaps increases as the datasets become sparser. For faster recommendation, we showed that sparsity can be induced in the resulting item similarity matrix with minimal reduction in the recommendation quality.

Modeling Global and Local Preferences of Users in Collaborative Filtering

A novel user modeling approach for collaborative filtering called MPCF that is applicable to both rating prediction and top-N recommendation tasks is presented in this thesis. MPCF models the users as a combination of global preference and local preferences components to better capture the high level and finer preferences of users corresponding to their different preferences for item features. The recommendation score is computed as a sum of the scores from the components representing global preference and local preferences. We presented two different approaches to model the different use cases

which affect the user preferences at a finer level. The first approach (MPCFi) models the strong preferences of users corresponding to a subset of item features along with a set of local preferences that correspond to the remaining set of item features on which the users are willing to make a tradeoff, whereas the second approach (MPCFs) models the users preferences on all items as part of the global preference component and each of the local preferences correspond to tradeoffs that the users are willing to take on the item features. The results showed that the proposed methods outperform rest of the state-of-the-art methods in terms of both the rating prediction and the top-N recommendation performance.

Combined Neighborhood and Latent Factor Models for Top-N Recommender Systems

Neighborhood methods are good in capturing the localized neighborhood of users/items, thereby providing direct context to users for the recommended items while the Latent Factor models are good in capturing the overall global relations between the users and items. Thus, there is a need to combine these two class of methods to gain both the benefits associated with them. In this thesis we have developed a new method called ClustMF, which is designed to capture the benefits of both the neighborhood based models and the latent factor models. The benefits of latent factors models are utilized by modeling the users and items similar to the standard MF based methods and benefits of the neighborhood models are brought into the model, by introducing user and item biases at the item and user cluster level. The experimental evaluation showed that the proposed method outperforms rest of the state-of-the-art methods for top-N recommendation performance.

8.2 Future Research Directions

Given the wide spread usage of recommender systems in the recent years, different problems in this domain pose variety of interesting challenge in the field of data mining and machine learning. For the future research direction, the rest of this chapter provides an outline of the various problems associated with the recommender systems.

Along with the user-item preferences data in the form of ratings or consumption,

there is additional data available in the form of user and item features, and richer data associated with preferences like descriptive user reviews, user implicit feedback etc. In the recent years, many methods have been proposed to incorporate each of these additional information. However, there is still lot of scope to utilize this rich data to better model the users preferences. With the advent of the online social networks and it's wide spread adoption, the recommendation methods need to utilize this addition social signals to better model the user behavior. Majority of the social connections are based on the common interests and preferences. Thus, utilizing these signals will help to serve better recommendations to the users. Increased utility of recommender systems has made the deployment of these methods for a wide range of applications like recommending users in a social network, vacation packages on a travel website and a partner on a online dating service. Each of these applications have a different kind of user behavior and thus different kind of feedback data available. Given the exponential increase in the number of users online and the information available corresponding to these users behavior, there is a need to devise methods which not only produce high quality recommendations but also scale to the large amounts of available data. Thus, scalability is an important aspect that every new recommendation algorithm needs to consider.

References

[1] Santosh Kabbur, Xia Ning, and George Karypis. Fism: factored item similarity models for top-n recommender systems. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 659–667. ACM, 2013.

[2] Santosh Kabbur and George Karypis. Nlmf: Nonlinear matrix factorization methods for top-n recommender systems. In *Data Mining Workshop (ICDMW), 2014 IEEE International Conference on*, pages 167–174. IEEE, 2014.

[3] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, 17(6):734–749, 2005.

[4] F. Ricci, L. Rokach, B. Shapira, and P.B. Kantor. Recommender systems handbook. *Recommender Systems Handbook:, ISBN 978-0-387-85819-7. Springer Science+ Business Media, LLC, 2011*, 1, 2011.

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

[6] R. Pan, Y. Zhou, B. Cao, N.N. Liu, R. Lukose, M. Scholz, and Q. Yang. One-class collaborative filtering. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 502–511. IEEE, 2008.

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

[8] 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.

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

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

[11] 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.

[12] 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.

[13] 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*, pages 285–295. ACM, 2001.

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

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

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

[17] Yehuda Koren. Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 4(1):1, 2010.

[18] Thomas Hofmann and D Hartmann. Collaborative filtering with privacy via factor analysis. In *Proceedings of the 2005 ACM Symposium on Applied Computing*, pages 791–795, 2005.

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

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

[21] Robert M Bell and Yehuda Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, pages 43–52. IEEE, 2007.

[22] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Major components of the gravity recommendation system. *ACM SIGKDD Explorations Newsletter*, 9(2):80–83, 2007.

[23] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4(2):133–151, 2001.

[24] Jasson DM Rennie and Nathan Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *Proceedings of the 22nd international conference on Machine learning*, pages 713–719. ACM, 2005.

[25] Benjamin M Marlin. Modeling user rating profiles for collaborative filtering. In *Advances in neural information processing systems*, page None, 2003.

[26] Thomas Hofmann. Latent semantic models for collaborative filtering. *ACM Transactions on Information Systems (TOIS)*, 22(1):89–115, 2004.

[27] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Incremental singular value decomposition algorithms for highly scalable recommender systems. In *Fifth International Conference on Computer and Information Science*, pages 27–28. Citeseer, 2002.

[28] P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 39–46, 2010.

[29] Olivier Bousquet and Léon Bottou. The tradeoffs of large scale learning. In *Advances in neural information processing systems*, pages 161–168, 2008.

[30] Andriy Mnih and Ruslan Salakhutdinov. Probabilistic matrix factorization. In *Advances in neural information processing systems*, pages 1257–1264, 2007.

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

[32] Robert M Bell, Yehuda Koren, and Chris Volinsky. The bellkor solution to the netflix prize, 2007.

[33] Robert M Bell and Yehuda Koren. Lessons from the netflix prize challenge. *ACM SIGKDD Explorations Newsletter*, 9(2):75–79, 2007.

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

[35] Nathan Srebro, Noga Alon, and Tommi S Jaakkola. Generalization error bounds for collaborative prediction with low-rank matrices. In *Advances In Neural Information Processing Systems*, pages 1321–1328, 2004.

[36] Markus Weimer, Alexandros Karatzoglou, and Alex Smola. Improving maximum margin matrix factorization. *Machine Learning*, 72(3):263–276, 2008.

[37] Vikas Sindhwani, Serhat Selcuk Bucak, Jianying Hu, and Aleksandra Mojsilovic. One-class matrix completion with low-density factorizations. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 1055–1060. IEEE, 2010.

[38] Patrik O Hoyer. Non-negative matrix factorization with sparseness constraints. *The Journal of Machine Learning Research*, 5:1457–1469, 2004.

[39] Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online learning for matrix factorization and sparse coding. *The Journal of Machine Learning Research*, 11:19–60, 2010.

[40] Francis Bach, Julien Mairal, and Jean Ponce. Convex sparse matrix factorizations. *arXiv preprint arXiv:0812.1869*, 2008.

[41] Kai Yu and Volker Tresp. Learning to learn and collaborative filtering. In *Neural Information Processing Systems Workshop on Inductive Transfer*, volume 10, 2005.

[42] 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*, pages 19–28. ACM, 2009.

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

[44] Robert Bell, Yehuda Koren, and Chris Volinsky. Modeling relationships at multiple scales to improve accuracy of large recommender systems. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 95–104. ACM, 2007.

[45] Douglas W Oard, Jinmook Kim, et al. Implicit feedback for recommender systems. In *Proceedings of the AAAI workshop on recommender systems*, pages 81–83. Wollongong, 1998.

[46] X. Ning and G. Karypis. Slim: Sparse linear methods for top-n recommender systems. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 497–506. IEEE, 2011.

[47] Judea Pearl. *Causality: models, reasoning and inference*, volume 29. Cambridge Univ Press, 2000.

[48] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[49] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005.

[50] Jason Weston, Ron J Weiss, and Hector Yee. Nonlinear latent factorization by embedding multiple user interests. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 65–68. ACM, 2013.

[51] Neil D Lawrence and Raquel Urtasun. Non-linear matrix factorization with gaussian processes. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 601–608. ACM, 2009.

[52] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*, pages 791–798. ACM, 2007.

[53] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-thieme. Ls: Bpr: Bayesian personalized ranking from implicit feedback. In *In: Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI*, 2009.

[54] Badrul M Sarwar, George Karypis, Joseph Konstan, and John Riedl. Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering. In *Proceedings of the fifth international conference on computer and information technology*, volume 1. Citeseer, 2002.

[55] Mark OConnor and Jon Herlocker. Clustering items for collaborative filtering. In *Proceedings of the ACM SIGIR workshop on recommender systems*, volume 128. Citeseer, 1999.

[56] Songjie Gong. A collaborative filtering recommendation algorithm based on user clustering and item clustering. *Journal of Software*, 5(7):745–752, 2010.

[57] Wee Sun Lee. Collaborative learning for recommender systems. In *ICML*, volume 1, pages 314–321, 2001.

[58] Shyong K Lam Al Mamunur Rashid, George Karypis, and John Riedl. Clustknn: a highly scalable hybrid model-& memory-based cf algorithm. *Proceeding of WebKDD*, 2006.

[59] Lyle H Ungar and Dean P Foster. Clustering methods for collaborative filtering. In *AAAI Workshop on Recommendation Systems*, volume 1, 1998.

[60] Luo Si and Rong Jin. Flexible mixture model for collaborative filtering. In *ICML*, volume 3, pages 704–711, 2003.

[61] Gui-Rong Xue, Chenxi Lin, Qiang Yang, WenSi Xi, Hua-Jun Zeng, Yong Yu, and Zheng Chen. Scalable collaborative filtering using cluster-based smoothing. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 114–121. ACM, 2005.

[62] Sonny Han Seng Chee, Jiawei Han, and Ke Wang. Rectree: An efficient collaborative filtering method. In *Data Warehousing and Knowledge Discovery*, pages 141–151. Springer, 2001.

[63] Kyoung-jae Kim and Hyunchul Ahn. A recommender system using ga k-means clustering in an online shopping market. *Expert systems with applications*, 34(2):1200–1209, 2008.

[64] Thomas George and Srujana Merugu. A scalable collaborative filtering framework based on co-clustering. In *Data Mining, Fifth IEEE International Conference on*, pages 4–pp. IEEE, 2005.

[65] Xiaoyuan Su and Taghi M Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009:4, 2009.

[66] Nicolas Usunier, David Buffoni, and Patrick Gallinari. Ranking with ordered weighted pairwise classification. In *Proceedings of the 26th annual international conference on machine learning*, pages 1057–1064. ACM, 2009.

[67] Shipeng Yu, Kai Yu, Volker Tresp, and Hans-Peter Kriegel. Collaborative ordinal regression. In *Proceedings of the 23rd international conference on Machine learning*, pages 1089–1096. ACM, 2006.

[68] Jason Weston, Samy Bengio, and Nicolas Usunier. Wsabie: Scaling up to large vocabulary image annotation. In *IJCAI*, volume 11, pages 2764–2770, 2011.

[69] Markus Weimer, Alexandros Karatzoglou, Quoc Viet Le, and Alex Smola. Maximum margin matrix factorization for collaborative ranking. *Advances in neural information processing systems*, 2007.

[70] Nathan N Liu and Qiang Yang. Eigenrank: a ranking-oriented approach to collaborative filtering. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 83–90. ACM, 2008.

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

[72] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, pages 89–96. ACM, 2005.

[73] Nguyen Duy Phuong and Tu Minh Phuong. Collaborative filtering by multi-task learning. In *Research, Innovation and Vision for the Future, 2008. RIVF 2008. IEEE International Conference on*, pages 227–232. IEEE, 2008.

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

[75] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pages 10–10, 2010.

[76] Rainer Gemulla, Erik Nijkamp, Peter J Haas, and Yannis Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 69–77. ACM, 2011.

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

[78] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Scalable collaborative filtering approaches for large recommender systems. *The Journal of Machine Learning Research*, 10:623–656, 2009.

[79] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Investigation of various matrix factorization methods for large recommender systems. In *Data Mining Workshops, 2008. ICDMW'08. IEEE International Conference on*, pages 553–562. IEEE, 2008.

[80] Léon Bottou. Online algorithms and stochastic approximations. In David Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998. revised, oct 2012.

[81] George Karypis. Cluto-a clustering toolkit. Technical report, DTIC Document, 2002.