LOCAL APPROACHES FOR COLLABORATIVE FILTERING

A Thesis
Presented to
The Academic Faculty

by

Joonseok Lee

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in
Computational Science

College of Computing
Georgia Institute of Technology
August 2015

Copyright © 2015 by Joonseok Lee

LOCAL APPROACHES FOR COLLABORATIVE FILTERING

Approved by:

Prof. Guy Lebanon, Advisor
College of Computing
Georgia Institute of Technology

Prof. Hongyuan Zha, Co-Advisor
College of Computing
Georgia Institute of Technology

Prof. Le Song
College of Computing
Georgia Institute of Technology

Prof. Polo Chau
College of Computing
Georgia Institute of Technology

Dr. Yoram Singer
Machine Learning Research Group
Google Inc.

Date Approved: February 19, 2015

To my family.

ACKNOWLEDGEMENTS

My five years of Ph.D. study was a really pleasant time. I am sincerely grateful to everyone who helped me to successfully finish this otherwise hard and uncertain journey.

The greatest fortune I had during my Ph.D. was meeting my advisor, Guy Lebanon. He opened the great door of machine learning and recommendation systems to me. Starting with no background in machine learning and little research experience, he motivated, guided, encouraged, and supported me whenever I need help. Thanks to his constant advices and guidences, I was able to come to the right way.

Collaborating with my two supervisors in Google, Yoram Singer and Samy Bengio, was my greatest honor and pleasure. With their mathematical insights and technical help, I grew up one more step. I challenged a quite different area during my Microsoft Research internship, but I was able to successfully finish it thanks to bottomless support of Ariel Fuxman, Bo Zhao, and Yuanhua Lv. I truly enjoyed my time in Amazon as well, which was possible thanks to my kind and friendly mentors, Ted Sandler and Ainur Yessenalina.

I appreciate all the invaluable comments by the committee members, Hongyuan Zha, Le Song, and Polo Chau. Hongyuan opened my eyes to wider world. Thanks again for his invaluable comments and advice. It was a great time to work for Le's machine learning class as a T.A. I also feel grateful to all of his valuable comments in our discussion. Polo provided many concrete and detailed comments on this thesis. Thanks again.

This thesis was made of lots of collaboration with Mingxuan Sun and Seungyeon Kim. With constructive discussions and sharing experiences, I learned a lot of lessons from them. I would like to thank for all of their help. Besides academic cooperation, Seungyeon helped me a lot to get settled down in Atlanta at the beginning. Thanks to his help, I was able to start my journey in a foreign land far away from home without any problem.

All of my labmates made my time in graduate school truly enjoyable. I am grateful to academic siblings in SMLV lab, Joshua Dillon, Krishnakumar Balasubramanian,

Sanjeet Hajarnis, and Fuxin Li, for sharing interesting ideas and wonderful time in the lab. I thank other lab mates, Jaegul Choo, Jingu Kim, Nishant Mehta, Dongyeol Lee, Ravi Sastry, Shuanghong Yang, Ke Zhou, Quoc Long Tran, Nan Du, Changhyun Lee, Bo Xie, Minsuk Kahng, Spoorthi Ravi, Yichen Wang, Hanna Kim, Edward Choi, and Joachim Perros for making my graduate school life more funny and enjoyable.

I will never forget my greatest time with my friends in Atlanta. Sehoon Ha, Kisung Lee, Sungihk Yang, Hyojong Kim, James Yang, Hyojoon Kim, Chayoung Lee, Stephan Lee, and Joohwan Lee: thanks again for sharing this great journey with me.

Most importantly, I can't express in words my gratitude to my parents, for their endless support and love. It was my greatest pleasure to see they were happy when I told all the minor achievements. Joonyoung, my brother, I know you are covering my role as the first son at all time. Without your support, I simply couldn't have done this long study.

My wife, Sookyung, you are the greatest driving force in my life. I always thank for being with me, without any condition. This thesis is made of your love and support. I love you.

Contents

List of Tables

List of Figures

LIST OF SYMBOLS AND ABBREVIATIONS

ARFF	Attribute-Relation File Format
ASYM	Asymmetric Loss
BPMF	Bayesian Probabilistic Matrix Factorization [115]
BSD	Berkeley Software Distribution
CF	Collaborative Filtering
DFC	Divide-and-Conquer Matrix Factorization [85]
ECF	Ensembles of Collaborative Filtering
FWLS	Feature-Weighted Linear Stacking [123]
GCR	Global Collaborative Ranking
HLU	Half-Life Utility Score [12]
LCR	Local Collaborative Ranking
LLORMA	Local Low-Rank Matrix Approximation
LRMA	Low-Rank Matrix Approximation
MAE	Mean Absolute Error
MF	Matrix Factorization
MMMF	Maximum Margin Matrix Factorization
MSE	Mean Squared Error
NDCG	Normalized Discounted Cumulative Gain
NMAE	Normalized Mean Absolute Error
NMF	Non-negative Matrix Factorization [79]
NPCA	Non-parametric Principal Component Analysis [147]
NLPMF	Nonlinear Probabilistic Matrix Factorization [75]
PMF	Probabilistic Matrix Factorization [114]
PREA	Personalized Recommendation Algorithms Toolkit
RMSE	Root-Mean Squared Error
SVD	Singular Value Decomposition

SVM	Support Vector Machine
SVT	Singular Value Thresholding
UJMP	Universal Java Matrix Package
M	The observed rating matrix
$\hat{M}$	The estimation of the rating matrix M
n_1	The number of users (rows) in M
n_2	The number of items (columns) in M
m	The number of observed entries in M
r	The rank of factorized matrix $\hat{M} \approx UV^\top$
U	The user profile matrix in incomplete SVD of M
V	The item profile matrix in incomplete SVD of M
Ω	The set of observed entries of M
q	The number of models combined to an ensemble model.
$\mathcal{U}$	The set of users in M
$\mathcal{I}$	The set of items in M
$\mathcal{F}$	The set of recommender systems $f(u, i)$.
$\mathcal{H}$	The set of feature functions $h(u, i)$.
$F(u, i)$	An ensemble of recommender systems $f_1(u, i), ..., f_k(u, i)$.
$\mathcal{P}$	Projection operator. $[\mathcal{P}_\Omega(M)]_{a,b} = M_{a,b}$ if $(a, b) \in \Omega$, 0 otherwise.
$\mathcal{T}$	A mapping from an index to a matrix
$\hat{\mathcal{T}}$	Local approximation of M centered on an element with LLORMA.
$\hat{\hat{\mathcal{T}}}$	Global approximation of M with LLORMA.
K_h^s	A kernel function with bandwidth h centered on a point s.
$B_h(s)$	The neighborhood of indices near a matrix element s.
$\mathcal{E}(\hat{\mathcal{T}})(s, h)$	Average squared-error within a neighborhood of a matrix element s.
$[n]$	The set of integers $\{1, ..., n\}$
$\|X\|_F$	Frobenius norm of matrix X.
$\|X\|_\infty$	Infinite norm of matrix X.
$\|X\|_*$	Nuclear (trace) norm of matrix X.
$\odot$	Entry-wise matrix product. $[A \odot B]_{i,j} = A_{i,j}B_{i,j}$

SUMMARY

Recommendation systems are emerging as an important business application as the demand for personalized services in E-commerce increases. Collaborative filtering techniques are widely used for predicting a user's preference or generating a list of items to be recommended. In this thesis, we develop several new approaches for collaborative filtering based on model combination and kernel smoothing. Specifically, we start with an experimental study that compares a wide variety of CF methods under different conditions. Based on this study, we formulate a combination model similar to boosting but where the combination coefficients are functions rather than constant. In another contribution we formulate and analyze a local variation of matrix factorization. This formulation constructs multiple local matrix factorization models and then combines them into a global model. This formulation is based on the local low-rank assumption, a slightly different but more plausible assumption about the rating matrix. We apply this assumption to both rating prediction and ranking problems, with both empirical validations and theoretical analysis.

We contribute with this thesis in four aspects. First, the local approaches we present significantly improve the accuracy of recommendations both in rating prediction and ranking problems. Second, with the more realistic local low-rank assumption, we fundamentally change the underlying assumption for matrix factorization-based recommendation systems. Third, we present highly efficient and scalable algorithms which take advantage of parallelism, suited for recent large scale datasets. Lastly, we provide an open source software implementing the local approaches in this thesis as well as many other recent recommendation algorithms, which can be used both in research and production.

Chapter I

INTRODUCTION

1.1 Recommendation Systems

As the demand for personalized services in E-commerce increases, recommendation systems are emerging as an important business application. Amazon.com, for example, provides personalized product recommendations based on previous purchases. Other examples include music recommendations in Pandora, movie recommendations in Netflix, and friend recommendations in Facebook.

Broadly speaking, any software system which actively suggests an item to purchase, to subscribe, or to invest can be regarded as a recommender system. In this broad sense, an advertisement also can be seen as a recommendation. In this thesis, we mainly consider a narrower definition of *personalized* recommendation system that base recommendations on user specific information.

There are two main approaches to personalized recommendation systems: content-based filtering and collaborative filtering. The former makes explicit use of domain knowledge concerning users or items. The domain knowledge may correspond to user information such as age, gender, occupation, or location [28], or to item information such as genre, producer, or length in the case of movie recommendation.

The latter category of collaborative filtering (CF) does not use user or item information with the exception of a partially observed rating matrix. The rating matrix holds ratings of items (columns) by users (rows) and is typically binary, for example like vs. dislike, or ordinal, for example, one to five stars as in Netflix movie recommendation. The rating matrix may also be gathered implicitly based on user activity such as a web search followed by click through, which may be interpreted as a positive value judgement for the chosen hyper-link [52]. In general, the rating matrix is extremely sparse, since it is unlikely that each user experienced and provided ratings for all items.

Hybrid collaborative and content-based filtering strategies combine the two approaches above, using both the rating matrix, and user and item information. [103, 150, 104, 88, 66, 7, 104, 125]. Such systems typically obtain improved prediction accuracy over

content-based filtering systems and over collaborative filtering systems.

We can also classify recommender systems according to their main goals. [41] Recommending good items may be the most important goal in many recommender systems. Amazon.com, for example, tries to suggest items which are potentially attractive to particular users. Another goal is optimizing utility, the profit to the company for instance. This can be seen as a slight modification to the first goal, as a weighted sum of recommended items. Lastly, predicting unseen ratings on an item by a user is also a popular use of recommender systems. Netflix, for example, estimates how many stars will be given by a user to a movie. Based on this prediction, it can recommend movies to those who gave high scores.

This thesis, among the various recommender systems, focuses on collaborative filtering (CF), mainly for rating prediction. As collaborative filtering methods use only the rating matrix which is similar in nature across different domains, the results and conclusions in this thesis can be applied to any CF-based recommender systems independent of the domain.

1.2 Collaborative Filtering

Collaborative filtering systems are usually categorized into two subgroups: memory-based and model-based methods. Memory-based methods simply store the rating matrix and issue recommendations based on the relationship between the queried user and item and the rest of the rating matrix. Model-based methods fit a parameterized model to the given rating matrix and then issue recommendations based on the fitted model.

The most popular memory-based CF methods are neighborhood-based methods, which predict ratings by referring to users whose ratings are similar to the queried user, or to items that are similar to the queried item. This is motivated by the assumption that if two users have similar ratings on some items they will have similar ratings on the remaining items. Alternatively, if two items have similar ratings by a portion of the users, the two items will have similar ratings by the remaining users.

Specifically, user-based CF methods [12] identify users that are similar to the queried user, and estimate the desired rating to be the average ratings of these similar users. Similarly, item-based CF [117] identify items that are similar to the queried item, and estimate the desired rating to be the average of the ratings of these similar items.

Neighborhood methods vary considerably in how they compute the weighted average of ratings. Specific examples of similarity measures that influence the averaging weights are include Pearson correlation, vector cosine, and mean-squared-difference (MSD). Neighborhood-based methods can be extended with default votes, inverse user frequency, and case amplification [12]. A recent neighborhood-based method [128] constructs a kernel density estimator for incomplete partial rankings and predicts the ratings that minimize the posterior loss.

Model-based methods, on the other hand, fit a parametric model to the training data that can later be used to predict unseen ratings and issue recommendations. Model-based methods include cluster-based CF [134, 22, 26, 118, 146], Bayesian classifiers [92, 93], and regression-based methods [136]. The slope-one method [82] fits a linear model to the rating matrix, achieving fast computation and reasonable accuracy.

A recent class of successful CF models are based on low-rank matrix factorization. The regularized SVD method [11] factorizes the rating matrix into a product of two low-rank matrices (user-profile and item-profile) that are used to estimate the missing entries. An alternative method is Non-negative Matrix Factorization (NMF) [78] that differs in that it constrain the low-rank matrices forming the factorization to have non-negative entries. Recent variations are Probabilistic Matrix Factorization (PMF) [115], Bayesian PMF [114], Non-linear PMF [75], Maximum Margin Matrix Factorization (MMMF) [124, 108], and Nonlinear Principal Component Analysis (NPCA) [147].

Some recent work suggested that combining different CF models improve the prediction accuracy. Specifically, a memory-based method linearly combined with a latent factor method [8, 70] retains the advantages of both models. Ensembles of maximum margin matrix factorizations were explored to improve the result of a single MMMF model [29]. A mixture of experts model [125] proposes to linearly combine the prediction results of more than two models. Feature-weighted linear stacking [123] uses dynamic weights for linear combination of recommenders.

Some representative methods listed above will be briefly reviewed in Chapter 2. For more information on recommender systems and collaborative filtering, several well-written surveys [1, 126, 57] may be useful. Also, Several experimental studies may provide useful information [12, 53, 80].

Table 1.1: Standard Datasets for Collaborative Filtering.

Dataset Name	User Count	Item Count	Rating Count	Density
MovieLens 100K	943	1,682	100,000	6.30%
MovieLens 1M	6,039	3,883	1,000,181	4.27%
MovieLens 10M	69,877	10,681	9,905,064	1.33%
Yelp	43,878	12,742	215,903	0.03%
EachMovie	72,916	1,628	2,811,983	2.37%
Netflix	480,046	17,770	95,947,878	1.12%
BookCrossing	278,858	271,379	1,149,780	0.002%

1.2.1 Netflix Prize and Dataset

The Netflix competition[1] was held between October 2006 and July 2009, when BellKor's Pragmatic Chaos team won the million-dollar-prize. The goal of this competition was improving the prediction accuracy of the Netflix movie recommendation system by 10%. The winning team used a hybrid method with temporal dynamics to account for dates in which ratings were reported [70, 9]. The second-placed team, The Ensemble [123], achieved comparable performance to the winner team, with a linear combination with non-constant weights.

Although the Netflix competition has finished, the dataset used in that competition is still used as a standard dataset for evaluating CF methods. It has 480,046 users and 17,770 items with 95,947,878 ratings. This represents a density level of 1.12% (total number of entries divided by observed entries). Table 1.1 lists other standard datasets including MovieLens, EachMovie, and BookCrossing.

1.3 Research Questions

Since Netflix competition, a plethora of new recommendation algorithms have been proposed. Most of them are arguing that they are the state-of-the-art, and performing better than existing methods. The question we focus in this thesis is what is the best way of building a recommendation system with collaborative filtering. We pose the following questions, and will search an answer for each of them throughout this thesis.

Q1. What are the existing CF models for rating prediction, and what is the best?

[1] http://www.netflixprize.com

As a starting question, we first survey the current state-of-the-art CF-based recommendation algorithm. Several more detailed questions will follow: What are the existing CF models? How do they predict unseen ratings? What is the best-performing CF algorithm, in what sense? Is this always superior to others?

If one is always better than others, our answer will be simple; we always use the best-performing recommender system for every application. Otherwise, we need to research the next questions. To answer this question, we first survey existing models, implement them, and experimentally evaluate them in multifaceted ways, in Chapters 3 and 4.

Q2. If one is not always the best, how can we combine multiple CF models?

Given that multiple models perform better or worse than others depending on the region of the rating matrix, that is, depending on the target user and item, how can we combine models so as to improve the overall prediction accuracy the most? Many possible solutions can be used, such as decision trees.

To maximize the advantage of combining multiple models, we consider a weight function that is dependent on the target user and item. A previous work [123] proposed a similar approach with human-crafted features (weights). Can we automate this feature learning process? How much can it be efficient and scalable? Chapter 5 will discuss these questions and our solution.

Q3. Instead of combining existing models, can we learn local models as well as weights?

Without assuming that we already have a set of (state-of-the-art) recommendation models in hand, can we learn participating models as well as the weights at the same time? In this way, each model can focus more on some local properties with particular set of users and items, which were not possible with a global model. There are several questions to implement this idea: How do we divide the entire matrix into local regions? What form of local models do we learn? How should we combine local models to produce the final estimation? What theoretical assumption and guarantees do we have? We answer these questions in Chapter 6.

Q4. Can we use local models to improve quality of ranking as well?

In addition to the rating prediction problem, recently people focus more on ranking problem in recommendation systems. As our main practical purpose with a recommendation system is finding out and recommending several most preferrable items

Table 1.2: Summary of research questions we pose and solve in this thesis.

No.	Research Questions	Chap.
Q1	Is there a CF model performing always the best? If so, what is it?	3, 4
Q2	How can we combine existing CF models to improve performance?	5
Q3	Can we learn local models and weights at the same time?	6
Q4	Can we apply our local approach to ranking problems?	7

for each user, we do not need to waste our effort to estimate the number of stars (preference score) precisely. Instead, what we really need is the top k items potentially the user will like. In this sense, we can formulate recommendation problem as a ranking problem. In Chapter 7, we apply our local approach to this ranking problem formulation to achieve better accuracy and faster computation.

Table 1.2 summarizes research questions we listed in this section.

1.4 Thesis Statement

Local approaches, dividing users and items into subcommunities, learning group-specific models for each of them, and combining them to estimate unseen preferences, significantly improve the performance of collaborative filtering-based recommendation systems in terms of rating prediction and ranking *accuracy* as well as computational *efficiency* and *scalability*.

1.5 Research Contributions and Impact

We summarize our contribution to the research community as well as the wider audiences in society.

1.5.1 Better Quality of Recommendations

First of all, we present several local approaches that significantly improve the accuracy of recommendations. Ensembles of CF with non-constants weights in Chapter 5 and Local Low-Rank Matrix Approximation (LLORMA) in Chapter 6 lower root-mean squared error (RMSE) significantly. Although the improvement in RMSE looks small (less than 0.001), it has been shown since the Netflix competition that lowering the RMSE by 0.0001 is extremely difficult. Local collaborative ranking in Chapter 7 also improves ranking quality in terms of average precision and NDCG.

Local low-rank matrix approximation was not just verified by offline tests with standard datasets, but also was implemented and tested in industry including Google and Amazon during the author's internship. It was verified that local approaches in this thesis were effective to improve recommendation quality in real production, though we cannot provide exact figures publicly.

1.5.2 Theoretical Aspects

In addition to the improvement shown with experiments, we present important fundamental observations for collaborating filtering. The global low-rank assumption has been prevailing in most previous works, but the local low-rank assumption presented in Chapter 6 fundamentally changes the underlying assumption for recommendation systems using matrix factorization. We show that the diminishing returns phenomenon we undergo with higher rank under the global low-rank assumption is mainly because we are stuck at local optimum, not because it is the best solution. We also provide theoretical analysis of the proposed local low-rank matrix approximation method.

1.5.3 Computational Aspects

The algorithms presented in this thesis are highly efficient and scalable for large scale dataset. Local low-rank matrix approximation (Chapter 6) and local collaborative ranking (Chapter 7) can run either in clusters or in a multiprocessor machine in parallel. As the learning process takes less time with a local model than with a global one, and as every local model can be learned at the same time (at least in theory, assuming that we have enough machines), the overall computation time for learning process is even shorter than previous global models despite even superior accuracy. We verify efficiency of our proposed algorithm with large scale data including Netflix dataset as well as real production data from Amazon Instant Video service. With sufficient computing resources, we proved that local approaches in this thesis can run fast enough for real production use.

1.5.4 Open Source Software

We distribute local approaches in this thesis (LLORMA and LCR) with many other baselines in the toolkit PREA presented in Chapter 3. (`http://prea.gatech.edu`) Researchers both in academia and industry will take advantage of this open source

toolkit. For research scientists, state-of-the-art algorithms implemented in PREA will be great baselines to compare with their new methods. In industry, engineers can easily compare many candidate recommendation algorithms to find the one best fitting to their needs in applications. In some cases, simple and fast algorithm will be the best, while some other applications may allow heavy offline computation to get more accurate estimation.

PREA toolkit (Chapter 3) has been widely used in various areas of research communities, including recommendation systems [99, 95, 96, 97, 98, 14, 63, 62, 116, 113], natural language processing [68], crowd-sourcing [61, 60], optimization [111, 106], computer vision [127], and computer networks [5, 6, 4, 43]. Even outside of computer science and engineering, PREA is used as a tool for matrix factorization in Material Science and Engineering, Aerospace Engineering, and so on.

1.6 Organization

The remainder of this dissertation is organized as follows. We start with reviewing brief history of collaborative filtering and representative methods in Chapter 2. Specifically, we will focus on memory-based (neighborhood-based) methods and matrix-factorization approaches in detail. Also, we review evaluation schemes and metrics widely used in research community.

In Chapter 3, we introduce recommendation algorithms toolkit PREA (Personalized Recommendation Algorithms Toolkit), including why it is needed, how it is structured, and what features it has. In Chapter 4, we compare state-of-the-art CF methods in a standardized manner using the toolkit. The experiment is designed to investigate what algorithms perform better in various situations and with several different evaluation measures.

Chapter 5 describes a CF ensemble model similar to mixture of experts, using dynamic weights depending on input, based on an observation that each CF methods performs better or worse for particular set of users and items from the experimental study. In Chapter 6, we further extend the ensemble model to learn both individual models and their dynamic weights together, by significantly relaxing the low-rank assumption to local models. We present two types of learning algorithms and discuss their theoretical bounds. In Chapter 7, we apply this local low-rank assumption to the ranking problem. We develop an efficient parallel learning algorithm with various ranked loss functions, and present our experimental results on standard datasets.

Chapter 8 reviews related work. Lastly, in Chapter 9, we summarize our work presented in this thesis, concluding with our contributions and future research directions.

Chapter II

REVIEW OF COLLABORATIVE FILTERING

We start by reviewing previous research in recommendation systems and collaborative filtering. Section 2.1 introduces classical recommendation systems, including the very first use of the term *collaborative filtering*. In the following sections, we briefly explain some representative methods both in neighborhood-based and matrix factorization approaches. Lastly, we introduce how recommendation systems are evaluated in Section 2.5.

2.1 Classical Examples

In the early 1990's, the Internet or data communication was not that prevalent. Only few people used dedicated news groups or e-mail services. We start by introducing several pioneering examples of recommendation systems, although some terminologies may differ from those used in today's recommendation systems or machine learning research community.

Tapestry [40] tried to control flood of text information by filtering out unimportant ones, such as spam mails. It was this work that introduced the term *collaborative filtering* for the first time, in the sense that people *collaborate* to help one another perform *filtering*, by recording their reactions to documents they read. Tapestry system is based on databases, using index and its dedicated language TQL (Tapestry Query Language). With this system, users can annotate the usefulness of a particular text, which can be used for other users to filter them out. Figure 2.1 shows architecture of Tapestry system. This is regarded as the first idea of filtering useful information, based on other users' feedback, though it did not discuss detailed algorithm for how to find similar users, or how to recommend in a personalized manner.

GroupLens [109] was another system based on text service, specifically on news. At the time it was proposed, the number of garbage articles was getting explosive with the emergence of Internet on public. This was making difficult to filter useful ones by traditional methods such as manual filtering or splitting threads of articles. The goal of GroupLens was enabling users to predict the quality of news articles before reading

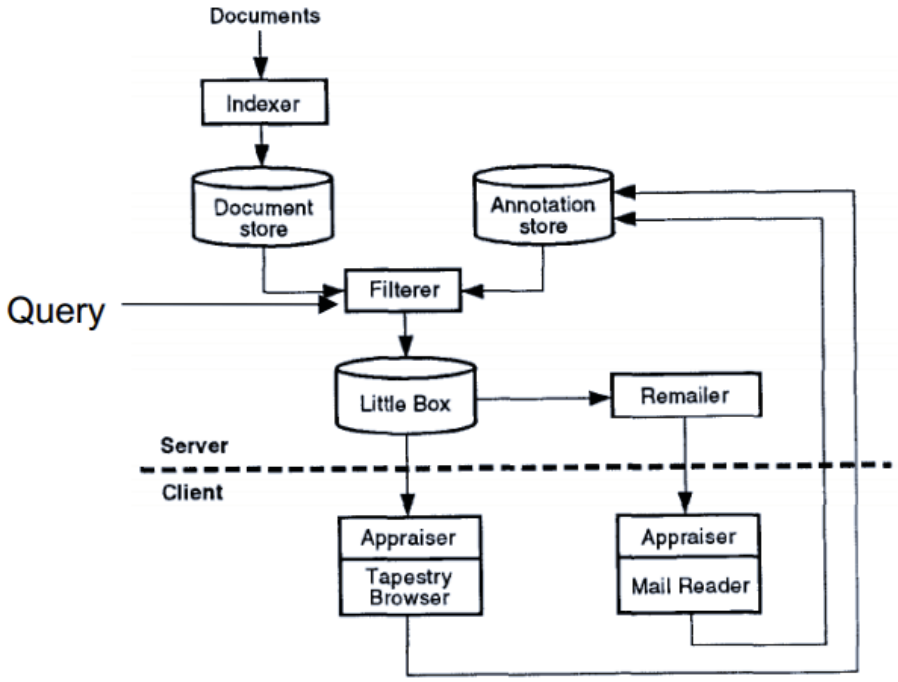

Figure 2.1: System architecture of Tapestry [40], which introduced the concept of collaborative filtering for the first time.

them. GroupLens proposed user-based collaborative filtering, which finds other users who have similar taste, then shows their ratings on the article. Similar users were calculated by Pearson-r correlation, and scores were weighted-averaged based on the similarity between users. (User-based collaborative filtering will be discussed in more detail in Section 2.3.) GroupLens also discussed how to use the calculated score as well as presented its graphical representation. Figure 2.2 illustrates GroupLens system flow.

Virtual community [50] was the first recommender systems on the multimedia (movie) domain. In human-computer interaction perspective, it tried to solve a problem of searching or recommending multimedia data and to develop a user interface for such systems. The goal of virtual community was interacting and influencing other people, without causing communication costs. It suggested four design goals, which are considered important in modern recommendation systems as well: 1) ease of use, 2) confident recommendation, 3) scalability, and 4) generalized framework. By adopting both train/test set validation (widely used in machine learning and data mining community) and user study (widely used in human computer interaction community), it established evaluation schemes used for recommendation systems.

Last example is Ringo [121], a personalized music recommendation system. This

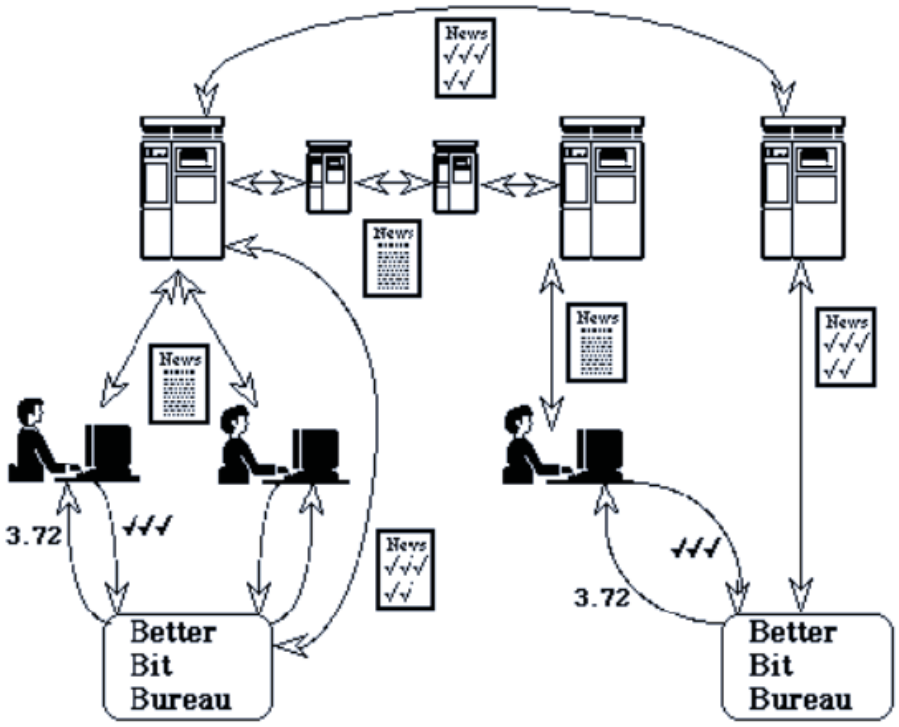

Figure 2.2: System flow of GroupLens [109], a pioneer of memory-based collaborative filtering system.

work is considered as an important example, since it described detailed algorithm as well as various similarity measures and evaluation metrics. Under the assumption that there are general trends and patterns within the taste of a person, it applied user-based collaborative filtering approach. Specifically, it builds and maintains user profiles from their subjective ratings on items. When it processes a query from a user, it looks for other users having similar interests by comparing these profiles. It finally recommends a list of items that those similar users prefer.

2.2 Rating Prediction Problem

Suppose we have n_1 users and n_2 items. Those users can annotate their subjective ratings on items. The set T of possible ratings can be either a binary set (e.g, {Like, Dislike} or {+1, -1}), an ordinal (discrete) set (e.g, {A,B,C,D,F} or $\{1, 2, 3, 4, 5\}$), or a bounded continuous set (e.g, $\{r \in \mathbb{R} : 1 \leq r \leq 5\}$). A rating matrix $M \in T^{n_1 \times n_2}$ contains users in rows $1, ..., n_1$, items in columns $1, ..., n_2$, and ratings in cells. When a user u rates an item i, we *observe* a rating in $M_{u,i}$. Table 2.1 is an example of a rating matrix with 5 users and 5 items with $T = \{1, 2, 3, 4, 5\}$, where higher score is

Table 2.1: An example of rating matrix.

	Item a	Item b	Item c	Item d	Item e
User A	5				2
User B		3		1	4
User C	4		2		2
User D		5		2	
User E	5		1		

considered higher preference. For instance, the user A likes item a very much, but does not like item e.

Since it is almost impossible for a user to experience all items, most recommendation system dataset is extremely sparse. In other words, very small portion of ratings are observed and most are unknown. We define *density* as the ratio of observed ratings to the whole matrix size. For the rating matrix in Table 2.1, for example, its density is

$$d = \frac{2+3+3+2+2}{5 \times 5} = \frac{12}{25} = 48\%. \tag{2.1}$$

Note that density of standard datasets like Netflix or MovieLens ranges around 1% to 5%, as shown in Table 1.1.

Rating prediction problem is predicting the unseen ratings from the sparse rating matrix M, making use of all observed ratings. Semantically, it corresponds to predicting how much the user likes the item. Unlike contents-based systems, we assume that we do not have any contents information such as users' age or gender, items' genre or release year. We describe representative algorithms to solve this problem in next section.

2.3 Neighborhood-based Approaches

The idea of neighborhood-based collaborative filtering stems from an observation that similar users or items share taste. We expect to find interesting items within preferred items of similar users, for example.

This approach can be applied to both user-centric and item-centric views. In a user-centric view, to recommend items to a particular user u, we refer to a set of other users who are similar to u. Inspecting ratings by those similar users, we try to list items which the user u may like but he did not see yet. In an item-centric world, on the other hand, we seek items which are similar to items the user u likes. These two

basic directions construct user-based and item-based collaborative filtering algorithm, described in detail below.

2.3.1 User-based Collaborative Filtering

Suppose we have a rating matrix M, and we are about to predict an unseen rating of item i by user u. In user-based collaborative filtering, we first seek *neighbors* of the user u. For this, we need a similarity measure between two users. At this point, we assume a black box computing similarity between two users. This will be discussed later. By calculating similarity between u and all other users, we can select k most similar users to u. To predict the rating $M_{u,i}$, we use ratings of these k similar users. The easiest way may be the simple average of available ratings on the item i by those similar users. More sophisticated way is an weighted average, based on similarity between the target user u and its neighbors. Formally,

$$\hat{M}_{u,i} = \frac{\sum_{v \in \mathcal{U}} sim(u,v) M_{v,i}}{\sum_{v \in \mathcal{U}} sim(u,v)} \tag{2.2}$$

where $\mathcal{U}$ is the set of similar users with u, and sim is the similarity black box.

Let's illustrate this approach by an example, using Table 2.1. Suppose we are about to predict a rating by user A on item c. Let's assume that the neighbor size $k = 2$, the neighbor user set $\mathcal{U} = \{C, E\}$, and user similarity is $sim(A, C) = 0.8$, $sim(A, E) = 0.6$, respectively. Then, we can compute (2.2) as follows:

$$\hat{M}_{A,C} = \frac{0.8 \times 2 + 0.6 \times 1}{0.8 + 0.6} = \frac{2.2}{1.4} \approx 1.57 \tag{2.3}$$

That is, we predict that the user A would have rated item c as score of 1.57, indicating that he does not really like it. This makes sense semantically, as both neighbor users C and E rated the item c parsimoniously, 2 and 1 points respectively.

Now, the only thing we need is how to create the black box computing similarity between two users. There are several ways to compute this, including the following:

Pearson Correlation

$$sim(u,v) = \frac{\sum_j (M_{u,j} - \bar{M}_u)(M_{v,j} - \bar{M}_v)}{\sqrt{\sum_j (M_{u,j} - \bar{M}_u)^2} \sqrt{\sum_j (M_{v,j} - \bar{M}_v)^2}} \tag{2.4}$$

where $\bar{M}_u$ and $\bar{M}_v$ are the average of ratings by user u and v, respectively, and the summation is only over observed ratings.

Cosine Similarity

$$sim(u, v) = \frac{M_u \cdot M_v}{|M_u||M_v|} \tag{2.5}$$

where M_u and M_v are the row vectors corresponding user u and v, respectively.

Mean Squared Difference

$$sim(u, v) = \frac{1}{|C|} \sum_{i \in C} (M_{u,i} - M_{v,i})^2 \tag{2.6}$$

where C is a set of items rated both by user u and v.

Mean Absolute Difference

$$sim(u, v) = \frac{1}{|C|} \sum_{i \in C} |M_{u,i} - M_{v,i}| \tag{2.7}$$

where C is a set of items rated both by user u and v.

2.3.2 Item-based Collaborative Filtering

Item-based CF [117] resembles user-based CF, only the role of rows and columns is interchanged. To determine the rating of item i by user u, it seeks neighbors of the item i, rather than user u. By calculating similarity between i and all other items, we can select k most similar items to i, just as we did with users in user-based method. The predicted rating can be calculated exactly in the same way:

$$\hat{M}_{u,i} = \frac{\sum_{j \in \mathcal{I}} sim(i, j) M_{u,j}}{\sum_{j \in \mathcal{I}} sim(i, j)} \tag{2.8}$$

where $\mathcal{I}$ is the set of similar items to i and sim is the similarity function between two items. We can use similarity measures in (2.4) – (2.7) using column vectors i and j of rating matrix M, instead of row vectors.

[117] argues item-based approach performs more accurately than user-based counterpart, because item neighborhood relationship tends to be more stable than user relationships. That is, users tend to change their preference as time goes, so the user neighbor group may not be accurate in many cases. (The success of temporal dynamics used by Netflix winner [72] is also based on this observation.) Item relationship, on the other hand, tends to be more stable, since the intrinsic property of items (e.g, genre) does not change as time goes. Item-based approach can take advantage of this in computational aspect as well, by pre-computing item similarity offline.

2.3.3 Extensions

There are some important extensions applicable to memory-based collaborative filtering approaches. [12]

Instead of computing similarity only on observed ratings, we apply *default voting* for unseen ratings. This is effective especially when the rating matrix is very sparse, as we have much smaller chance to find co-rated items or co-rating users. The default value may be determined differently depending on the type of dataset. For example, we may apply neutral value for rating matrix of movies, as we do not know the preference of unseen movies. For a purchase history data, however, we may give negative preference on unseen part, as it may indicate dislike.

Inverse user frequency is based on an observation that universally liked items are not that useful in capturing similarity. Inverse user frequency emphasizes less common items with higher weights, while degrade weights of popular items.

Case amplification is used to emphasize highly-weighted cases further, while make lowly-weighted cases negligible. This is effective to reduce noise in the dataset.

2.3.4 Complexity, Pros and Cons

Both user-based and item-based CF belong to memory-based methods, since it maintains the whole rating matrix in memory and predicts ratings when it is asked. As it does nothing but storing the whole matrix in the memory, training time is negligible. If we pre-calculate user or item similarity offline, user-based CF takes $O(n_1^2 n_2)$ and item-based CF takes $O(n_1 n_2^2)$, respectively. For querying time, it takes $O(n_1)$ searching time for user-based CF and $O(n_2)$ for item-based CF. (Recall that n_1 is the number of users and n_2 is the number of items.)

Memory-based collaborative filtering is simple and easy to understand. Since it uses similar users or items to predict a rating or to generate a recommendation list, it is also easy to *explain* why those items are recommended. This justification elevates credibility of the recommendation system. [129]. However, memory-based approaches are not as accurate as matrix factorization methods discussed in the next section. Also, memory-based CF is not that scalable, so it is not proper for large-scale systems requiring accurate prediction.

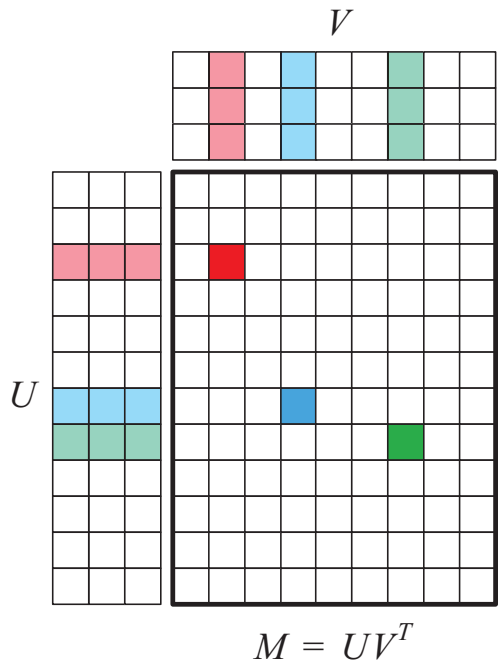

Figure 2.3: An example of matrix factorization.

2.4 Matrix Factorization Approaches

Matrix factorization is another approach to solve rating prediction problem defined in Section 2.2. Recall that we have a sparse rating matrix $M \in \mathbb{R}^{n_1 \times n_2}$, and we are going to fill in missing values in this matrix, based on observed entries.

In matrix factorization, we approximate the given matrix M with a product of two low-rank (say r), dense matrices $U \in \mathbb{R}^{n_1 \times r}$ and $V \in \mathbb{R}^{n_2 \times r}$:

$$M \approx UV^\top \tag{2.9}$$

where $r \ll \min(m, n)$.

Each entry in M is approximated as an inner product of two vectors, one row from U and one row from V. Specifically, $M_{u,i}$ is approximated by $U_u V_i^\top$. Figure 2.3 graphically shows how this works. The red element ($M_{3,2}$), for example, is expressed as the inner-product of two (light) red vectors, corresponding row vector of U and column vector of $V^\top$. Blue and green elements are also expressed in the same way.

Our learning task is determining the value of matrices U and V by enforcing $U_i^\top V_j$ to be similar to $M_{i,j}$ for all observed entries. Although the rating matrix M is sparse, U and V would be dense matrices, provided that we have at least one observation for all users (rows) and all items (columns). As U and V are dense matrices, we can

estimate unseen entries by taking inner-product of two corresponding vectors from U and V. The exact way of learning U and V differs depending on algorithms. We introduce some representative methods below.

2.4.1 Regularized SVD

Singular Value Decomposition (SVD) decomposes a matrix $M \in \mathbb{R}^{n_1 \times n_2}$ into

$$R = U\Sigma V^\top = \sum_{k=1}^{d} \sigma_k u_k v_k^\top \tag{2.10}$$

where $U \in \mathbb{R}^{n_1 \times d}$ is a set of left singular vectors, $V \in \mathbb{R}^{n_2 \times d}$ is a set of right singular vectors, and $\Sigma \in \mathbb{R}^{d \times d}$ is a diagonal matrix with singular values. Without loss of generality, we can sort singular vectors in decreasing order of σ_k. When we take singular vectors corresponding to r largest singular values, we get the best rank-r approximation to M

$$\hat{M} = U\Sigma V^\top = \sum_{k=1}^{r} \sigma_k u_k v_k^\top, \tag{2.11}$$

in terms of least squares.

Our goal in collaborative filtering is the opposite: given M, we want to find U and V, where minimizes squared error between M and $UV^\top$ on observed entries. The objective function $E(U, V)$ we want to minimize is given by

$$E(U,V) = \sum_{(u,i)\in\Omega} \left([UV^\top]_{u,i} - M_{u,i}\right)^2 = \sum_{(u,i)\in\Omega} \left(\sum_{k=1}^{r} U_{u,k}V_{i,k} - M_{u,i}\right)^2. \tag{2.12}$$

where Ω is a set of indices of observed entries. In addition to (2.12), we may need regularization to avoid overfitting. We add $R(U)$ and $R(V)$ to the end of (2.12), preventing values in U and V from being too large. Several different form of regularization can be possible, but we use L_2-regularization (ridge regression) to make $E(U, V)$ differentiable. The modified objective function is then

$$E(U,V) = \sum_{(u,i)\in\Omega} \left([UV^\top]_{u,i} - M_{u,i}\right)^2 + \lambda R(U) + \lambda R(V) \tag{2.13}$$

$$= \sum_{(u,i)\in\Omega} \left(\sum_{k=1}^{r} U_{u,k}V_{i,k} - M_{u,i}\right)^2 + \lambda \sum_{u=1}^{m}\sum_{k=1}^{r} U_{u,k}^2 + \lambda \sum_{i=1}^{n}\sum_{k=1}^{r} U_{i,k}^2. \tag{2.14}$$

where λ controls relative importance between least square fitting and regularization terms. (Note that we can use different regularization coefficient for U and V.)

To minimize (2.14), we use gradient-descent, since least square problem of a partially-observed matrix is non-convex. Algorithm 2.1 describes gradient-descent framework. Considering a single term in $E(U, V)$, we have

$$E_{u,i}(U,V) = \left(\sum_{k=1}^{r} U_{u,k}V_{i,k} - M_{u,i}\right)^2 + \lambda\sum_{k=1}^{r} U_{u,k}^2 + \lambda\sum_{k=1}^{r} U_{i,k}^2. \tag{2.15}$$

Partial derivatives of (2.15) with respect to single elements in U and V are given by

$$\frac{\partial E_{u,i}(U,V)}{\partial U_{u,k}} = 2V_{i,k}\left([UV^\top]_{u,i} - M_{u,i}\right) + 2\lambda V_{i,k} \tag{2.16}$$

$$\frac{\partial E_{u,i}(U,V)}{\partial V_{i,k}} = 2U_{u,k}\left([UV^\top]_{u,i} - M_{u,i}\right) + 2\lambda U_{u,k} \tag{2.17}$$

which are used for updating in Algorithm 2.1. The parameter μ in Algorithm 2.1 controls learning speed. It runs until $UV^\top$ converges to M over observed entries Ω. Due to the non-convexity of the problem, the converged U and V may depend on initial values.

Algorithm 2.1 Regularized SVD

Input: $M \in \mathbb{R}^{n_1 \times n_2}, \mu$
while $\|M - UV^\top\| > \epsilon$ **do**
  for all $(u, i) \in \Omega, k = 1, \cdots, r$ **do**
    $U_{u,k} := U_{u,k} - \mu\frac{\partial E_{u,i}(U,V)}{\partial U_{u,k}}$
    $V_{i,k} := V_{u,k} - \mu\frac{\partial E_{u,i}(U,V)}{\partial V_{i,k}}$
  end for
end while
Output: U, V

2.4.2 Non-negative Matrix Factorization

Non-negative matrix factorization (NMF) is a matrix factorization technique approximating $M \approx UV^\top$ under the constraint that all entries in U and V should be non-negative. This requirement can be important in some applications where the representation of each element is inherently non-negative, or it seeks low-rank matrices which are enforced to have only non-negative values. For example, a text document is represented as a vector of non-negative numbers with the term-frequency encoding. Each element in this representation is the number of appearances of each term in the document, so it is non-negative. Another example is image processing. Digital images are represented by a matrix of pixel intensities, which is inherently

non-negative. In natural sciences such as chemistry or biology, chemical concentrations or gene expressions are also non-negative. [67] In recommendation systems, a rating matrix is also usually non-negative. Although other matrix factorization methods may allow negative entries in factorized low-rank matrices, it still makes sense to enforce non-negativity as the original data is non-negative.

Objective function which is minimized in NMF is either Euclidean distance

$$E(U,V) = \sum_{(u,i)\in\Omega} \left([UV^\top]_{u,i} - M_{u,i}\right)^2, \tag{2.18}$$

or Kullback-Leibler divergence

$$E(U,V) = \sum_{(u,i)\in\Omega} \left(M_{u,i} \log \frac{M_{u,i}}{[UV^\top]_{u,i}} - M_{u,i} + [UV^\top]_{u,i} \right). \tag{2.19}$$

Lee and Seung [79, 119] present multiplicative update rules for both Euclidean distance and Kullback-Leibler divergence objective functions. For Euclidean distance, the update rules are

$$V_{i,k} \leftarrow V_{i,k} \frac{[U^\top M]_{i,k}}{[U^U V^\top]_{i,k}}, \tag{2.20}$$

$$U_{u,k} \leftarrow U_{u,k} \frac{[MV]_{u,k}}{[UV^\top V]_{u,k}}. \tag{2.21}$$

For Kullback-Leibler divergence, the update rules are

$$V_{i,k} \leftarrow V_{i,k} \frac{\sum_u U_{u,k} M_{u,i} / [UV^\top]_{u,i}}{\sum_u U_{u,k}}, \tag{2.22}$$

$$U_{u,k} \leftarrow U_{u,k} \frac{\sum_i V_{i,k} M_{u,i} / [UV^\top]_{u,i}}{\sum_i V_{i,k}}, \tag{2.23}$$

which were proved to converge in [119]. Note that these multiplicative update rules are possible thanks to the non-negativity constraints, and converge faster than additive update rules.

2.4.3 Probabilistic Models

Matrix factorization based on SVD was extended to probabilistic models by two seminal papers [115, 114]. Probabilistic matrix factorization (PMF) [115] is represented as a graphical model in the left panel of Figure 2.4. Assuming Gaussian observation noise, conditional distribution over the observed ratings is given by

$$p(M|U,V,\sigma^2) = \prod_{u=1}^{n_1} \prod_{i=1}^{n_2} \left[\mathcal{N}(M_{u,i}|U_u^\top V_i, \sigma^2)\right]^{I_{u,i}} \tag{2.24}$$

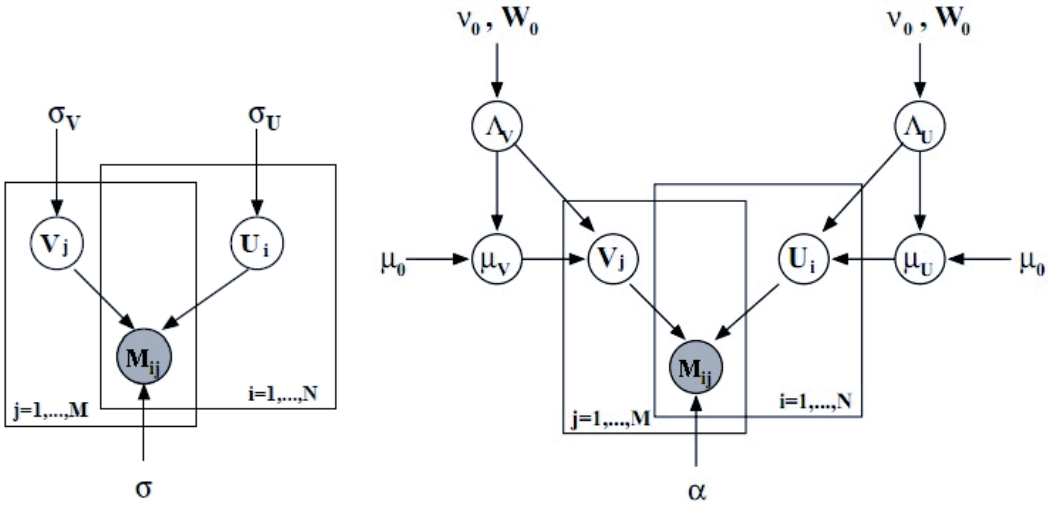

Figure 2.4: Matrix factorization using graphical models. *Left:* Probabilistic matrix factorization (PMF) [115], *Right:* Bayesian probabilistic matrix factorization (BPMF) [114].

where $I_{u,i}$ is 1 if $(u, i) \in \Omega$, and 0 otherwise. Priors are also assumed to have zero-mean spherical Gaussian distributions,

$$p(U|\sigma_U^2) = \prod_{u=1}^{n_1} \mathcal{N}(U_u|0, \sigma_U^2 \mathbf{I}) \tag{2.25}$$

$$p(V|\sigma_V^2) = \prod_{i=1}^{n_2} \mathcal{N}(V_i|0, \sigma_V^2 \mathbf{I}) \tag{2.26}$$

Putting them together, the log of the posterior distribution is given by

$$\begin{aligned} &\ln p(U, V|M, \sigma^2, \sigma_U^2, \sigma_V^2) \\ &= -\frac{1}{2\sigma^2} \sum_{u=1}^{n_1} \sum_{i=1}^{n_2} I_{ui}(M_{u,i} - U_u^\top V_i)^2 - \frac{1}{2\sigma_U^2} \sum_{u=1}^{n_1} U_u^\top U_u - \frac{1}{2\sigma_V^2} \sum_{i=1}^{n_2} V_i^\top V_i \\ &\quad - \frac{1}{2} \left(\left(\sum_{u=1}^{n_1} \sum_{i=1}^{n_2} I_{u,i} \right) \ln \sigma^2 + mr \ln \sigma_U^2 + nr \ln \sigma_V^2 \right) + C, \end{aligned} \tag{2.27}$$

where r is the rank of low-rank approximation, and C is a constant that does not depend on the parameters. If we fix hyperparameters $\{\sigma_U,\ \sigma_V\}$ as constants, the log-posterior shrinks to objective function of SVD (2.15).

Although it looks similar to SVD, PMF extends SVD with probabilistic interpretation. In SVD, regularization parameters were chosen somehow by hand. [115] proposed type II maximum likelihood estimator for choosing the regularization parameters. This leads to natural extension to fully-Bayesian treatment, introduced in Bayesian

PMF [114]. Unlike PMF using point estimates, Bayesian PMF (BPMF) estimates the whole posterior probability distribution. The graphical model used in BPMF is shown in the right panel of Figure 2.4.

As in (2.24), the likelihood of the observed ratings are given by

$$p(M|U,V,\sigma^2) = \prod_{u=1}^{n_1}\prod_{i=1}^{n_2}\left[\mathcal{N}(M_{u,i}|U_u^\top V_i,\sigma^2)\right]^{I_{u,i}}. \tag{2.28}$$

The prior distributions over U and V are extended to have hyperparameters instead of zero-mean and fixed precision in PMF:

$$p(U|\mu_U,\Lambda_U) = \prod_{u=1}^{n_1}\mathcal{N}(U_u|\mu_U,\Lambda_U^{-1}\mathbf{I}) \tag{2.29}$$

$$p(V|\mu_V,\Lambda_V) = \prod_{i=1}^{n_2}\mathcal{N}(V_i|\mu_V,\Lambda_V^{-1}\mathbf{I}) \tag{2.30}$$

where Λ_U and Λ_V are precisions, corresponding to the inverse of variances σ_U^2 and σ_V^2 used in PMF. BPMF further introduces Gaussian-Wishart priors over user and item hyperparameters $\Theta_U = \{\mu_U,\Lambda_U\}$ and $\Theta_V = \{\mu_V,\Lambda_V\}$:

$$p(\Theta_U|\Theta_0) = \mathcal{N}(\mu_U|\mu_0,(\beta_0\Lambda_U)^{-1})\mathcal{W}(\Lambda_U|\Lambda_0,\nu_0) \tag{2.31}$$

$$p(\Theta_V|\Theta_0) = \mathcal{N}(\mu_V|\mu_0,(\beta_0\Lambda_V)^{-1})\mathcal{W}(\Lambda_V|\Lambda_0,\nu_0) \tag{2.32}$$

where $\mathcal{W}$ is Wishart distribution, and $\Theta_0 = \{\mu_0,\nu_0,W_0\}$ is its parameters. The predictive distribution for unseen rating $\hat{M}$ is given by

$$p(\hat{M}_{u,i}|M,\Theta_0) = \iint p(\hat{M}_{u,i}|U_u,V_i)p(U,V|M,\Theta_U,\Theta_V)p(\Theta_U,\Theta_V|\Theta_0)d\{U,V\}d\{\Theta_U,\Theta_V\}. \tag{2.33}$$

As exact evaluation of (2.33) is analytically intractable, [114] used MCMC (Markov Chain Monte-Carlo)-based sampling for approximate inference. With Gibbs sampling, (2.33) can be approximated by

$$p(\hat{R}_{ui}|R,\Theta_0) \approx \frac{1}{K}\sum_{k=1}^{K} p(\hat{M}_{u,i}|U_u^{(k)},V_i^{(k)}). \tag{2.34}$$

Figure 2.5 shows performance of BPMF with $r = 30$ and $r = 60$. As indicated in the figure, it takes relatively long time to achieve good test RMSE.

2.4.4 Other Matrix Factorization Models

Besides several methods we introduced so far, there are hundreds of variations and extensions on matrix factorization methods for collaborative filtering. We list some of them before finishing this section.

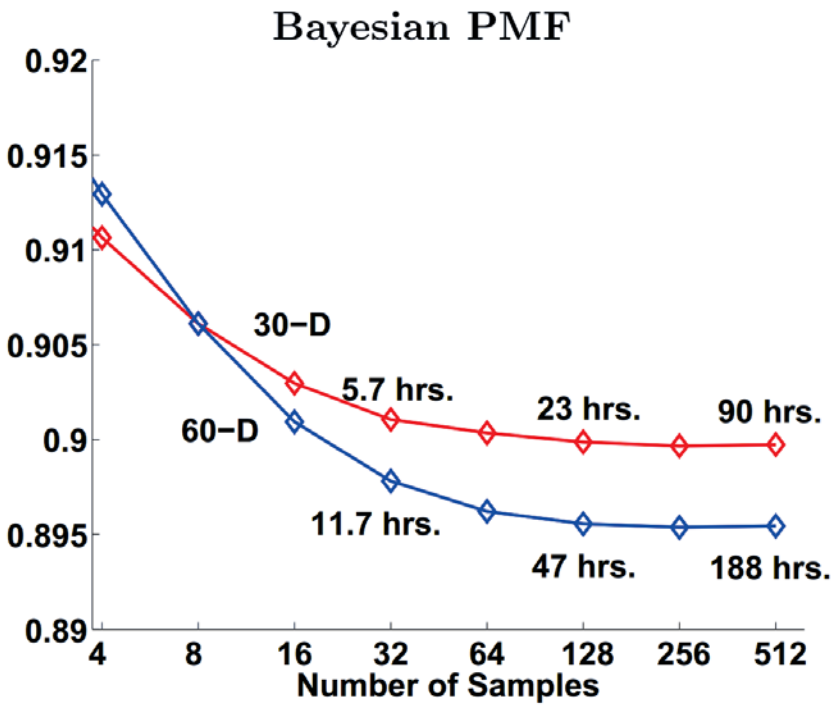

Figure 2.5: Accuracy and computational overhead of Bayesian probabilistic matrix factorization (BPMF) [114].

Maximum Margin Matrix Factorization (MMMF) [124] resembles maximum margin classifiers such as support vector machines (SVM). It utilizes low-norm, instead of low-rank, factorizations by solving a semi-definite optimization problem. [108] improved original MMMF to faster and more scalable version, by a direct gradient-based optimization algorithm. [29] proposed ensembles of MMMF models, achieving better prediction accuracy.

Non-linear Matrix Factorization (NLPMF) [75] proposed a non-linear model for matrix factorization using Gaussian process latent variable models. Applying stochastic gradient descent, NLPMF is scalable to millions of observations without approximate methods. Rating prediction is done by a similar way with neighborhood-based methods discussed in Section 2.3.

Fast Non-parametric Matrix Factorization [147] developed a fast optimization algorithm for non-parametric matrix factorization method. Non-parametric models can be preferred to parametric low-rank models thanks to its flexibility, but its drawback is computational overhead and limited scalability. Fast Non-parametric Principal Component Analysis (NPCA) algorithm turns out to be efficient as well as achieves good accuracy.

2.5 Evaluation of Recommendation Systems

Recommendation systems can be evaluated in various ways. When a system is deployed as a real application, it may be tested with real users on a real situation. That is, one or more recommendation engines are implemented on a target system, and each user is directed to test those recommenders. From the system log, we can observe how users are influenced by the system. We may also survey them directly. This kind of user study is widely used in human computer interaction (HCI) or computer-supported cooperative work (CSCW) research community. However, user study is very costly in general. Also, we need to take burden to open imperfect version to real users. This may give negative experience, making them to avoid using the system in the future.

Instead of online experiment with real users, we can do offline experiments using previously gathered dataset. In this section, we focus on evaluation of *recommendation algorithms*, in machine learning and data mining perspective. We introduce various evaluation metrics and strategies which have been used in recommendation system research community [49, 41] as well as which is used throughout this thesis.

2.5.1 Metrics for Prediction Accuracy

When our main task is predicting unseen ratings, we usually want to quantify how accurate the predicted rating is. Formally, we define a recommendation system $f(u, i)$ as a function, returning predicted rating for item i by user u. For a set of indices Ω of m test entries, we compare the predicted rating and the observed true rating. This test set is not supposed to be seen by the recommendation system before it finishes learning phase. The three most widely-used metrics include

Mean Absolute Error (MAE)

$$MAE = \frac{1}{m} \sum_{(u,i)\in\Omega} |f(u, i) - M_{u,i}|, \tag{2.35}$$

where $f(u, i)$ is our prediction on item i by user u, and $M_{u,i}$ is the ground truth,

Normalized Mean Absolute Error (NMAE)

$$NMAE = \frac{MAE}{r_{max} - r_{min}}, \tag{2.36}$$

where r_{max} and r_{min} indicate the maximum and minimum possible rating in the dataset, respectively. This measure surely scales between $[0, 1]$, and

Root Mean Squared Error (RMSE)

$$RMSE = \sqrt{\frac{1}{m} \sum_{(u,i)\in\Omega} (f(u,i) - M_{u,i})^2}. \tag{2.37}$$

$RMSE$ tends to give more penalty to big mistakes than MAE. Traditionally, MAE was used in general. Since the Netflix prize, $RMSE$ has become more popular.

Asymmetric Measures

In addition to these three metrics, [80] introduced asymmetric measures, which have been widely used in classification problems. Asymmetric losses reflect the idea that false-positive and false-negative (See Table 2.2.) may result in quite different situations. For example, diagnosing a patient with cancer as normal brings about much serious result than its opposite. In recommendation systems, the same problem can happen. Specifically, recommending an item which is not preferable may disgrace the trustworthiness more than missing an item which is preferable.

Formally, the metrics above can be generalized to the form

$$ASYM = \frac{1}{m} \sum_{(u,i)\in\Omega} L(f(u,i), M_{u,i}) \tag{2.38}$$

where L is defined by a matrix or a function returning a real number. Based on the spirit above, an example of asymmetric loss can be defined as

$$L(x,y) = \begin{bmatrix} 0 & 0 & 0 & 3 & 4 \\ 0 & 0 & 0 & 2 & 3 \\ 0 & 0 & 0 & 1 & 2 \\ 6 & 4 & 2 & 0 & 0 \\ 8 & 6 & 4 & 0 & 0 \end{bmatrix}. \tag{2.39}$$

$L(x,y)$ in (2.39) indicates the penalty when the system predicts a rating as x (row) for real rating y (column), on a system with ordinal ratings between 1 and 5. For instance, predicting 1 as 2 or 2 as 1 does not give penalty, since the item is not recommended to the user. This is same for the case of 4 as 5 or 5 as 4 as they are still recommended after all. When we predict 4 as 1, $L(x,y)$ gives penalty corresponding to their difference as MAE. However, if it is the opposite way, it gives twice penalty of that, as we believe recommending bad items will cause more serious problem. This kind of loss matrix or loss function can be defined arbitrarily based on our belief. We observe that MAE or RMSE can be seen as a special case of this asymmetric measure. Since all of these measures are *errors*, lower values mean better performance.

Table 2.2: Classification matrix for recommendation systems.

	Recommended	Not-recommended
Preferred	True-Positive (TP)	False-Negative (FN)
Not-preferred	False-Positive (FP)	True-Negative (TN)

2.5.2 Metrics for Classification Accuracy

Another way of evaluating recommendation algorithm is viewing recommendation process as a classification problem. Based on the ground truth, we can classify each rating as one of $\{\text{Preferred}, \text{Not-preferred}\}$. We want to *recommend* preferred items, but *not to recommend* not-preferred. Among the four cases in Table 2.2, TP and TN are considered correct. Using the table, there are two important measures for accuracy of classifiers:

Precision measures the ratio of correct recommendations among the list of recommended items. If we have 6 preferred items by the user in the recommended list of 10 items, the precision is 0.6, for example.

$$Prec = \frac{\{\text{Recommended} \bigcap \text{Preferred}\}}{\{\text{Recommended}\}} = \frac{TP}{TP + FP} \tag{2.40}$$

Recall measures the ratio of the user's preferred items covered by the recommendation list. If there are 20 preferred items by the user in the system and the system suggests 5 of them, the recall is 0.25, regardless of the length of recommendation list.

$$Recall = \frac{\{\text{Recommended} \bigcap \text{Preferred}\}}{\{\text{Preferred}\}} = \frac{TP}{TP + FN} \tag{2.41}$$

Recall has a property that it is monotonically non-decreasing with the increase of recommendations. An extreme silly recommender suggesting all items to all users always achieves highest recall. However, increasing the number of recommended items in general reduce its precision. In other words, there is a trade-off between precision and recall.

F1 is a good measure taking both of them into account:

$$F1 = \frac{2 \cdot Prec \cdot Recall}{Prec + Recall} \tag{2.42}$$

The higher all of these three measures are, the better the recommender system is.

2.5.3 Metrics based on Ranks

Rank-based Metrics use relative order of recommended items rather than exact scores of them. Below is a list of rank-based measures, which are also used in web search engine evaluation.

Half-Life Utility Score (HLU) [12] is one of the most popular metrics based on ranking. Suppose we recommend an ordered list of items to a user. Naturally, we would expect that the user will see the list from the top to the bottom, and he may stop either when he finds an interesting one or when he feels the list is not useful to him. We want to estimate how likely the user will actually retrieve the recommended items. Assuming an exponential decay with a half-life α, HLU evaluates how many actually preferred items are located in the front of the recommended list. Overall score is in relative scale to the maximum possible score of the target user. Formally, an HLU score HLU_u for user u and overall score HLU is given by

$$HLU_u = \sum_{j=1}^{n_1} \frac{1}{2^{(idx(j)-1)/(\alpha-1)}} \tag{2.43}$$

$$HLU = \frac{\sum_u HLU_u}{\sum_u HLU_u^{max}} \tag{2.44}$$

where $idx(j)$ is the index of the item j in the recommended list, and α is the half-life, the number of items in the list where the probability of retrieval drops to half. For example, if $\alpha = 5$, we assume that users will see the item ranked in the 6th position with half probability of the one in the 1st position. HLU_u^{max} is the maximum possible score for the user u. This is achieved when the recommended items are sorted by the user's actual preference. [41] argues that this is a proper measure when the recommender system is aimed to optimize utility.

Normalized Discounted Cumulative Gain (NDCG) is a popular metric for measuring effectiveness of a web search engine or other information retrieval applications. Discounted Cumulative Gain (DCG) measures the usefulness of an item based on its rank in a recommended list. NDCG normalizes the DCG in order to compensating varying length of recommendation list. For this, DCG score is divided by maximum possible gain with the given list. This is conceptually similar to HLU, but gives slightly different result. Formal definition of DCG and NDCG with a list of p

recommendations is given by

$$DCG_u = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2 i} \tag{2.45}$$

$$NDCG_u = \frac{DCG_u}{DCG_u^{max}} \tag{2.46}$$

where rel_i is the relevance of the item in the index i in the recommended list to the target user. NDCG is the normalized version of DCG by maximum possible one. This DCG_u^{max} is also achieved when the recommendation list is sorted by the user's actual preference.

Kendall's Tau is a distance measure, representing the number of discordant pairs among m items. Formally,

$$KT_u = \sum_{i=1}^{m} \sum_{j<i} I[(f(u,i) - f(u,j))(M_{u,i} - M_{u,j}) > 0] \tag{2.47}$$

where $F(u,i)$ is our prediction of rating on item i by user u, and $M_{u,i}$ is the ground truth of it. I is an indicator function returning 1 if the condition is true, and 0 otherwise.

Spearman's distance is the square of Euclidean distance between two rankings interpreted as vectors. Formally, for a predicted ranking vector x and a ground truth ranking vector y over m items, Spearman's distance is

$$SP_u = \sum_{i=1}^{m} (x_i - y_i)^2. \tag{2.48}$$

Before finishing this section, we relate the two important categories of evaluation: rating prediction accuracy (Section 2.5.1) and ranking-based metrics (Section 2.5.3). One question we may have is whether reducing RMSE is really meaningful to users. This is a natural question, since reduction in RMSE seems almost negligible. For instance, the Netflix prize winner achieved RMSE of 0.8567, but so many participants failed to win the million dollar prize as their RMSE were slightly larger than 0.8567, such as 0.86 or so.

[72] conducted an interesting experiment to verify that reducing RMSE is actually a meaningful task, in top-k recommendation situation. Assuming that we recommend k items to users, the order is important as we discussed in 2.5.3. Our goal in recommendation is therefore locating interesting items on the top (or relatively high rank) of the list. The experiment measured the place of interesting items (observed as

highest score) within the recommendation list among 1,000 randomly selected items. Comparing the location of interesting items in recommended list, [72] found that a system with lower RMSE actually locates interesting items on higher rank in average. This experiment proves lots of efforts the research community made on reducing RMSE were actually meaningful in practice.

Chapter III

RECOMMENDATION ALGORITHMS TOOLKIT

In this chapter, we introduce our recommendation algorithm toolkit named PREA (Personalized Recommendation Algorithms Toolkit)[1], which is publicly available as an open-source software.

3.1 Motivation

A wide variety of algorithms have been proposed by the research community for recommendation systems. Unlike classification where comprehensive packages are available, existing recommendation systems toolkits lag behind. They concentrate on implementing traditional algorithms rather than the rapidly evolving state-of-the-art. Implementations of modern algorithms are scattered over different sources which makes it hard to have a fair and comprehensive comparison.

Our PREA implements a wide variety of recommendation systems algorithms. PREA offers implementation of modern state-of-the-art recommendation algorithms as well as most traditional ones. In addition, it provides many popular evaluation methods and datasets. As a result, it can be used to conduct fair and comprehensive comparisons between different recommendation algorithms. The implemented evaluation routines, datasets, and the open-source nature of PREA makes it easy for third parties to contribute additional implementations.

Not surprisingly, the performance of recommendation algorithms depends on the data characteristics. For example, some algorithms may work better or worse depending on the amount of missing data (sparsity), distribution of ratings, and the number of users and items. It is likely that an algorithm may perform better on one dataset and worse on another. Furthermore, the different evaluation methods that have been proposed in the literature may have conflicting orderings over algorithms [41, 80]. PREA's ability to compare different algorithms using a variety of evaluation metrics may clarify which algorithms perform better under what circumstance.

[1] http://prea.gatech.edu/

3.2 Program Structure and Implementation

PREA is a multi-platform Java Software (version 1.6 or higher required). It is compatible with MS Windows, Linux, and Mac OS. The toolkit consists of three groups of classes, as shown in Figure 3.1. The top-level routines of the toolkit may be directly called from other programming environments like Matlab. The top group implements recommendation algorithms. The bottom-left group contains basic data structures and data management part. The last group (bottom-right) implements evaluation metrics, statistical distributions, and other useful functions.

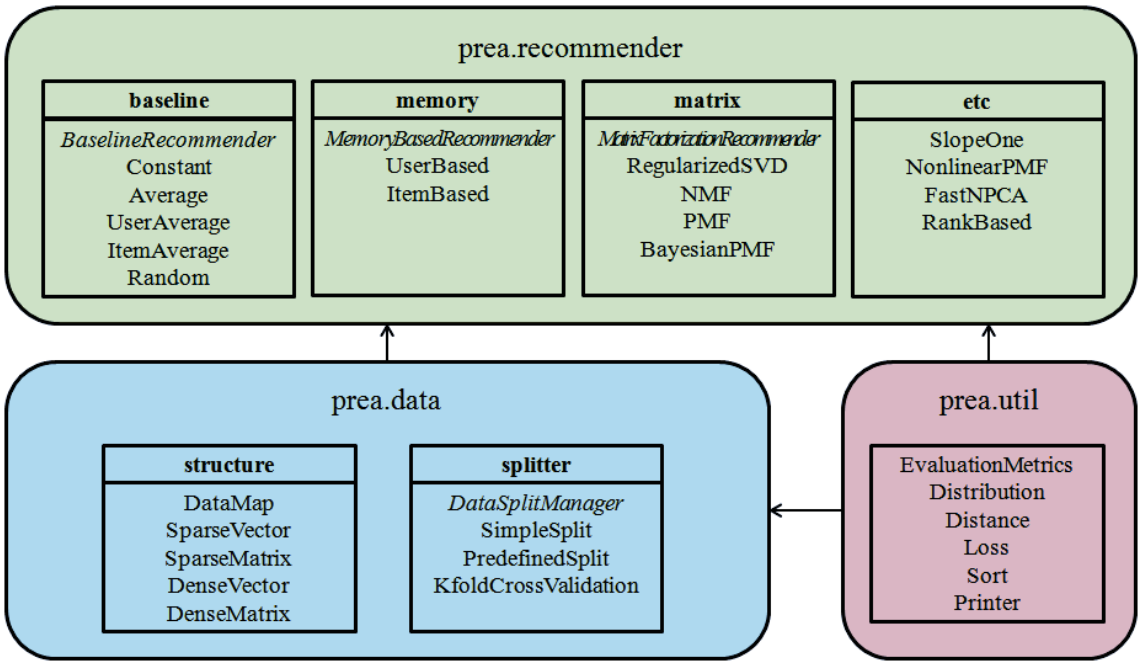

Figure 3.1: Class Structure of PREA Toolkit

The input data file format of PREA is ARFF[2], which is used in the widely-used machine learning toolkit WEKA[3].

Since users provide ratings for only a small subset of items in general, virtually all recommendation systems datasets are sparse. For this reason, PREA implements sparse (rather than dense) ARFF format. Also, data representation in the toolkit uses a sparse matrix representation for the rating matrices. Specifically, we use Java's data structure *HashMap* to create a *DataMap* class and a *SparseVector* class. We tried some other options including *TreeMap*, but we chose *HashMap* due to its superior performance over others. Figure 3.2 (left) shows an example of a sparse vector,

[2]Full description about ARFF can be found at http://www.cs.waikato.ac.nz/~ml/weka/arff.html
[3]http://www.cs.waikato.ac.nz/~ml/weka/

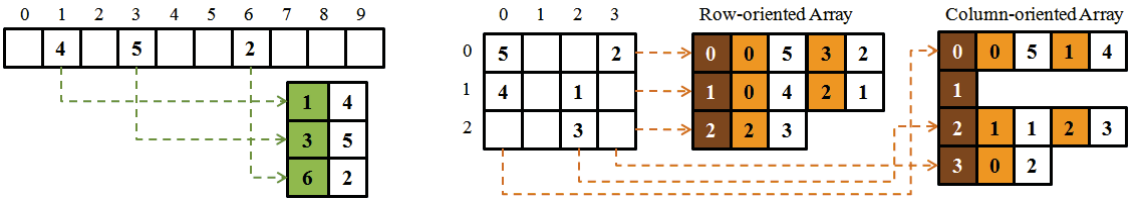

Figure 3.2: Sparse Vector (left) and Matrix (right) Implementation

containing data only in indices 1, 3, and 6.

We construct a *SparseMatrix* class using an array of *SparseVector* objects. To facilitate fast access to rows and columns, both row-oriented and column-oriented reference arrays are kept. This design is also useful for fast transposing of sparse matrices, simply done by interchanging rows and columns. Figure 3.2 (right) shows an example of sparse matrix.

PREA also uses dense matrices in some cases. For example, dense representations are used for low-rank matrix factorizations or other algebraic operations that do not maintain sparsity. The dense representations are based on the matrix implementations in the Universal Java Matrix Package (UJMP)[4].

Below is the list of rating prediction algorithms in PREA:

- Baselines (Constant, Random, User/item Average): make little use of personalized information for recommending items.
- Memory-based Neighborhood algorithms (User-based, item-based collaborative filtering and their extensions including default vote, inverse user frequency): predict ratings of unseen items by referring those of similar users or items.
- Matrix Factorization methods (SVD, NMF, PMF, Bayesian PMF, Non-linear PMF): build low-rank user/item profiles by factorizing training dataset with linear algebraic techniques.
- Others: recent state-of-the-art algorithms such as Fast NPCA and Rank-based collaborative filtering.

We provide popular evaluation metrics as follows:

- Accuracy for rating predictions (RMSE, MAE, NMAE): measure how much the predictions are similar to the actual ratings.

[4]http://www.ujmp.org/

- Rank-based evaluation metrics (HLU, NDCG, Kendall's Tau, and Spearman): score depending on similarity between orderings of predicted ratings and those of ground truth.

3.3 Evaluation Framework

For ease of comparison, PREA provides the following unified interface for running the different recommendation algorithms.

- Instantiate a class instance according to the type of the recommendation algorithm.
- Build a model based on the specified algorithm and a training set.
- Given a test set, predict user ratings over unseen items.
- Given the above prediction and held-out ground truth ratings, evaluate the prediction using various evaluation metrics.

Note that lazy learning algorithms like the constant models or memory-based algorithms may skip the second step.

The train and test split can be constructed in various ways. [41] introduced four strategies for splitting training and test set: (a) predicting each test case based on other ratings prior to that, (b) using ratings after a specific date as test set, (c) splitting the ratings into a fixed proportion of train and test set for each user, or (d) sampling some portion of ratings uniformly randomly, regardless of date, user, or item.

Moving from (a) to (d) loses reality of simulation, being simpler and faster. We implemented train/test set split with option (d). Although it seems to lose some reality, we can actually apply this method without much distortion, as we can assume that individual preference is not changing so fast in many applications. Also, this method is the only choice when the dataset does not provide the rated date information. For more general usage, we split the set without using date information.

3.4 Contribution

PREA contributes to recommendation system research community and industry by (a) providing an easy and fair comparison among most traditional recommendation algorithms, (b) supporting state-of-the-art recommendation algorithms which have

not been available on other toolkits, and (c) implementing different evaluation criteria emphasizing different performance aspects rather than a single evaluation measure like mean squared error.

The open-source nature of the software (available under the Berkeley Software Distribution (BSD) license) may encourage other recommendation systems experts to add their own algorithms to PREA. More documentation for developers as well as user tutorials are available on the web (`http://prea.gatech.edu`).

Table 3.1 compares PREA with other open-source recommendation toolkits. Mahout[5], which incorporated Taste[6], provides an implementation of many memory-based algorithms. It also supports powerful mathematical and statistical operations as it is a general-purpose machine learning toolkit. Duine[7] provides a way to combine multiple algorithms into a hybrid system and also addressed the cold-start situation. Cofi[8] implements several traditional algorithms with a simple design based on providing wrappers for publicly available datasets. MyMedia[9] is a C#-based recommendation toolkit which supports most traditional algorithms and several evaluation metrics.

As indicated in Table 3.1, existing toolkits widely provide simple memory-based algorithms, while recent state-of-the-art algorithms are often not supported. Also, these toolkits are generally limited in their evaluation metrics (with the notable exception of MyMedia). In contrast, PREA provides a wide coverage of the the most up-to-date algorithms as well as various evaluation metrics, facilitating a comprehensive comparison between state-of-the-art and newly proposed algorithms in the research community.

[5]https://cwiki.apache.org/confluence/display/MAHOUT/Recommender+Documentation
[6]http://taste.sourceforge.net/old.html
[7]http://www.duineframework.org/
[8]http://www.nongnu.org/cofi/
[9]http://www.ismll.uni-hildesheim.de/mymedialite/

Table 3.1: Features Comparison with other Collaborative Filtering Toolkits

Category	Feature	PREA	Mahout	Duine	Cofi	MyMedia
Baselines	Constant	O			O	O
	User/Item Average	O	O	O	O	O
	Random	O				O
Memory-based CF	User-based CF [126]	O	O	O	O	O
	Item-based CF [117]	O	O	O	O	O
	Default Vote, Inv-User-Freq [12]	O	O			
	Slope-One [82]	O	O			O
Matrix Factorization	SVD [102]	O	O		O	O
	NMF [79]	O				
	PMF [115]	O				
	Bayesian PMF [114]	O				
	Non-linear PMF [75]	O				
Other methods	Fast NPCA [147]	O				
	Rank-based CF [128]	O				
Evaluation Metric	(N)MAE	O	O	O	O	O
	RMSE	O	O		O	O
	HLU/NDCG	O				O
	Kendall's Tau, Spearman	O				
	Precision/Recall/F1		O			O
Miscellaneous	Sparse Vector/Matrix	O	O	O	O	O
	Wrapper for other languages	O			O	O
	Graphical User Interface		O			O
	Release Year	2011	2005	2009	2004	2009
	Language	Java	Java	Java	Java	C#
	License	BSD	LGPL	LGPL	GPL	GPL

Chapter IV

EMPIRICAL ANALYSIS OF COLLABORATIVE FILTERING

In this chapter, we use PREA introduced in Chapter 3 to compare several collaborative filtering techniques – both classic and recent state-of-the-art – in a variety of experimental contexts. Specifically, we compare their performance on various number of items, number of users, and sparsity level, using a number of performance criteria. Our conclusions identify what algorithms work well and in what conditions.

4.1 Experimental Design

Table 4.1 lists recommendation algorithms to be compared. There are three elementary baselines: a constant function (identical prediction for all users and all items), user average (constant prediction for each user-based on their average ratings), and item average (constant prediction for each item-based on their average ratings). The memory-based methods listed in Table 4.1 are classical methods that perform well and are often used in commercial settings. The methods listed under the matrix factorization and others categories are more recent state-of-the-art methods proposed in the research literature.

In our experiments we used the Netflix dataset, a standard benchmark in the CF literature that is larger and more recent than alternative benchmarks. To facilitate measuring the dependency between prediction accuracy and dataset size and density, we sorted the rating matrix so that its rows and columns are listed in order of descending density level. We then realized specific sparsity pattern by selecting the top k rows and ℓ columns and subsampling to achieve the required sparsity.

Figure 4.1 shows the density level of the sorted rating matrix. For instance, the top right corner of the sorted rating matrix containing the top 5,000 users and top 2,000 items has 52.6% of density. In other words, there 47.4% of the ratings are missing. The density of the entire dataset is around 1%. We subsample a prescribed level of density which will be used for training as well as 20% more for the purpose of testing. We cross-validate each experiment 10 times with different train-test splits.

Table 4.1: List of Recommendation Algorithms used in Experiments

Category	Algorithms
Baseline	Constant User Average Item Average
Memory -based	User-based [126] User-based with Default Values [12] Item-based [117] Item-based with Default Values
Matrix Factorization -based	Regularized SVD [102] NMF [79] PMF [115] Bayesian PMF [114] Non-linear PMF [75]
Others	Slope-One [82] NPCA [147] Rank-based CF [128]

The experiments were conducted on a dual Intel Xeon X5650 processor (6 Core, 12 Threads, 2.66GHz) with 96GB of main memory.

4.2 Observations

4.2.1 Dependency on Data Size and Density

We start by investigating the dependency of prediction accuracy on the dataset size, in terms of user count and item count, and on density of observed ratings. Of particular interest, is the variability in that dependency across different CF algorithms. This variability holds the key to determining which CF algorithms should be used in a specific situation. We start below by considering the univariate dependency of prediction accuracy on each of these three quantities: number of users, number of items, and density level. We then conclude with an investigation of the multivariate dependency between the prediction accuracy and these three variables.

Figure 4.2 graphs the dependency of mean absolute error (MAE) on the number of users (top row), the number of items (middle row), and density (bottom row), with each of the three panels focusing on CF methods in a specific category (memory-based and baselines, model-based, and other). When one variable is varied, the others were fixed to 5,000 users, 2,000 items, and 3% of density. The RMSE evaluation measure

User Count \ Item Count	500	1000	1500	2000	2500	3000	3500	4000	4500	5000
1000	82.0%	74.3%	68.3%	62.8%	58.2%	54.0%	50.5%	47.2%	44.3%	41.8%
2000	80.0%	71.5%	64.7%	58.8%	53.8%	49.4%	45.7%	42.4%	39.5%	37.0%
3000	78.5%	69.4%	62.3%	56.1%	51.0%	46.6%	42.9%	39.6%	36.7%	34.2%
4000	77.3%	67.8%	60.4%	54.1%	48.9%	44.5%	40.8%	37.5%	34.7%	32.2%
5000	76.4%	66.5%	58.9%	52.6%	47.3%	42.8%	39.2%	35.9%	33.2%	30.8%
6000	75.6%	65.4%	57.7%	51.2%	46.0%	41.5%	37.8%	34.6%	31.9%	29.6%
7000	74.8%	64.4%	56.5%	50.1%	44.8%	40.3%	36.7%	33.5%	30.9%	28.6%
8000	74.0%	63.5%	55.5%	49.0%	43.7%	39.3%	35.7%	32.6%	29.9%	27.7%
9000	73.3%	62.6%	54.5%	48.0%	42.7%	38.4%	34.8%	31.7%	29.1%	26.9%
10000	72.7%	61.8%	53.7%	47.1%	41.9%	37.5%	34.0%	31.0%	28.4%	26.2%

User Count \ Item Count	500	1000	1500	2000	2500	3000	3500	4000	4500	5000
1000	82.0%	66.7%	56.1%	46.6%	39.8%	33.1%	29.3%	24.3%	21.2%	18.7%
2000	78.0%	59.2%	46.2%	35.6%	28.1%	21.7%	18.0%	13.8%	11.3%	9.4%
3000	75.5%	55.2%	41.6%	31.1%	23.8%	17.9%	14.6%	11.0%	8.9%	7.3%
4000	73.5%	52.1%	38.5%	28.1%	21.0%	15.5%	12.4%	9.2%	7.5%	6.1%
5000	72.9%	50.3%	36.3%	25.9%	19.1%	14.0%	11.1%	8.1%	6.3%	5.2%
6000	71.4%	48.4%	34.0%	24.0%	17.6%	12.8%	9.9%	7.2%	5.8%	4.7%
7000	70.3%	46.7%	32.5%	22.7%	16.2%	11.6%	9.0%	6.6%	5.2%	4.2%
8000	68.6%	44.8%	30.9%	21.2%	15.1%	10.9%	8.4%	6.1%	4.9%	3.9%
9000	67.8%	43.6%	29.6%	20.3%	14.4%	10.2%	7.8%	5.7%	4.5%	3.6%
10000	66.8%	42.2%	28.3%	19.2%	13.6%	9.8%	7.4%	5.3%	4.2%	3.3%

Figure 4.1: Rating density (cumulative in the top panel and non-cumulative in the bottom panel) in Netflix rating matrix, sorted by descending density of rows and columns. See text for more details.

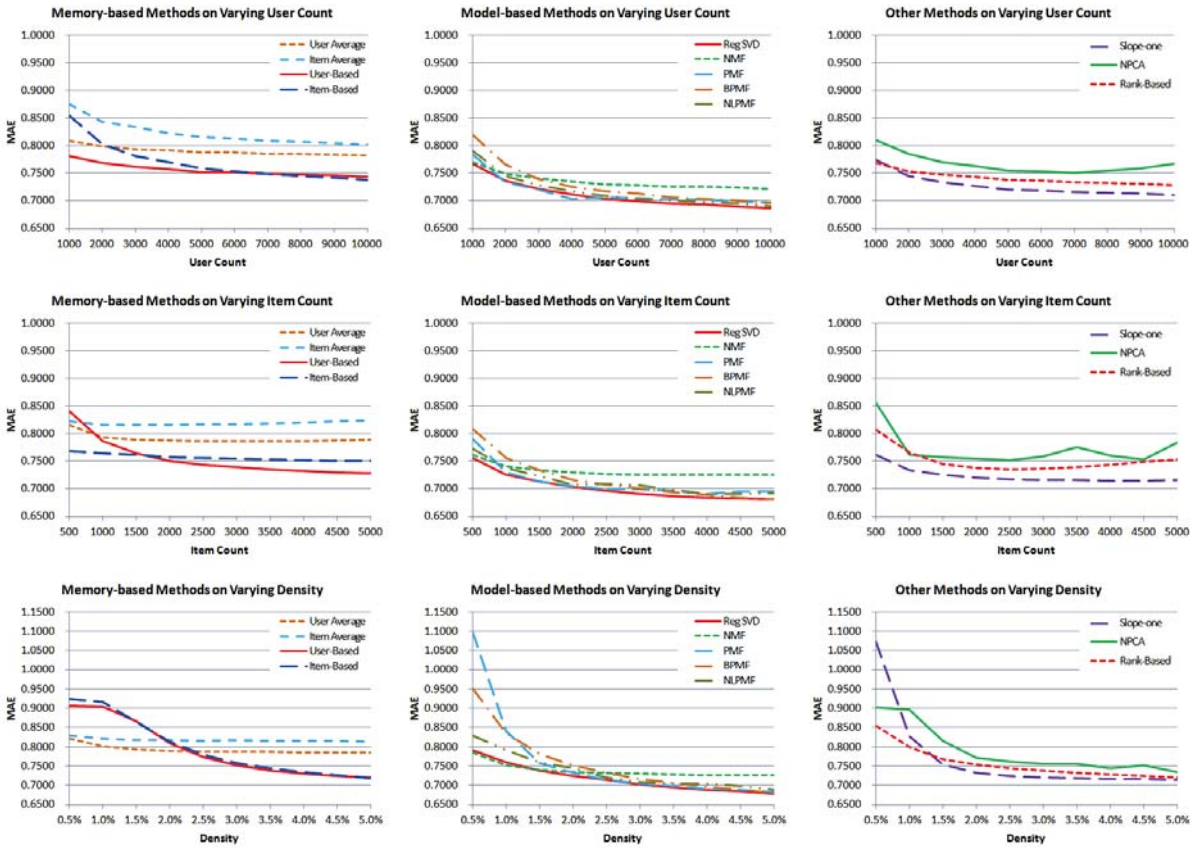

Figure 4.2: Prediction loss as a function of user count (top), item count (middle), and on density (bottom).

shows very similar trend. Also, the default voting variants of the user-based and item-based CF methods did not produce notable changes. Here is a list of our observations:

- In general, the best performing algorithm is regularized SVD.
- Matrix factorization methods in general show better performance when the dataset size is sufficiently large. ($> 3,000$ users and $> 1,000$ items).
- When the dataset size is sufficiently small, there is very little difference between matrix factorization and the simpler neighborhood-based methods. Also, the simple baselines (user average and item average) work remarkably well for low density.
- Regularized SVD, PMF, BPMF, and NLPMF tend to be the most sensitive to variation in dataset size. Interestingly, item average and item-based are sensitive only to variation in user count, while user-based is only to variation in item count.
- Slope-one, NMF, NPCA, and rank-based are relatively insensitive to the dataset size.
- User-based, item-based, slope-one, PMF, and BPMF are largely dependent on density, while the three baselines and NMF are relatively independent from density.
- User-based CF performs well when user count is small and item count is large. Item-based CF, on the other hand, works well when item count is small and user count is large. They show similar dependency pattern on density level. This is in stark contrast to their different dependencies on the user and item count.
- As the density level increases the differences in prediction accuracy of the different algorithm shrink.
- The performance of slope-one and PMF degrade significantly at low densities, performing worse than the weakest baselines. Nevertheless both algorithms feature outstanding performance at high densities.

The univariate dependencies examined previously show important trends but are limited since they examine variability of one quantity while fixing the other quantities

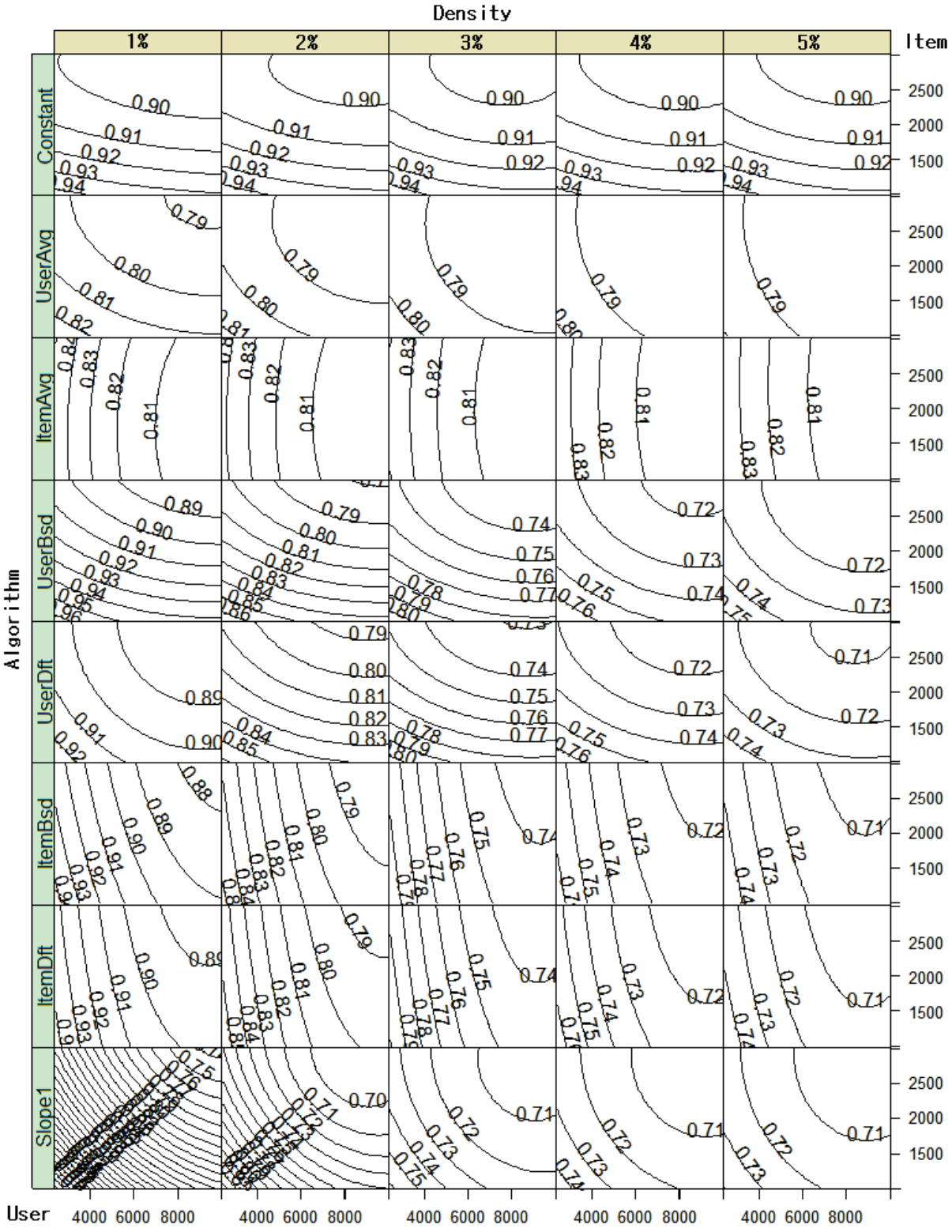

Figure 4.3: MAE Contours for simple method (Lower values mean better performance.)

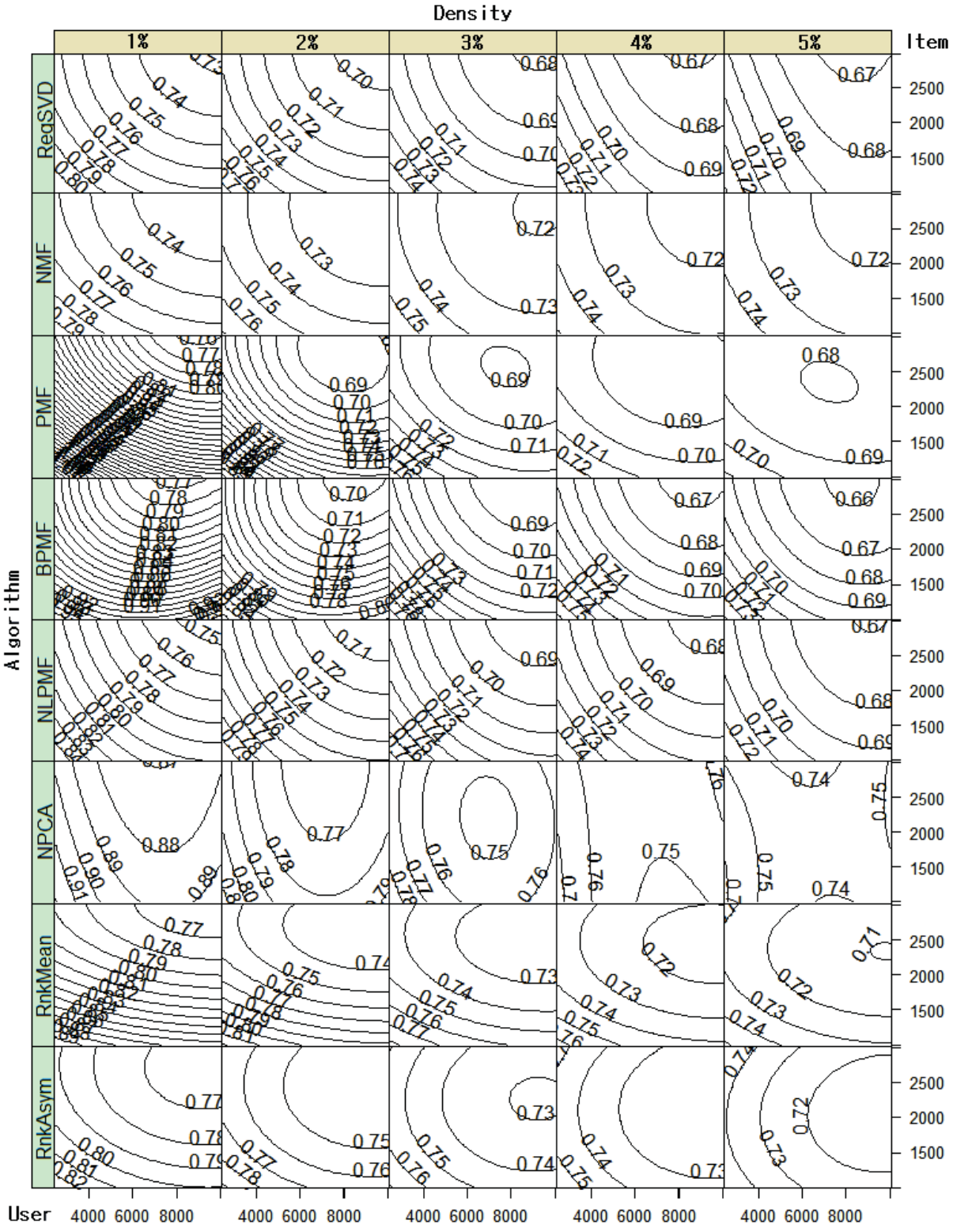

Figure 4.4: MAE Contours for advanced method (Lower values mean better performance.)

to arbitrary values. We now turn to examine the dependency between prediction loss and the three variables (user count, item count, and density) altogether. We do so by graphing the MAE as a function of user count, item count, and density.

Figure 4.3–4.4 shows the equal height contours of the MAE as a function of user count (x axis) and item count (y axis) for multiple density levels (horizontal panels) and for multiple CF algorithms (vertical panels). Note that all contour plots share the same x and y axes scales and so are directly comparable. Intervals between different contour lines represent a difference of 0.01 in MAE and so more contour lines represent higher dependency on the x and y axis. Analogous RMSE graphs show similar trend to these MAE graphs. Below is a summary of our observations:

- The univariate relationships discovered earlier for fixed values of the remaining two variables do not necessarily hold in general. For example, the conclusion that PMF is relatively sensitive to user count and item count at 1% sparsity level and 3% sparsity level is not longer valid for 5% density levels. It is thus important to conduct a multivariate, rather than univariate, analysis of the dependency of the prediction loss on the problem parameters.
- The shape of the MAE contour curves vary from algorithm to algorithm. For example, the contour curves of constant, user average, user-based, and user-based with default values are horizontal, implying that these algorithms depend largely on the number of items, regardless of user count. On the other hand, item average, item-based, and item-based with default values show vertical contour lines, showing a high dependency on the user count.
- Higher dependency on user count and item count is correlated with high dependency on density.
- Generally speaking, baselines are relatively insensitive, memory-based methods are dependent on one variable (opposite to their names), and matrix factorization methods are highly dependent on both dataset size and density.

4.2.2 Accuracy Comparison

Figure 4.5 shows the best performing algorithm (in terms of MAE) as a function of user count, item count, and density. This was drawn by superimposing contour lines in Figure 4.3 and 4.4 and by selecting the best-performing one in each area. Our observations are as follows:

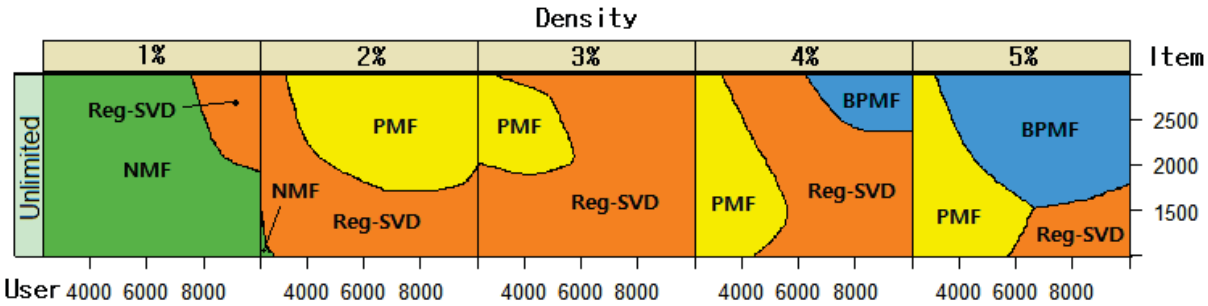

Figure 4.5: Best-performing algorithms in MAE for given user count, item count, and density.

- The identity of the best performing algorithm varies non-linearly, dependent on user count, item count, and density.
- NMF is dominant low density cases while BPMF works well for high density cases (especially for high user and item count).
- Regularized SVD and PMF perform well for density levels 2%-4%.

We can conclude that matrix factorization methods are generally better in most regions. BPMF performs best with sufficient information, while NMF works best when the data is relatively lacking. Analogous RMSE graphs show similar trends with regularized SVD outperforming other algorithms in most regions.

4.2.3 Computational Considerations

As Figure 4.6 shows, the computation time varies significantly between different algorithms. It is therefore important to consider computational issues when deciding on the appropriate CF algorithm. We consider three distinct scenarios, listed below.

- **Unlimited Time Resources:** We assume in this case that we can afford arbitrarily long computation time. This scenario is realistic in some cases involving static training set, making offline computation feasible.
- **Constrained Time Resource:** We assume in this cases some mild constraints on the computation time. This scenario is realistic in cases where the training set is periodically updated, necessitating periodic re-training with updated data. We assume here that the training phase should tale within an hour or so. Since practical datasets like Netflix full set are much bigger than the subsampled one in our experiments, we use much shorter time limit: 5 minutes and 1 minute.

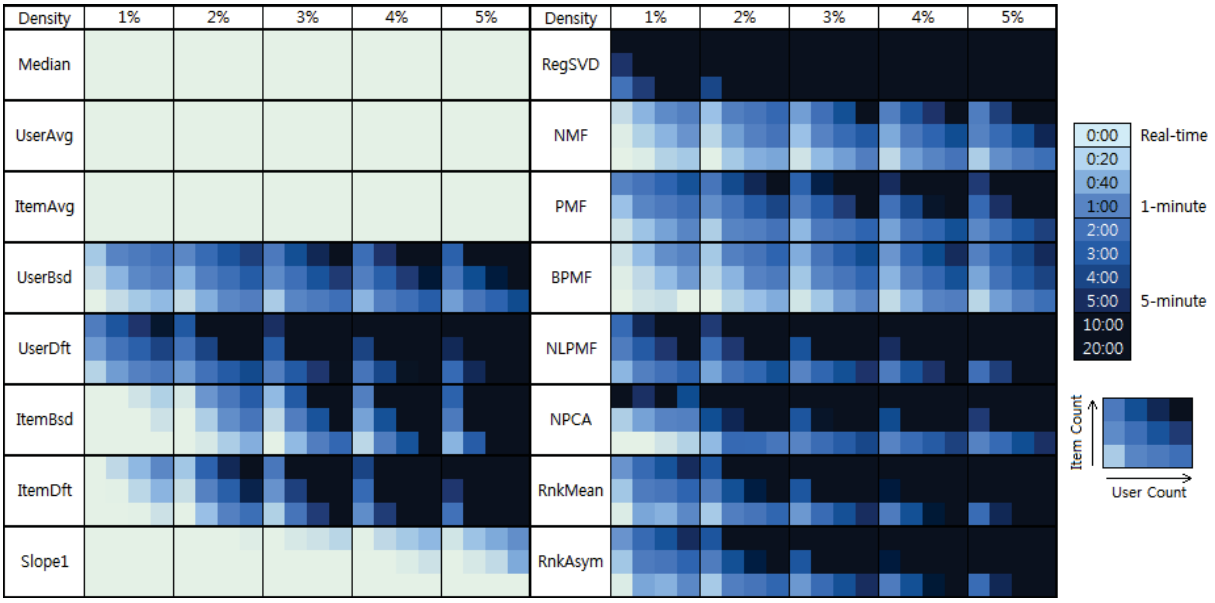

Figure 4.6: Computation time (Train time + Test time) for each algorithm. Legend on the right indicates relation between color scheme and computation time. Time constraints (5 minutes and 1 minutes) used in this article are marked as well. User count increases from left to right, and item count increases from bottom to top in each cell. (Same way to Figure 4.3–4.4.)

- **Real-time Applications:** In this case, we assume severe constraints on the computation time. This scenario is realistic in cases where the training set changes frequently and radically. We require here that the training phase should not exceed several seconds.

Figure 4.5 and 4.6 show the best performing CF algorithm (in terms of MAE) in several different time constraint cases, as a function of the user count, item count and density. We make the following observations.

- When there are no computation constraints, the conclusions from the previous sections apply. Specifically, NMF performs best for sparse dataset, BPMF performs best for dense dataset, and regularized SVD and PMF perform the best otherwise (PMF works well with smaller user count while Regularized SVD works well smaller item counts).

- When the time constraint is 5 minutes, Regularized SVD, NLPMF, NPCA, and Rank-based CF (the ones colored darkly in Figure 4.6) are not considered. In this setting, NMF works best for sparse data, BPMF works best for dense and large data and PMF works best otherwise.

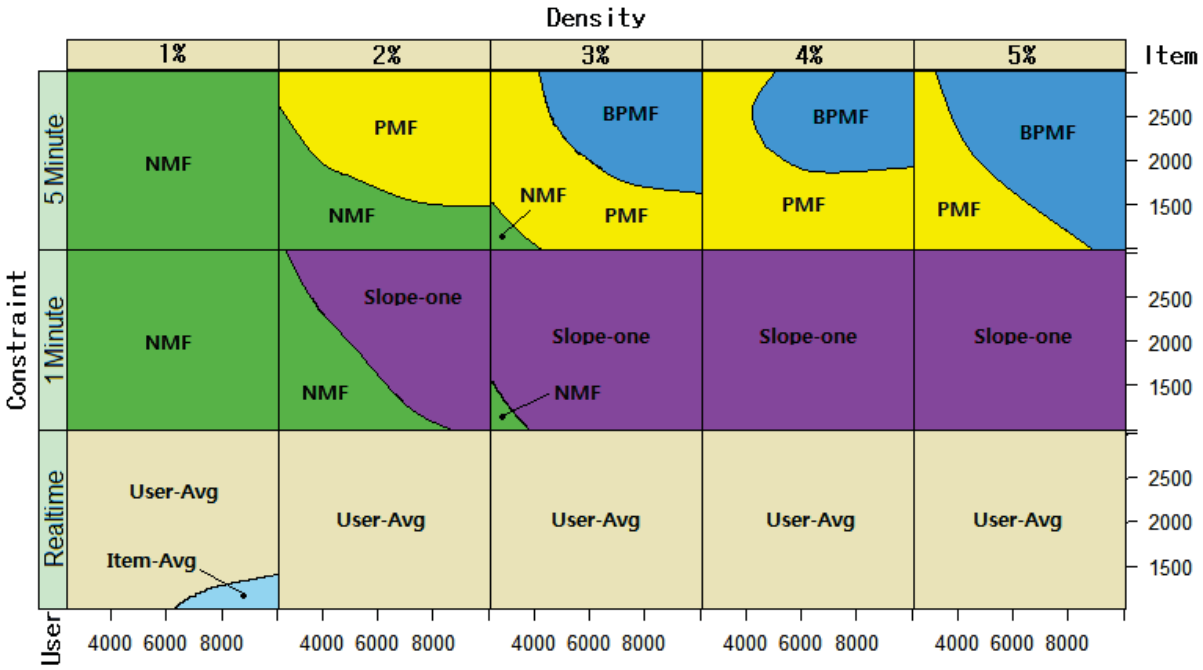

Figure 4.7: Best-performing algorithms with varied constraints.

- When the time constraint is 1 minutes, PMF and BPMF are additionally excluded from consideration. Slope-one works best in most cases, except for the sparsest data where NMF works best.
- In cases requiring real-time computation, the user average is the best algorithm, except for a small region where item average is preferred.

4.3 Conclusion

In addition to the conclusions stated in Section 4.2, we have identified seven groups of CF methods, where CF methods in the same group share certain experimental properties:

- Baselines: Constant, User Average, Item Average
- Memory-based methods: User-based, Item-based (with and without default values)
- Matrix-Factorization I: Regularized SVD, PMF, BPMF, NLPMF
- Matrix-Factorization II: NMF
- Others (Individually): Slope-one, NPCA, Rank-based CF.

Table 4.2 displays for each of these groups the dependency, accuracy, computational cost, and pros and cons.

To sum up, we conclude with this experimental study that no CF algorithm performs best in every situation. Depending on variables such as dataset size or density, each algorithm may perform well or not. In other words, some simple method may perform quite well for some special cases, while state-of-the-art methods may fail to beat such simple methods in some cases. We complete this section with summary of our three major conclusions.

- **Accuracy**: Matrix-Factorization-based methods generally have the highest accuracy. Specifically, regularized SVD, PMF and its variations perform best as far as MAE and RMSE, except in very sparse situations, where NMF performs the best. Slope-one achieves computational efficiency and not so bad accuracy at the same time. Memory-based methods, however, do not have special merit other than simplicity.

- **Dependency**: All algorithms vary in their accuracy, based on the user count, item count, and density. The strength and nature of the dependency, however, varies from algorithm to algorithm and bivariate relationships change when different values are assigned to the third variable. In general cases, high dependence on the user count and item count is correlated with high dependency on density, which appeared to be the more influential factor.

- **General Trade-off**: There is trade-off between better accuracy and other factors such as low variance in accuracy, computational efficiency, memory consumption, and a smaller number of adjustable parameters. That is, the more accurate algorithms tend to depend highly on dataset size and density, to have higher variance in accuracy, to be less computationally efficient, and to have more adjustable parameters. A careful examination of the experimental results can help resolve this trade-off in a manner that is specific to the situation at hand. For example, when computational efficiency is less important, Matrix-Factorization methods are the most appropriate, and when computational efficiency is important, slope-one could be a better choice.

Table 4.2: Summary of Pros and Cons for Recommendation Algorithms

Category		Baselines	Memory-based	Matrix-factorization		Others		
Algorithms		Constant, User Average, Item Average	User-based, Item-based, w/default	Reg-SVD, PMF, BPMF, NLPMF	NMF	Slope-one	NPCA	Rank-based
Dependency	Size	**Very low**	Low	High	Low	High	**Very low**	Fair
	Density	**Very low**	High	Very high	Low	Very high	High	Fair
Accuracy	Dense	Very poor	Good	**Very good**	Good	Good	Poor	Fair
	Sparse	Poor	Very poor	Fair	**Very good**	Poor	Poor	Fair
Asymmetric accuracy		Poor	Fair	**Very good**	Good	Good	**Very good**	**Very good**
HLU/NDCG		Very poor	Fair	**Very good**	Fair	Good	Fair	Fair
Kendall's Tau/Spearman		Very poor	Fair	Good	Fair	Fair	Fair	Fair
Computation	Train	**No**	**No**	Slow	Fast	**Very fast**	Slow	No
	Test	**Very fast**	Very slow	Fair	Fair	Fast	Slow	Very slow
Memory consumption		**Low**	High	High	High	**Low**	Very high	High
Adjustable parameters		**No**	Few	Many	Many	**No**	Few	Few
Overall Merits		Computation takes little time.	Do not need to train.	Perform best with high-density data.	Perform best with sparse data. Train is fast.	Perform well in spite of short time.	Perform well when using asymmetric measures.	Perform well when using asymmetric measures.
Overall Demerits		Accuracy is very poor.	Testing takes very long time. Uses lots of memory.	Many parameters should be adjusted. Computation takes long.	Many parameters should be adjusted.	Perform poorly without large/dense dataset.	Uses extremely large memory during training.	Computation takes too long time.

Chapter V

ENSEMBLES OF COLLABORATIVE FILTERING METHODS

We observed in the previous chapter that CF algorithms perform better or worse depending on data size or density. In addition to this, we found out that they predict ratings for some users or some items more accurately than others. For example, a method predicts famous items especially better than others. Departing from this observation, we propose an ensemble method with dynamic weights, which are learned automatically from data. We explain dynamic weights and stagewise feature induction method in turn, then conclude with experimental result.

5.1 Ensemble Methods

Ensemble learning in general refers to a process of combining multiple models such as classifiers or regressors in a strategic manner to solve a particular computational intelligence problem. Ensemble learning is primarily used to improve the performance of a model, or reduce the likelihood of an unfortunate selection of a poor one. [105]

To see the power of ensemble methods, let's think of the simplest ensemble model, an average of q models:

$$F(x) = \frac{1}{q}\sum_{k=1}^{q} f_k(x). \tag{5.1}$$

Suppose that $f_k(x) = h(x) + \epsilon_k(x)$, where $h(x)$ is the truth and $\epsilon_k(x)$ is the error of kth model. Then, the expected sum-of-squared error of kth model is given by

$$\mathbb{E}_x[\epsilon_k(x)^2] = \mathbb{E}_x[(f_k(x) - h(x))^2]. \tag{5.2}$$

Thus, the average error E_{ind} of individual models is given by

$$E_{\text{ind}} = \frac{1}{q}\sum_{k=1}^{q} \mathbb{E}_x[\epsilon_k(x)^2]. \tag{5.3}$$

In the same manner, the expected error E_{ens} of the ensemble model is

$$\begin{aligned} E_{\text{ens}} &= \mathbb{E}_x[(F(x) - h(x))^2] \\ &= \mathbb{E}_x\left[\left(\frac{1}{q}\sum_{k=1}^{q} f_k(x) - h(x)\right)^2\right] = \mathbb{E}_x\left[\left(\frac{1}{q}\sum_{k=1}^{q} \epsilon_k(x)\right)^2\right] \end{aligned} \tag{5.4}$$

If we assume that the error has zero mean ($\mathbb{E}_x[\epsilon_k(x)] = 0$) and uncorrelated ($\mathbb{E}_x[\epsilon_k(x)\epsilon_l(x)] = 0$ when $k \neq l$), we have $E_{\text{ens}} \geq \frac{1}{q}E_{\text{ind}}$. On the other hand, it can be proved that $E_{\text{ens}} \leq E_{\text{ind}}$ where the equality is held when errors are completely correlated. Putting them together, we can conclude

$$\frac{1}{q}E_{\text{ind}} \leq E_{\text{ens}} \leq E_{\text{ind}}, \tag{5.5}$$

meaning that ensemble models perform usually better than individual models, as the error of each individual model is neither completely correlated nor independent.

Although ensemble models have enjoyed success in many machine learning problems including classification, regression, or clustering, it is somewhat surprising that they have had relatively little success in the collaborative filtering literature. Generally speaking, ensemble or combination methods have shown only a minor improvement over the top-performing CF methods. The cases where ensemble methods did show an improvement (for example, the Netflix prize winner [72] and runner up), relied heavily on manual feature engineering, manual parameter setting, and other tinkering. In this chapter, we suggest a successful application of ensemble methods on collaborative filtering domain.

5.2 Dynamic Weights

For convenience, we denote a CF method as a function $f(u, i)$, returning a preference score for an item i for a user u, and the family of potential CF methods as $\mathcal{F}$. Ensemble methods combining multiple CF models from $\mathcal{F}$ are denoted with upper-cases.

As discussed in the previous chapter, recommendation systems perform better than others depending on situations. In addition to this, we also found that different recommendation systems perform better than others for some users and items but not for others. In other words, the relative strengths of two distinct CF models $f_1(u, i), f_2(u, i) \in \mathcal{F}$ depend on the user u and the item i whose rating is being predicted. One example of two such systems appears in Figure 5.1 that graphs the test-set loss of two recommendation rules (user average and item average) as a function of the

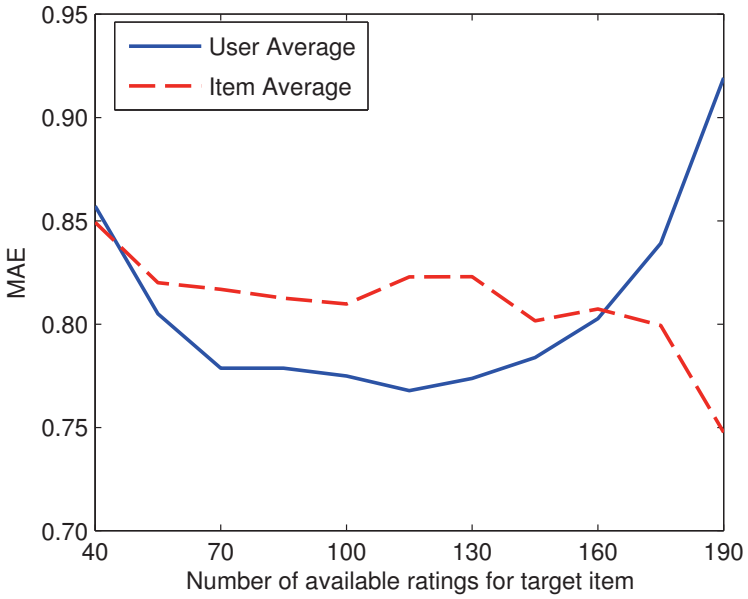

Figure 5.1: An example of two simple algorithms that perform differently depending on item popularity.

number of available ratings for the recommended item i. The two recommendation rules outperform each other, depending on whether the item in question has few or many ratings in the training data. We conclude from this graph and experimental result in Chapter 4 that algorithms that are inferior in some circumstances may be superior in other circumstances.

An inescapable conclusion is that the weights α_k in the combination should be functions of user u and item i rather than constants. That is, a linear combination of q recommender systems $f_k(u,i)$ is written by

$$F^{(q)}(u,i) = \sum_{k=1}^{q} \alpha_k(u,i) f_k(u,i) \tag{5.6}$$

where $\alpha_k(u,i) \in \mathbb{R}$. We define non-constant combination weights $\alpha_k(u,i)$ that are functions of the user and item that are being predicted. We propose the following algebraic form

$$\alpha_k(u,i) = \beta_k \, h_k(u,i), \quad \beta_k \in \mathbb{R}, \quad h_k \in \mathcal{H} \tag{5.7}$$

where β_k is a parameter and h_k is a function selected from a family $\mathcal{H}$ of candidate feature functions.

The combination (5.6) with non-constant weights (5.7) enables some CF methods f_k to be emphasized for some user-item combinations through an appropriate selection

of the β_k coefficients. We assume that $\mathcal{H}$ contains the constant function, capturing the constant-weight combination within our model.

Substituting (5.7) into (5.6) we get

$$F^{(q)}(u,i) = \sum_{k=1}^{q} \beta_k \, h_k(u,i) \, f_k(u,i), \quad \beta_k \in \mathbb{R}, \quad h_k \in \mathcal{H}, \quad f_k \in \mathcal{F}. \tag{5.8}$$

Note that since h_k and f_k are selected from the sets of CF methods and feature functions respectively, we may have $f_j = f_l$ or $h_j = h_l$ for $j \neq l$. This is similar to boosting and other stagewise algorithms where one feature or base learner may be chosen multiple times, effectively updating its associate feature functions and parameters. The total weight function associated with a particular $f \in \mathcal{F}$ is $\sum_{k:f_k=f} \beta_k h_k(u,i)$.

A simple way to fit $\beta = (\beta_1, \ldots, \beta_q)$ is least squares

$$\beta^* = \arg\min_{\beta \in C} \sum_{u,i \in \Omega} \left(F^{(q)}(u,i) - M_{u,i} \right)^2, \tag{5.9}$$

where $M_{u,i}$ denotes the rating of user u on item i in the training data and the summation ranges over all ratings in the training set. A variation of (5.9), where β is constrained such that $\alpha_k(u,i) \geq 0$, and $\sum_{k=1}^{q} \alpha_k(u,i) = 1$ endows F with the following probabilistic interpretation

$$F(u,i) = \mathsf{E}_p \left\{ f \mid u, i \right\}, \tag{5.10}$$

where f represents a random draw from $\mathcal{F}$, with probabilities $p(f|u,i)$ proportional to $\sum_{k:f_k=f} \beta_k h_k(u,i)$. In contrast to standard combination models with fixed weights, (5.10) forms a conditional expectation, rather than an expectation.

Feature-weighted linear stacking (FWLS) [123] also has suggested ensembles with non-constant weights. FWLS selects 25 useful features for weight functions by close human inspection on data. Although it has achieved comparable performance to the winner in Netflix competition, this manual selection of features is too labor intensive. Also, a feature may or may not be useful depending on dataset. To overcome these drawbacks, we present an approach automatically inducing local features from data in the next subsection.

5.3 Stagewise Feature Induction

In contrast to [123] that manually defined features, we induce the features h_k from data. The features $h_k(u,i)$ should emphasize users u and items i that are likely to

lead to variations in the relative strength of the $f_1, \ldots, f_q$. In this section, we consider two issues: 1) defining the set $\mathcal{H}$ of candidate features, and 2) a strategy for selecting features from $\mathcal{H}$ to add to the combination F.

5.3.1 Candidate Feature Families $\mathcal{H}$

We denote the sets of users and items by $\mathcal{U}$ and $\mathcal{I}$ respectively, and the domain of $f \in \mathcal{F}$ and $h \in \mathcal{H}$ as $\mathcal{U} \times \mathcal{I}$. The set $\Omega \subset \mathcal{U} \times \mathcal{I}$ is the set of user-item pairs present in the training set, and the set of user-item pairs that are being predicted is $\omega \in \Omega^c$.

We consider the following three unimodal functions on $\mathcal{U} \times \mathcal{I}$, parameterized by a location parameter or mode $\omega^* = (u^*, i^*) \in \mathcal{U} \times \mathcal{I}$ and a bandwidth $h > 0$

$$
\begin{aligned}
K^{(1)}_{h,(u^*,i^*)}(u,i) &\propto \left(1 - \frac{d(u^*,u)}{h}\right) I\left(d(u^*,u) \leq h\right), \\
K^{(2)}_{h,(u^*,i^*)}(u,i) &\propto \left(1 - \frac{d(i^*,i)}{h}\right) I\left(d(i^*,i) \leq h\right), \\
K^{(3)}_{h,(u^*,i^*)}(u,i) &\propto \left(1 - \frac{d(u^*,u)}{h}\right) I\left(d(u^*,u) \leq h\right) \cdot \left(1 - \frac{d(i^*,i)}{h}\right) I\left(d(i^*,i) \leq h\right),
\end{aligned}
\tag{5.11}
$$

where $I(A) = 1$ if A holds, and 0 otherwise. The first function is unimodal in u, centered around u^*, and constant in i. The second function is unimodal in i, centered around i^*, and constant in u. The third is unimodal in u, i and centered around (u^*, i^*).

There are several possible choices for the distance functions in (5.11) between users and between items. For simplicity, we use in our experiments the angular distance

$$
d(x,y) = \arccos\left(\frac{\langle x, y\rangle}{\|x\| \cdot \|y\|}\right) \tag{5.12}
$$

where the inner products above are computed based on the user-item rating matrix expressing the training set (ignoring entries not present in both arguments).

The functions (5.11) are the discrete analogs of the triangular kernel $K_h(x) = h^{-1}(1 - |x - x^*|/h)I(|x - x^*| \leq h)$ used in non-parametric kernel smoothing [137]. Their values decay linearly with the distance from their mode (truncated at zero), and feature a bandwidth parameter h, controlling the rate of decay. As h increases the support size $|\{\omega \in \Omega : K(\omega) > 0\}|$ increases and $\max_{\omega \in \Omega} K(\omega)$ decreases.

The unimodal feature functions (5.11) capture locality in the $\mathcal{U} \times \mathcal{I}$ space by measuring proximity to a mode, representing a user u^*, an item i^*, or a user-item pair. We define

the family of candidate features $\mathcal{H}$ as all possible additive mixtures or max-mixtures of the functions (5.11), parameterized by a set of multiple modes $\boldsymbol{\omega}^* = \{\omega_1^*, \ldots, \omega_r^*\}$

$$K_{\boldsymbol{\omega}^*}(u,i) \propto \sum_{j=1}^{r} K_{\omega_j^*}(u,i) \tag{5.13}$$

$$K_{\boldsymbol{\omega}^*}(u,i) \propto \max_{j=1,\ldots,r} K_{\omega_j^*}(u,i). \tag{5.14}$$

Using this definition, features functions $h_k(u,i) \in \mathcal{H}$ are able to express a wide variety of locality information involving multiple potential modes.

5.3.2 Feature Induction Strategy

We discuss in this subsection the strategy for identifying useful features from $\mathcal{H}$ and adding them to the model F in a stagewise manner.

Adapting the stagewise learning approach to the model (5.8) we have

$$F^{(q)}(u,i) = \sum_{k=1}^{q} \beta_k \, h_k(u,i) \, f_k(u,i), \tag{5.15}$$

$$(\beta_k, h_k, f_k) = \underset{\beta_k \in \mathbb{R}, h_k \in \mathcal{H}, f_k \in \mathcal{F}}{\arg\min} \sum_{(u,i)\in\Omega} \left(F^{(k-1)}(u,i) + \beta_k h_k(u,i) f_k(u,i) - M_{u,i} \right)^2 .$$

It is a well-known fact that stagewise algorithms sometimes outperform non-greedy algorithms due to resistance to overfitting (see [110], for example). This explains the good generalization ability of boosting and stage-wise linear regression.

From a computational standpoint, (5.15) scales nicely with q and with the training set size. The one-dimensional quadratic optimization with respect to β is solved via a closed form, but the optimization over $\mathcal{F}$ and $\mathcal{H}$ has to be done by brute force or by some approximate method such as sampling. The computational complexity of each iteration is thus $O(|\mathcal{H}| \cdot |\mathcal{F}| \cdot |R|)$, assuming no approximation are performed.

Since we consider relatively small families $\mathcal{F}$ of CF methods, the optimization over $\mathcal{F}$ does not pose a substantial problem. The optimization over $\mathcal{H}$ is more problematic since $\mathcal{H}$ is very large, and potentially infinite. We address this difficulty by restricting $\mathcal{H}$ to a finite collection of additive or max-mixtures kernels with r modes, randomly sampled from the users or items present in the training data. Our experiments conclude that it is possible to find useful features from a surprisingly small number of randomly-chosen samples.

Algorithm 5.1 details the stagewise algorithm. It starts with the best-performing candidate as $f_1(u, i)$ with a constant feature. Then, in each iteration, it seeks the best candidate algorithm $f_k(u, i)$ and corresponding feature function $h_k(u, i)$ which produces the lowest validation error when combined with the current combination $F^{(k-1)}$. Then, we optimize over β by least squares. Note that $\mathcal{H}'$ in Line 7 denotes the set of samples of $h \in \mathcal{H}$. This is because we cannot search over all $h \in \mathcal{H}$ due to the computational infeasibility.

Algorithm 5.1 Learning β for combined model

1: $f_1 \leftarrow \arg\min_{f \in \mathcal{F}} RMSE(f)$
2: $h_1 \leftarrow 1$
3: $\beta_1 \leftarrow 1$
4: **for** $k = 2 \rightarrow q$ **do**
5: $\quad \mathcal{H}' \leftarrow$ random samples from $\mathcal{H}$
6: $\quad$ **for** $f \in \mathcal{F}$ **do**
7: $\quad\quad$ **for** $h \in \mathcal{H}'$ **do**
8: $\quad\quad\quad$ // Preparing data matrices:
9: $\quad\quad\quad$ **for** $n = 1 \rightarrow m$ **do**
10: $\quad\quad\quad\quad d_n \leftarrow f(u_n, i_n) - F^{(k-1)}(u_n, i_n)$
11: $\quad\quad\quad$ **end for**
12: $\quad\quad\quad Z \leftarrow \begin{pmatrix} d_1 & d_1 h(u_1, i_1) \\ d_2 & d_2 h(u_2, i_2) \\ \vdots & \vdots \\ d_m & d_m h(u_m, i_m) \end{pmatrix}$
13: $\quad\quad\quad y \leftarrow \begin{pmatrix} M_{u_1,i_1} - F^{(k-1)}(u_1, i_1) \\ M_{u_2,i_2} - F^{(k-1)}(u_2, i_2) \\ \vdots \\ M_{u_m,i_m} - F^{(k-1)}(u_m, i_m) \end{pmatrix}$
14: $\quad\quad\quad$ // Solving quadratic programming:
15: $\quad\quad\quad \beta_{f,h} \leftarrow \arg\min_{\beta} (Z\beta - y)^T (Z\beta - y)$
16: $\quad\quad\quad RMSE_{f,h} \leftarrow \sqrt{\frac{1}{m}(Z\beta_{f,h} - y)^T (Z\beta_{f,h} - y)}$
17: $\quad\quad$ **end for**
18: $\quad$ **end for**
19: $\quad f_k, h_k \leftarrow \arg\min_{f \in \mathcal{F}, h \in \mathcal{H}} RMSE_{f,h}$
20: $\quad \beta_k \leftarrow \beta_{f_k,h_k}$
21: **end for**
22: **return** $f_1, ..., f_q, h_1, ..., h_q, \beta_1, ..., \beta_q$

5.4 Experiment

5.4.1 Experimental Design

We used the recommendation algorithm toolkit PREA (Chapter 3) for candidate algorithms, including three simple baselines (Constant model, User Average, and Item Average) and five matrix-factorization methods (Regularized SVD, NMF [79], PMF [115], Bayesian PMF [114], and Non-Linear PMF [75]), and Slope-one [82]. We evaluate the performance using the Root Mean Squared Error (RMSE), measured on the test set.

Table 5.1 lists 5 experimental settings. SINGLE runs each CF algorithm individually, and chooses the one with the best average performance. CONST combines all candidate algorithms with constant weights α_k,

$$F^{(q)}(u,i) = \sum_{k=1}^{q} \alpha_k f_k(u,i). \tag{5.16}$$

FWLS [123] combines all candidate algorithms with non-constant weights as in (5.6). For CONST and FWLS, the weights are estimated from data by solving a least-square problem. STAGE combines CF algorithms in stage-wise manner. FEAT applies the feature induction techniques discussed in Section 5.3.2.

To evaluate whether the automatic feature induction in FEAT works better or worse than manually constructed features, we used in FWLS and STAGE manual features similar to the ones in [123] (excluding features requiring temporal data). Examples include number of movies rated per user, number of users rating each movie, standard deviation of the users' ratings, and standard deviation of the item's ratings.

The feature induction in FEAT used a feature space $\mathcal{H}$ with additive multi-mode smoothing kernels as described in Section 5.3.1 (for simplicity we avoided kernels unimodal in both u and i). The family $\mathcal{H}$ included 200 randomly sampled features

Table 5.1: Experimental setting. (C: Combination of multiple algorithms, W: Weights varying with features, S: Stage-wise algorithm, I: Induced features)

Method	C	W	S	I	Explanation
SINGLE					Best-performed single CF algorithm
CONST	O				Mixture of CF without features
FWLS	O	O			Mixture of CF with manually-designed features
STAGE	O	O	O		Stagewise mixture with manual features
FEAT	O	O	O	O	Stagewise mixture with induced features

(a new sample was taken for each of the iterations in the stagewise algorithms). The r in (5.14) was set to 5% of user or item count, and bandwidth h values of 0.05 (an extreme case where most features have value either 0 or 1) and 0.8 (each user or item has moderate similarity values). The stagewise algorithm continues until either five consecutive trials fail to improve the RMSE on validation set, or the iteration number reaches 100, which occur only in a few cases. We used similar L_2 regularization for all methods (both stagewise and non-stagewise), where the regularization parameter was selected among 5 different values based on a validation set.

5.4.2 Data and Candidate Algorithms

We experimented with the two standard MovieLens datasets: 100K and 1M, and with the Netflix dataset. In the Netflix dataset experiments, we sub-sampled the data since (a) running state-of-the-art candidate algorithms on the full Netflix data takes too long time – for example, Bayesian PMF was reported to take 188 hours [114], and (b) it enables us to run extensive experiments measuring the performance of the CF algorithms as a function of the number of users, number of items, voting sparsity, and facilitates cross-validation and statistical tests. More specifically, we sub-sampled from the most active k users and the most often rated ℓ items to obtain pre-specified data density levels. As shown in Table 5.2, we varied either the user or item count in the set $\{1000, 1500, 2000, 2500, 3000\}$, holding the other variable fixed at 1000 and the density at 1%, which is comparable density of the original Netflix dataset. We also conducted an experiment where the data density varied in the set $\{1\%, 1.5\%, 2\%, 2.5\%\}$ with fixed user and item count of 1000 each.

We set aside a randomly chosen 20% for test set, and used the remaining 80% for both for training the individual recommenders and for learning the ensemble model. It is possible, and perhaps more motivated, to use two distinct train sets for the CF models and the ensemble. However, in our case, we got high performance even in the case of using the same training dataset in both stages.

For stagewise methods, the 80% train set was divided to 60% training set and 20% validation set, used to determine when to stop the stagewise addition process. The non-stagewise methods used the entire 80% for training. The 10% of training set is used to select regularization parameter for both stagewise and non-stagewise. The results were averaged over 10 random data samples.

Table 5.2: Test error in RMSE (lower values are better) for single CF algorithms used as candidates and combined models. Data where n_1 or n_2 is 1500 or 2500 are omitted due to the lack of space, as it is shown in Figure 5.2. The best-performing one in each group is indicated in *italic*. The last row indicates p-value for statistical test of hypothesis FEAT $\succ$ FWLS.

Dataset		Netflix								MovieLens	
User Count		1000	2000	3000	1000		1000			943	6039
Item Count		1000	1000		2000	3000	1000			1682	3883
Density		1.0%	1.0%		1.0%		1.5%	2.0%	2.5%	6.3%	4.3%
Single CF	Constant	1.2188	1.2013	1.2072	1.1964	1.1888	1.2188	1.2235	1.2113	1.2408	1.2590
	UserAvg	1.0566	1.0513	1.0375	1.0359	*1.0174*	1.0566	1.0318	1.0252	1.0408	1.0352
	ItemAvg	1.1260	1.0611	1.0445	1.1221	1.1444	1.1260	1.1029	1.0900	1.0183	0.9789
	Slope1	1.4490	1.4012	1.3321	1.4049	1.3196	1.4490	1.3505	1.0725	0.9371	0.9017
	RegSVD	1.0623	*1.0155*	1.0083	*1.0354*	1.0289	*1.0343*	1.0154	*1.0020*	*0.9098*	*0.8671*
	NMF	1.0784	1.0205	*1.0069*	1.0423	1.0298	1.0406	*1.0151*	1.0091	0.9601	0.9268
	PMF	1.6180	1.4824	1.4081	1.4953	1.4804	1.4903	1.3594	1.1818	0.9328	0.9623
	BPMF	1.3973	1.2951	1.2949	1.2566	1.2102	1.3160	1.2021	1.1514	0.9629	0.9000
	NLPMF	*1.0561*	1.0507	1.0382	1.0361	1.0471	1.0436	1.0382	1.0523	0.9560	0.9415
Combined	SINGLE	1.0561	1.0155	1.0069	1.0354	1.0174	1.0343	1.0151	1.0020	0.9098	0.8671
	CONST	1.0429	1.0072	0.9963	1.0198	1.0102	1.0255	0.9968	0.9824	0.9073	0.8660
	FWLS	1.0288	1.0050	0.9946	1.0089	1.0016	1.0179	0.9935	0.9802	0.9010	0.8649
	STAGE	1.0036	0.9784	0.9668	0.9967	0.9821	0.9935	0.9846	0.9769	0.8961	0.8623
	FEAT	*0.9862*	*0.9607*	*0.9607*	*0.9740*	*0.9717*	*0.9703*	*0.9589*	*0.9492*	*0.8949*	*0.8569*
p-Value		0.0028	0.0001	0.0003	0.0008	0.0014	0.0002	0.0019	0.0013	0.0014	0.0023

5.4.3 Result and Discussion

Table 5.2 displays the performance in RMSE of each combination method, as well as the individual algorithms. Examining it, we observe the following partial order with respect to prediction accuracy: FEAT $\succ$ STAGE $\succ$ FWLS $\succ$ CONST $\succ$ SINGLE.

- **FWLS $\succ$ CONST $\succ$ SINGLE**: Combining CF algorithms (even only with constant weights) produces better prediction than the best-single CF method. Also, using non-constant weights improves performance further. This result is consistent with what has been known in literature [56, 123].
- **STAGE $\succ$ FWLS**: Figure 5.2 indicates that stagewise combinations where features are chosen with replacement are more accurate. The selection with replacement allow certain features to be selected more than once, correcting a previous inaccurate parameter setting.
- **FEAT $\succ$ STAGE**: Making use of induced features improves prediction accuracy further from stagewise optimization with manually-designed features.

Overall, our experiments indicate that the combination with non-constant weights and feature induction (FEAT) outperforms three baselines (the best single method,

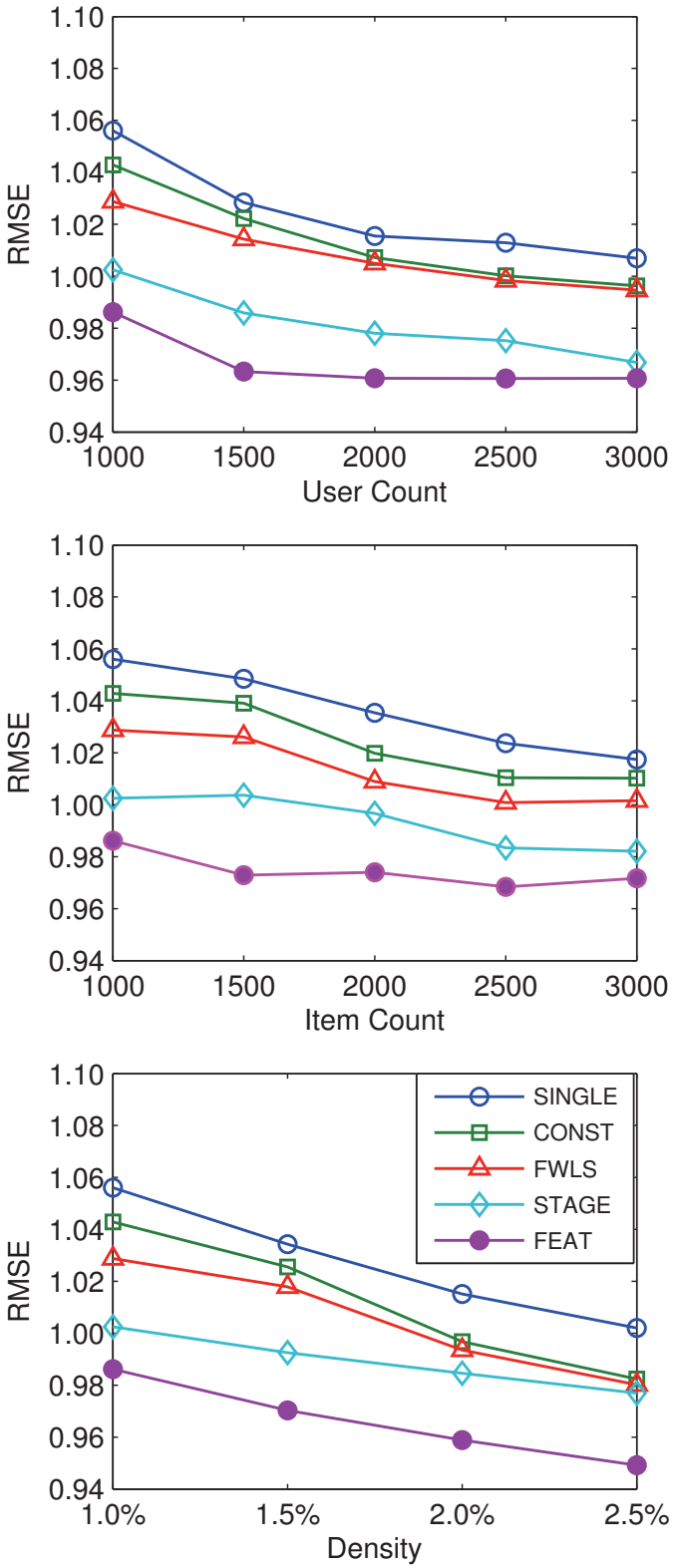

Figure 5.2: Performance trend with varied user count (top), item count (middle), and density (bottom) on Netflix dataset.

standard combinations with constant weight, and the FWLS method using manually constructed features [123]). We tested the hypothesis $RMSE_{FEAT} < RMSE_{FWLS}$ with paired t-test. Based on the p-values indicated in the last row of Table 5.2, we can reject the null hypothesis with significance of 99%. We conclude that our proposed combination outperforms state-of-the-art methods, and several previously proposed combination methods.

5.4.4 Scalability

We complete this section with a remark on scalability of the proposed algorithm. The proposed stagewise algorithm is very efficient, when compared to other feature selection algorithms such as step-wise or subset selection. Nevertheless, the large number of possible features may result in computational issues. In our experiments, we sampled from the space of candidate features a small subset of features that was considered for addition (the random subset is different in each iteration of the stagewise algorithm). In the limit $q \to \infty$, such a sampling scheme would recover the optimal ensemble as each feature will be selected for consideration infinitely often. Our experiments conclude that this scheme works well also in practice and results in significant improvement to the state-of-the-art even for a relatively small sample of feature candidates such as 200. Viewed from another perspective, this implies that randomly selecting such a small subset of features each iteration ensures the selection of useful features. In fact, the features induced in this manner were found to be more useful than the manually crafted features in the FWLS algorithm [123].

5.5 Summary

We started from an observation that the relative performance of different candidate recommendation systems $f(u, i)$ depends on u and i, for example on the activity level of user u and popularity of item i. This motivated the development of combination of recommendation systems with non-constant weights that emphasize different candidates based on their relative strengths in the feature space. In contrast to the FWLS method that focused on manual construction of features, we developed a feature induction algorithm that works in conjunction with stagewise least-squares. We formulate a family of feature functions, based on the discrete analog of triangular kernel smoothing. This family captures a wide variety of local information and is thus able to model the relative strengths of the different CF methods.

The combination with induced features outperformed any of the base candidates as well as other combination methods in literature. This includes the recently proposed FWLS method that uses manually constructed feature function. Our method scales up nicely to large data which is often present in real world recommendation systems. As our candidates included many of the recently proposed state-of-the-art recommendation systems our conclusions are significant for the engineering community as well as recommendation system scientists.

Chapter VI

LOCAL LOW-RANK MATRIX APPROXIMATION

Matrix factorization is the most common and successful approach for rating prediction in recommendation system, as we have seen in Chapter 4. The basic underlying assumption is the rating matrix is in low-rank, so we can express it as a product of two low-rank matrices. In this chapter, we propose a new matrix approximation model where we assume instead that the matrix is locally of low-rank, leading to a representation of the observed matrix as a weighted sum of low-rank matrices. As explained in Section 6.2.3, this can be seen as an extension of the ensemble method discussed in Chapter 5. In the followed sections, we analyze the accuracy of the proposed local low-rank modeling and show experimental evidence of its performance.

6.1 Low-Rank Matrix Approximation

Matrix approximation and completion are prevalent tasks in machine learning. Given few observed matrix entries $\{M_{a_1,b_1}, \ldots, M_{a_m,b_m}\}$, matrix completion constructs a matrix $\hat{M}$ that approximates M at its unobserved entries. Matrix approximation is used heavily in recommendation systems, text processing, computer vision, and bioinformatics. In recommendation systems, the matrix M corresponds to ratings of items (columns) by users (rows).

In general, the problem of completing a matrix M based on a few observed entries is ill-posed. There are uncountably infinite number of matrices that perfectly agree with the observed entries of M. Therefore, without additional assumptions, selecting or constructing the completion matrix $\hat{M}$ is under-specified and thus ill-defined. A popular assumption is that M is a low-rank matrix, which suggests that it is reasonable to assume that the completed matrix $\hat{M}$ has low-rank. More formally, we approximate a matrix $M \in \mathbb{R}^{n_1 \times n_2}$ by a rank r matrix $\hat{M} = UV^\top$, where $U \in \mathbb{R}^{n_1 \times r}$, $V \in \mathbb{R}^{n_2 \times r}$, and $r \ll \min(n_1, n_2)$. In many real datasets, the low-rank assumption is realistic. Further, low-rank approximations often yield matrices that generalize well to the unobserved entries.

We describe in this section two standard approaches for low-rank matrix approximation (LRMA). We start by establishing the notation used throughout the chapter. The original (partially observed) matrix is denoted by $M \in \mathbb{R}^{n_1 \times n_2}$. A low-rank approximation of M is denoted by $\hat{M} = UV^\top$, where $U \in \mathbb{R}^{n_1 \times r}$, $V \in \mathbb{R}^{n_2 \times r}$, and $r \ll \min(n_1, n_2)$. The set of integers $\{1, \ldots, n\}$ is abbreviated as $[n]$. The set of observed entries of M is denoted by $\Omega \stackrel{\text{def}}{=} \{(a_1, b_1), \ldots, (a_m, b_m)\} \subseteq [n_1] \times [n_2]$. The training set is therefore $\{M_{a,b} : (a, b) \in \Omega\}$. Mappings from matrix indices to a matrix space are denoted in calligraphic letters, e.g. $\mathcal{T}$, and are operators of the form $\mathcal{T} : [n_1] \times [n_2] \to \mathbb{R}^{n_1 \times n_2}$. We denote the entry (i, j) of the matrix $\mathcal{T}(a, b)$ as $\mathcal{T}_{i,j}(a, b)$. A projection $\mathcal{P}_\Omega$ with respect to a set of matrix indices Ω is the function $\mathcal{P}_\Omega : \mathbb{R}^{n_1 \times n_2} \to \mathbb{R}^{n_1 \times n_2}$ defined by

$$[\mathcal{P}_\Omega(M)]_{a,b} \stackrel{\text{def}}{=} \begin{cases} M_{a,b} & (a, b) \in \Omega \\ 0 & \text{otherwise.} \end{cases}$$

We denote by $\odot$ the entry-wise product (also known as the Hadamard or Schur products) of two matrices $[A \odot B]_{i,j} = A_{i,j} B_{i,j}$. We use three matrix norms:

Frobenius norm

$$\|X\|_F \stackrel{\text{def}}{=} \sqrt{\sum_i \sum_j X_{i,j}^2}, \tag{6.1}$$

Sup-norm

$$\|X\|_\infty \stackrel{\text{def}}{=} \sup_{i,j} |X_{i,j}|, \tag{6.2}$$

Nuclear (trace) norm

$$\|X\|_* \stackrel{\text{def}}{=} \sum_{i=1}^{r} \sigma_i(X), \tag{6.3}$$

For the nuclear norm $\sigma_i(X)$ is the i'th singular value of X where for symmetric matrices $\|X\|_* = \text{trace}(X)$.

Below are two popular approaches for constructing a low-rank approximation $\hat{M}$ of M. The first one, incomplete SVD, is based on minimizing the Frobenius norm of $\mathcal{P}_\Omega(M - \hat{M})$, while the second one is based on minimizing the nuclear norm of a matrix satisfying constraints constructed from the training set.

A1: Incomplete SVD. The incomplete SVD method constructs a low-rank approximation $\hat{M} = UV^\top$ by solving the problem

$$(U, V) = \arg\min_{U,V} \sum_{(a,b) \in \Omega} ([UV^\top]_{a,b} - M_{a,b})^2 \ , \tag{6.4}$$

or equivalently

$$\hat{M} = \arg\min_{X} \|\mathcal{P}_\Omega(X - M)\|_F \text{ s.t. } \text{rank}(X) = r. \tag{6.5}$$

A2: Nuclear norm minimization. An alternative to (6.5) that originated from the compressed sensing community [18] is to minimize the nuclear norm of a matrix subject to constraints constructed from the observed entries:

$$\hat{M} = \arg\min_{X} \|X\|_* \quad \text{s.t.} \quad \|\mathcal{P}_\Omega(X - M)\|_F < \epsilon \ . \tag{6.6}$$

Minimizing the nuclear norm $\|X\|_*$ is an effective surrogate for minimizing the rank of X, and solving (6.6) results in a low-rank matrix $\hat{M} = UV^\top$ that approximates the matrix M. One advantage of *A2* over *A1* is that we do not need to constrain the rank of $\hat{M}$ in advance. Note also that the problem defined by (6.6), while being convex, may not necessarily scale up easily to large matrices.

6.2 Local Models for Matrix Approximation

Instead of the global low-rank assumption used in Section 6.1, we assume that the matrix is only locally of low-rank. In this section, we detail how to learn models and how to generalize for predicting unseen ratings, under this local low-rank assumption.

6.2.1 Learning Local Models

In order to facilitate a local low-rank matrix approximation, we need to assume that there exists a metric structure over $[n_1] \times [n_2]$. The distance $d((a, b), (a', b'))$ reflects the similarity between the rows a and a' and columns b and b'. In the case of recommendation systems, for example, $d((a, b), (a', b'))$ expresses the relationship between users a, a' and items b, b'. The distance function may be constructed using the observed ratings $\mathcal{P}_\Omega(M)$ or additional information such as item-item similarity or side information on the users when available.

In the global matrix factorization setting in Section 6.1, we assume that the matrix $M \in \mathbb{R}^{n_1 \times n_2}$ has a low-rank structure. In the local setting, however, we assume that the model is characterized by multiple low-rank $n_1 \times n_2$ matrices. Specifically, we assume a mapping $\mathcal{T} : [n_1] \times [n_2] \to \mathbb{R}^{n_1 \times n_2}$ that associates with each row-column combination $[n_1] \times [n_2]$ a low rank matrix that describes the entries of M in its neighborhood (in particular this applies to the observed entries Ω):

$$\mathcal{T} : [n_1] \times [n_2] \to \mathbb{R}^{n_1 \times n_2} \quad \text{where} \quad \mathcal{T}_{a,b}(a, b) = M_{a,b} \ .$$

Note that in contrast to the global estimate in the previous section, our model now consists of multiple low-rank matrices, each describing the original matrix M in a particular neighborhood.

Without additional assumptions, it is impossible to estimate the mapping $\mathcal{T}$ from a set of $m < n_1 n_2$ observations. We assume, as is often done in non-parametric statistics, that the mapping $\mathcal{T}$ is slowly varying. (See formal definition in the sequel and Figure 6.2.) Since the domain of $\mathcal{T}$ is discrete, the classical definitions of continuity or differentiability are not applicable in our setting. We assume instead that $\mathcal{T}$ is Hölder continuous:

Definition 1. Let X be a metric space. A function $f : X \to \mathbb{R}^{n_1 \times n_2}$ is *Hölder continuous* with parameters $\beta > 0, C > 0$ if

$$\forall x, x' \in X : \ \|f(x) - f(x')\|_F \ \leq \ C\, d^{\beta}(x, x'). \tag{6.7}$$

Figure 6.1 shows a graphic illustration of the locally low-rank linear assumption: the operator $\mathcal{T}$ maps matrix entries to matrices whose image is (a) low-rank, (b) slowly changing, and (c) agrees locally with the original matrix. Assumption (b) implies that if $d(s, r)$ is small $\mathcal{T}(s)$ is similar to $\mathcal{T}(r)$, as shown by their spatial closeness in the embedding $\mathbb{R}^{n_1 \times n_2}$. Assumption (c) implies that for all $s \in [n_1] \times [n_2]$, the neighborhood $\{s' : d(s, s') < h\}$ in the original matrix M is approximately described by the corresponding entries of the low-rank matrix $\mathcal{T}(s)$ (shaded regions of M are matched by lines to the corresponding regions in $\mathcal{T}(s)$ that approximate them).

We would like to emphasize that for illustrative purposes, we assume in Figure 1 that there exists a distance function d whose neighborhood structure coincides with the natural order on indices. That is, $s = (a, b)$ is similar to $r = (c, d)$ if $|a - c|$ and $|b - d|$ are small.

Figure 6.2 shows the relationship between the neighboring entries of the original matrix and the operator image in more detail. The original matrix M (bottom) is described locally by two low-rank matrices $\mathcal{T}(t)$ (near t) and $\mathcal{T}(u)$ (near u). The lines connecting the three matrices identify identical entries: $M_t = \mathcal{T}_t(t)$ and $M_u = \mathcal{T}_u(u)$. The equation at the top right shows a relation tying the three patterned entries. Assuming the distance $d(t, u)$ is small, $\epsilon = \mathcal{T}_u(t) - \mathcal{T}_u(u) = \mathcal{T}_u(t) - M_u(u)$ is small as well.

Following common approaches in non-parametric statistics, we define a smoothing kernel $K_h(s_1, s_2)$, where $s_1, s_2 \in [n_1] \times [n_2]$, as a non-negative symmetric unimodal

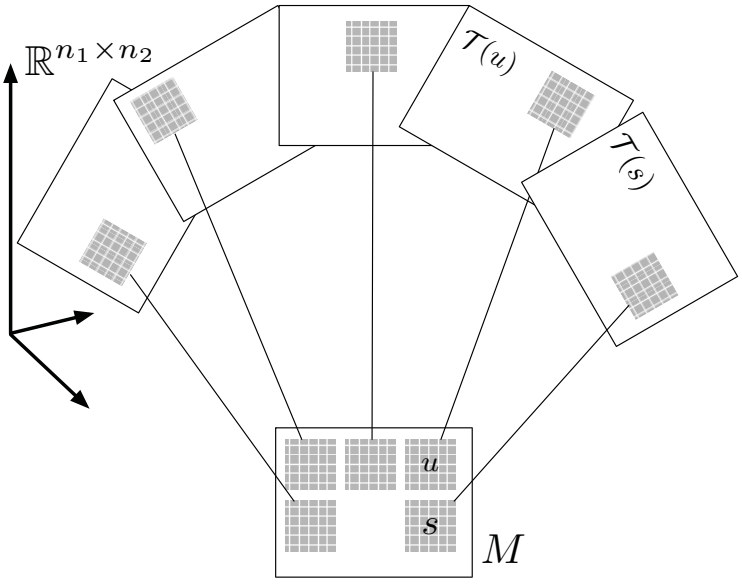

Figure 6.1: The locally low-rank linear assumption assumes an operator that maps matrix entries to matrices whose image is (a) low-rank, (b) slowly changing, and (c) agrees locally with the original matrix. See text for more details.

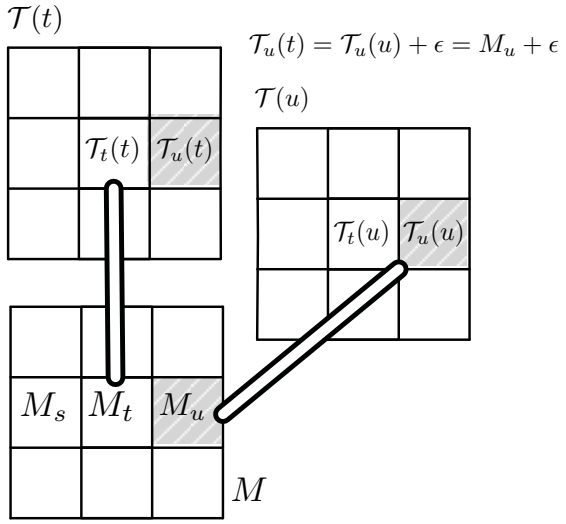

Figure 6.2: The relationship between the neighboring entries of the original matrix and the operator image. See text for more details.

function that is parameterized by a bandwidth parameter $h > 0$. A large value of h implies that $K_h(s, \cdot)$ has a wide spread, while a small h corresponds to narrow spread of $K_h(s, \cdot)$. Often, it is further assumed that $K_h(x) = K_1(x/h)/h$ and that the kernel integrates to 1: $\int K_h(x)\, dx = 1$. In our case, however, we have a discrete domain rather than a continuous domain. See for instance [137] for more information on smoothing kernels. Three popular smoothing kernels are the uniform kernel, the triangular kernel, and the Epanechnikov kernel, defined respectively as

$$K_h(s_1, s_2) \propto \mathbf{1}_{\{d(s_1,s_2)<h\}} \tag{6.8}$$

$$K_h(s_1, s_2) \propto (1 - h^{-1} d(s_1, s_2))\mathbf{1}_{\{d(s_1,s_2)<h\}} \tag{6.9}$$

$$K_h(s_1, s_2) \propto (1 - d(s_1, s_2)^2)\mathbf{1}_{\{d(s_1,s_2)<h\}}. \tag{6.10}$$

We denote by $K_h^{(a,b)}$ the matrix whose (i, j)-entry is $K_h((a, b), (i, j))$.

We describe below the local modifications of incomplete SVD (*A1*) and nuclear norm minimization (*A2*) matrix approximations. Both extensions estimate $\mathcal{T}(a, b)$ in the vicinity of $(a, b) \in [n_1] \times [n_2]$ given the samples $\mathcal{P}_\Omega(M)$.

Local-A1: Incomplete SVD

$$\hat{\mathcal{T}}(a, b) = \arg\min_X \|K_h^{(a,b)} \odot \mathcal{P}_\Omega(X - M)\|_F \quad \text{s.t.} \quad \text{rank}(X) = r \ . \tag{6.11}$$

Local-A2: Nuclear norm minimization

$$\hat{\mathcal{T}}(a, b) = \arg\min_X \|X\|_* \quad \text{s.t.} \quad \|K_h^{(a,b)} \odot \mathcal{P}_\Omega(X - M)\|_F < \epsilon \ . \tag{6.12}$$

The two optimization problems above describe how to estimate $\hat{\mathcal{T}}(a, b)$ for a particular choice of $(a, b) \in [n_1] \times [n_2]$. Conceptually, this technique can be applied at each test entry (a, b), resulting in the matrix approximation $\hat{M} \approx M$ where

$$\hat{M}_{a,b} = \hat{\mathcal{T}}_{a,b}(a, b), \quad (a, b) \in [n_1] \times [n_2] \ .$$

However, such a construction would require solving an optimization problem for each matrix entry (a, b) and is thus computationally prohibitive. Instead, we describe in the next subsection how to use a set of q local models $\hat{\mathcal{T}}(s_1), \ldots, \hat{\mathcal{T}}(s_q)$, where $s_1, \ldots, s_q \in [n_1] \times [n_2]$, to obtain a computationally efficient estimate $\hat{\hat{\mathcal{T}}}(s)$ for all $s \in [n_1] \times [n_2]$.

6.2.2 Global Approximations

The problem of recovering a mapping $\mathcal{T}$ from q values without assuming a strong parametric shape is known as non-parametric regression. We propose using a variation of locally constant kernel regression [137], also known as Nadaraya-Watson local regression:

$$\hat{\hat{\mathcal{T}}}(s) = \sum_{i=1}^{q} \frac{K_h(s_i, s)}{\sum_{j=1}^{q} K_h(s_j, s)} \hat{\mathcal{T}}(s_i). \tag{6.13}$$

Equation (6.13) is simply a weighted average of $\hat{\mathcal{T}}(s_1), \ldots, \hat{\mathcal{T}}(s_q)$, where the weights ensure that values of $\hat{\mathcal{T}}$ at indices close to s contribute more than indices further away from s (note that both the left hand side and the right hand side denote matrices). The denominator in (6.13) ensures that the weighted average sums to one.

In contrast to $\hat{\mathcal{T}}$, the estimate $\hat{\hat{\mathcal{T}}}$ can be computed for all $s \in [n_1] \times [n_2]$ efficiently and off-line (computing $\hat{\hat{\mathcal{T}}}(s)$ simply requires evaluating and averaging $\hat{\mathcal{T}}(s_i)$, $i = 1, \ldots, q$). The resulting matrix approximation is

$$\hat{\hat{M}}_{a,b} = \hat{\hat{\mathcal{T}}}_{a,b}(a, b), \quad (a, b) \in [n_1] \times [n_2]. \tag{6.14}$$

The accuracy of $\hat{\hat{\mathcal{T}}}$ as an estimator of $\hat{\mathcal{T}}$ improves with the number of local models q and the degree of continuity of $\hat{\mathcal{T}}$. The accuracy of $\hat{\hat{\mathcal{T}}}$ as an estimator of $\mathcal{T}$ is limited by the quality of the local estimators $\hat{\mathcal{T}}(s_1), \ldots, \hat{\mathcal{T}}(s_q)$. However, assuming that $\hat{\mathcal{T}}(s_1), \ldots, \hat{\mathcal{T}}(s_q)$ are accurate in the neighborhoods of $s_1, \ldots, s_q$, and q is sufficiently large, the estimation error $\hat{\hat{\mathcal{T}}}_{a,b}(a, b) - \mathcal{T}_{a,b}(a, b)$ is likely to be small. Detailed analysis will be continued in Section 6.3.

6.2.3 View as an Ensemble Method

The local method discussed in this chapter can be seen as an extension from ensemble method in Chapter 5. To see the similarity between them, we recall the prediction formula of both methods. We rewrite (5.8) for the former and (6.13) for the latter with same notations; inputs are user u and item i, each individual models are denoted as f_k, where $k = 1, ..., q$.

Ensembles of Collaborative Filtering (ECF) (Chapter 5):

$$F(u, i) = \sum_{k=1}^{q} \beta_k \, h_k(u, i) \, f_k(u, i) \tag{6.15}$$

$$F(u,i) = \sum_{k=1}^{q} \frac{K_h((u_k,i_k),(u,i))}{\sum_{l=1}^{q} K_h((u_l,i_l),(u,i))} f_k(u,i) \tag{6.16}$$

As we can see from (6.15) and (6.16), both of them 1) combine individual models based on locality using kernel smoothing, 2) predict unseen ratings via weighted sum of individual models, and 3) those weights are dependent on inputs.

The main difference between them is from where f_k comes. In ECF, we assume K individual models are given in advance, and we learn only weights $\beta_k h_k(u,i)$. In this model, locality is applied only to the non-constant weights, as each individual model is learned in a global manner in general. On the other hand, LLORMA learns both local models and their weights at the same time. In other words, LLORMA relaxes one assumption that we already have q recommender systems, extending the range of optimization into individual models as well.

6.3 Theoretical Analysis

In this section we theoretically analyze the estimation accuracy of LLORMA. Our analysis consists of two parts. In the first we analyze the large deviation of $\hat{\mathcal{T}}$ from $\mathcal{T}$. Then, based on this analysis, we derive a deviation bound on the global approximation $\hat{\hat{\mathcal{T}}}$. Our analysis technique is based on the seminal paper of [18]. The goal of this section is to underscore the characteristics of estimation error in terms of parameters such as the train set size, matrix dimensions, and kernel bandwidth.

6.3.1 Analysis of Local Models

Candes and Tao established that it is possible to estimate an $n_1 \times n_2$ matrix M of rank r if the number of observations $m \geq C\mu r n \log^6 n$, where $n = \min(n_1, n_2)$, C is a constant, and μ is the strong incoherence property parameter [18]. This bound is tight in the sense that it is close to the information theoretic limit of $\Omega(r\, n \log n)$.

The aforementioned result is not applicable to our case since the matrix M is not necessarily of low-rank. Concretely, when $r = O(n)$ the bound above degenerates into a sample complexity of $O(n^2 \log n)$ which is clearly larger than the number of entries in the matrix M. We develop below a variation on the results in [18] and [17] that applies to the *local-A2* compressed-sensing estimator $\hat{\mathcal{T}}$.

In our analysis, we make the following assumptions: (i) $\mathcal{T}$ is Hölder continuous, (ii) $\mathcal{T}(s)$ is a rank r matrix that satisfies the strong incoherence property, and (iii) the kernel K_h is a uniform kernel based on a product distance function. The Hölder continuity assumption on $\mathcal{T}$ can be replaced by the following weaker condition without affecting the results

$$\|K_h^s \odot (\mathcal{T}(s) - \mathcal{T}(s'))\|_F \leq \alpha d^\beta(s, s'). \tag{6.17}$$

We denote by $B_h(s)$ the neighborhood of indices near s, $B_h(s) \stackrel{\text{def}}{=} \{s' \in [n_1] \times [n_2] : d(s, s') < h\}$ and we use $n_1(h, s)$ and $n_2(h, s)$ to denote the number of unique row and column indices, respectively, in $B_h(s)$. Finally, we denote $\gamma = \min(n_1(h, s), n_2(h, s))$.

The proposition below provides a bound on the average squared-error within a neighborhood of s

$$\mathcal{E}(\hat{\mathcal{T}})(s, h) = \sqrt{\frac{1}{|B_h(s)|} \sum_{s' \in B_h(s)} \left(\hat{\mathcal{T}}_{s'}(s) - \mathcal{T}_{s'}(s)\right)^2}\ .$$

Proposition 1. *If $|\Omega \cap B_h(s)| \geq C\mu^2\gamma r \log^6 \gamma$, then with probability of at least $1 - \delta$,*

$$\mathcal{E}(\hat{\mathcal{T}})(s, h) \leq \frac{\alpha h^\beta}{\sqrt{|B_h(s)|}} \left(4\sqrt{\frac{\gamma(2+p)}{p}} + 2\right)\ ,$$

where $\gamma = \sqrt[3]{1/\delta}$ and $p = |\Omega \cap B_h(s)|/|B_h(s)|$.

Proof. Assumptions (i) and (iii) above imply that if $K_h(s, s') > 0$ then

$$\|K_h^s \odot (\mathcal{T}(s) - \mathcal{T}(s'))\|_\infty < \alpha h^\beta\ .$$

We can thus assume that if $d(s, s') < h$, an observation $M_{s'} = \mathcal{T}_{s'}(s')$ is equal to $\mathcal{T}_{s'}(s) + Z$ where Z is a random variable whose absolute value is bounded by αh^β. This means that we can use observations $M_{s'} = \mathcal{T}_{s'}(s')$ for estimating the local model $\mathcal{T}(s)$ as long as we admit a noisy measurement process.

Since K is a uniform kernel based on a product distance by assumption (iii), the set $B_h(s)$ is a Cartesian product set. We view this product set as a matrix of dimensions $n_1(h, s) \times n_2(h, s)$ that we approximate. (Note that $n_1(h, s)$ and $n_2(h, s)$ are monotonically increasing with h, and as $h \to \infty$, $n_1(h, s) = n_1$, $n_2(h, s) = n_2$.) The number of observed entries in this matrix approximation problem is $|\Omega \cap B_h(s)|$.

Applying Theorem 7 in [17] to the matrix completion problem described above, we get that if $|\Omega \cap B_h(s)| \geq C\mu^2 \gamma r \log^6 \gamma$, then with probability greater than $1 - \gamma^{-3}$,

$$\|K_h^s \odot (\mathcal{T}(s) - \hat{\mathcal{T}}(s))\|_F \leq \alpha h^\beta \left(4\sqrt{\frac{\gamma(2+p)}{p}} + 2 \right),$$

where $p = \frac{|\Omega \cap B_h(s)|}{|B_h(s)|}$ is the density of observed samples. Dividing by $\sqrt{|B_h(s)|}$ concludes the proof. □

If the observed samples are spread uniformly over the matrix, we have $p = m/(n_1 n_2)$, so

$$\begin{aligned} 4\sqrt{\gamma\frac{2+p}{p}} + 2 &= 4\sqrt{\gamma\frac{2+m/(n_1n_2)}{m/(n_1n_2)}} + 2 \\ &= 4\sqrt{\frac{\gamma(2n_1n_2+m)}{m}} + 2. \end{aligned}$$

Multiplying $\alpha h^\beta / \sqrt{|B_h(s)|}$ yields Corollary 1.

Corollary 1. Assume that the conditions of Proposition 1 hold and in addition the observed samples are spread uniformly with respect to d. Then, the following inequality holds

$$\mathcal{E}(\hat{\mathcal{T}})(s,h) \leq \frac{4\alpha h^\beta}{\sqrt{|B_h(s)|}} \sqrt{\frac{\gamma(2n_1n_2 \,+\, m)}{m}} + \frac{2\alpha h^\beta}{\sqrt{|B_h(s)|}}.$$

If in addition the matrix M is squared ($n_1 = n_2 = n$) and the distribution of distances d is uniform, then $n_1(h,s) = n_2(h,s) = n/h$, $|B_h(s)| = (n/h)^2$, and $\gamma = n/h$. In this case, the bound on $\mathcal{E}(\hat{\mathcal{T}})(s,h)$ becomes

$$4\alpha h^{\beta+1/2} \sqrt{\frac{2n}{m} + \frac{1}{n}} + \frac{2\alpha h^{\beta+1}}{n}. \tag{6.18}$$

In the case of a square matrix with uniformly spread samples, it is instructive to view n, m, h as monotonically increasing sequences, indexed by $k \in \mathbb{N}$ and assume that $\lim_{k\to\infty} n_{[k]} = \lim_{k\to\infty} m_{[k]} = \infty$. In other words, we consider the limit of matrices of increasing sizes with an increasing number of samples. In the case of uniformly distributed distances, the bound (6.18) will converge to zero if

$$\lim_{k\to\infty} \frac{h_{[k]}^{\beta+1}}{n_{[k]}} = \lim_{k\to\infty} \frac{h_{[k]}^{2\beta+1}}{n_{[k]}} = \lim_{k\to\infty} \frac{h_{[k]}^{2\beta+1} n_{[k]}}{m_{[k]}} = 0.$$

6.3.2 Analysis of Global Approximation

We start by showing that $\hat{\mathcal{T}}$ is Hölder continuous with high probability, and then proceed to analyze the estimation error of $\hat{\hat{\mathcal{T}}}$.

Proposition 2. *If $d(s,s') < h$ and Proposition 1 holds at s, s', then with probability at least $1-\delta$,*

$$\|K_h^s \odot (\hat{\mathcal{T}}(s) - \hat{\mathcal{T}}(s'))\|_F \leq \alpha h^\beta \left(8\sqrt{\frac{\gamma(2+p)}{p}} + 5 \right).$$

where $\gamma = \sqrt[3]{2/\delta}$.

Proof. Using the triangle inequality for $\|\cdot\|_F$,

$$\begin{aligned} \|K_h^s \odot (\hat{\mathcal{T}}(s) - \hat{\mathcal{T}}(s'))\|_F &\leq \|K_h^s \odot (\hat{\mathcal{T}}(s) - \mathcal{T}(s))\|_F \\ &+ \|K_h^s \odot (\hat{\mathcal{T}}(s') - \mathcal{T}(s'))\|_F \\ &+ \|K_h^s \odot (\mathcal{T}(s) - \mathcal{T}(s'))\|_F. \end{aligned}$$

We apply the bound from Proposition 1 to the first two terms and use the assumption that $\mathcal{T}$ is Hölder continuous to bound the third term. The adjustment to the confidence level $2\gamma^{-3}$ is obtained using the union bound. □

Proposition 3. *Assume that Proposition 1 holds. Then, with probability of at least $1-\delta$,*

$$\mathcal{E}(\hat{\hat{\mathcal{T}}})(s,h) \leq \frac{\alpha h^\beta}{\sqrt{|B_h(s)|}} \left(12\sqrt{\frac{\gamma(2+p)}{p}} + 7 \right).$$

where $\gamma = \sqrt[3]{(2|\Omega \cap B_h(s)| + 1)/\delta}$.

Proof. Using the triangle inequality we get

$$\begin{aligned} &\|K_h^s \odot (\hat{\hat{\mathcal{T}}}(s) - \mathcal{T}(s))\|_F \leq \\ &\quad \|K_h^s \odot (\hat{\mathcal{T}}(s) - \mathcal{T}(s))\|_F + \|K_h^s \odot (\hat{\hat{\mathcal{T}}}(s) - \hat{\mathcal{T}}(s))\|_F. \end{aligned} \tag{6.19}$$

We bound the first term using Proposition 1. Since $\hat{\hat{\mathcal{T}}}(s)$ is a weighted average of

$\hat{\mathcal{T}}(s_i)$, $i = 1, \ldots, q$ with $s_i \in B_h(s)$, the second term is bounded by

$$\begin{aligned}
&\|K_h^s \odot (\hat{\tilde{\mathcal{T}}}(s) - \hat{\mathcal{T}}(s))\|_F \\
&\quad = \left\| K_h^s \odot \left(\sum_i \frac{w_i}{\sum_j w_j} \hat{\mathcal{T}}(s_i) - \hat{\mathcal{T}}(s) \right) \right\|_F \\
&\quad = \left\| K_h^s \odot \sum_i \frac{w_i}{\sum_j w_j} (\hat{\mathcal{T}}(s_i) - \hat{\mathcal{T}}(s)) \right\|_F \\
&\quad \leq \sum_i \left\| \frac{w_i}{\sum_j w_j} K_h^s \odot (\hat{\mathcal{T}}(s_i) - \hat{\mathcal{T}}(s)) \right\|_F \\
&\quad \leq \sum_i \frac{w_i}{\sum_j w_j} \|K_h^s \odot (\hat{\mathcal{T}}(s_i) - \hat{\mathcal{T}}(s))\|_F.
\end{aligned}$$

There are $|\Omega \cap B_h(s)|$ summands in the above term. We bound each of them using Proposition 2. Together with the bound (6.19) this gives the desired result (after dividing by $\sqrt{|B_h(s)|}$). The adjustment to the confidence level $(2|\Omega \cap B_h(s)| + 1)\gamma^{-3}$ is obtained using the union bound. □

The constants in the proposition above can be improved considerably by using large deviation bounds that are tighter than the union bound.

6.4 Algorithms

We first develop the complete algorithm by concretizing what kernels to use, how to calculate distances, and how to select anchor points in this section. We propose two algorithms, parallel and global LLORMA. The former is computationally more efficient, while the latter is pursuing higher accuracy sacrifcing parallelism.

6.4.1 Parallel LLORMA

In the previous sections, we assumed a general kernel function $K_h(s_1, s_2)$, where $s_1, s_2 \in [n_1] \times [n_2]$. This kernel function may be defined in several ways. For simplicity, we assume a product form $K_h((a, b), (c, d)) = K_{h_1}(a, c) K'_{h_2}(b, d)$ where K and K' are kernels on the spaces $[n_1]$ and $[n_2]$, respectively. We used the Epanechnikov kernel (6.10) for both K, K' as it achieves the lowest integrated squared error [137], but other choices are possible as well.

The distance d in (6.10) can be defined using additional information from an outside source describing row (user) similarity or column (item) similarity. If there is no such

Algorithm 6.1 The Parallel LLORMA Algorithm

1: **Input:** $M \in \mathbb{R}^{n_1 \times n_2}$ whose entries are defined over Ω
2: **Parameters:** kernel function $K(\cdot)$ of widths h_1 and h_2
3: rank r and number of local models q
4: regularization values λ_U, λ_V
5: **for all** $t = 1, \ldots, q$ **parallel do**
6: Select (a_t, b_t) at random from Ω
7: **for all** $i = 1, \ldots, n_1$ **do**
8: Construct entry i: $[K^{a_t}]_i := K_{h_1}(a_t, i)$
9: **end for**
10: **for all** $j = 1, = \ldots, n_2$ **do**
11: Construct entry j: $[K^{b_t}]_j := K_{h_2}(b_t, j)$
12: **end for**
13: Set $(U^{(t)}, V^{(t)})$ to be the minimizer of:
14: $$\sum_{(i,j)\in\Omega} [K^{(a_t)}]_i \, [K^{(b_t)}]_j \left([U\,V^\top]_{i,j} - M_{i,j}\right)^2 + \lambda_U \sum_{i,k} U_{i,k}^2 + \lambda_V \sum_{j,k} V_{j,k}^2$$
15: **end for**
16: **Output:** $\left\{a_t,\, b_t,\, U^{(t)}, V^{(t)}\right\}_{t=1}^{q}$

information available (as is the case in our experiments), d can be computed solely based on the partially observed matrix M. In that case, we may use any distance measure between two row vectors (for K) or two column vectors (for K'). Empirically, we found that standard distance measures such as the 2-norm or cosine similarity do not perform well when M is sparse.

We therefore instead factorize M using standard incomplete SVD (6.4) $M \approx UV^\top$ and then proceed to compute d based on the distances between the rows of factor matrices U and V. Concretely, we used arc-cosine between users i and j:

$$d(i,j) = \arccos\left(\frac{\langle u_i, u_j\rangle}{\|u_i\| \cdot \|u_j\|}\right), \tag{6.20}$$

where u_i, u_j are the i and j rows of the matrix U. We tried numerous other distances and similarity scores such as the Euclidean distance and cosine similarity. The arc-cosine score empirically performed better than the other scores we experimented with.

Besides of the distance metric, the anchor points $(s_1, \ldots, s_q)$ that define $\hat{\bar{\mathcal{T}}}$ also play a significant role. There are several ways of choosing the anchor points. We randomly choose among training data points unless stated otherwise. Detailed discussion is on Section 6.5.3.

Algorithm 6.1 describes the learning algorithm for estimating the local models at the anchor points $\hat{\mathcal{T}}(s_i)$, with $i = 1, \ldots, q$. In line 14, we solve a weighted (by K_{h_1}

and K_{h_2}) SVD problem with L_2 regularization. This minimization problem can be computed with gradient-based methods. After these models are estimated, they are combined using (6.13) to create the estimate $\hat{\hat{\mathcal{T}}}(s)$ for all $s \in [n_1] \times [n_2]$.

The q iterations of the loop in Algorithm 6.1 (lines 5-15) are independent of each other, and thus can be computed in parallel. The complexity of Algorithm 6.1 is q times the complexity of solving a single regularized SVD problem. Note, however, that Algorithm 6.1 may be in fact faster than global SVD since (a) the q loops may be computed in parallel, and (b) the rank used in the local SVD model can be significantly lower than the rank used in a global SVD model. If the kernel K_h has limited support ($K_h(s, s')$ is non-zero only for a few values of s' for any given s) the regularized SVD problems in Algorithm 6.1 would be more sparse than the global SVD problem, resulting in an additional speedup.

Before finishing this section, we comment about the computational complexity of Algorithm 6.1. Kernel smoothing part requires $O(n_1)$ and $O(n_2)$ times, respectively. Also, regularized SVD optimization part requires $O(mr)$. Thus, the overall complexity to learn one low-rank approximation is $O(\max(n_1, n_2) + mr)$. With sufficient parallel processors, our learning process will run within the same time. Practically, it may take several constant times of it.

6.4.2 Global LLORMA

Recall that the parallel LLORMA algorithm from Section 6.4.1 constructs q local models based on q different anchor points. It then combines the models via kernel regression to produce $\hat{M}$ using (6.13) which yields the following approximation,

$$\hat{M}_{u,i} = \sum_{t=1}^{q} \frac{K(u_t, u)K(i_t, i)}{\sum_s K(u_s, u)K(i_s, i)} [U^{(t)} V^{(t)\top}]_{u,i}. \tag{6.21}$$

That is, the algorithm learns each local model independently based on different subsets of the matrix (with some potential overlap). Alternatively, we can directly optimize a joint loss using all local models while bearing in mind the form of the final model as given by (6.21). In this section we describe an algorithmic alternative that minimizes the following loss with respect to $\{U^{(t)}, V^{(t)}\}_{t=1}^{q}$,

$$\sum_{(u,i)\in\Omega} (\hat{M}_{u,i} - M_{u,i})^2 = \sum_{(u,i)\in\Omega} \left(\sum_{t=1}^{q} \frac{K(u_t, u)K(i_t, i)}{\sum_s K(u_s, u)K(i_s, i)} [(U^{(t)} V^{(t)\top}]_{u,i} - M_{u,i} \right)^2 . \tag{6.22}$$

This optimization problem can be solved using gradient-based methods, as we use for the global incomplete SVD. Hence, we solve jointly multiple SVD problems with different weights $(K(u_t, u)K(i_t, i))$ multiplying the original matrix. Since we can decompose M into weighted sums,

$$M_{u,i} = \frac{\sum_t K(u_t, u)K(i_t, i)}{\sum_s K(u_s, u)K(i_s, i)} M_{u,i} \ ,$$

the objective function given by (6.22) can be rewritten as follows,

$$\sum_{(u,i)\in\Omega} \left(\sum_{t=1}^{q} \frac{K(u_t, u)K(i_t, i)}{\sum_s K(u_s, u)K(i_s, i)} [(U^{(t)}V^{(t)\top}]_{u,i} \ - \ \sum_{t=1}^{q} \frac{K(u_t, u)K(i_t, i)}{\sum_s K(u_s, u)K(i_s, i)} M_{u,i} \right)^2$$
$$= \sum_{(u,i)\in\Omega} \left(\sum_{t=1}^{q} \frac{K(u_t, u)K(i_t, i)}{\sum_s K(u_s, u)K(i_s, i)} \left([(U^{(t)}V^{(t)\top}]_{u,i} - M_{u,i} \right) \right)^2 \ . \tag{6.23}$$

Let us now examine the difference between the above objective with the one tacitly employed by parallel LLORMA algorithm, which amounts to,

$$\sum_{(u,i)\in\Omega} \sum_{t=1}^{q} K(u_t, u)K(i_t, i) \left([(U^{(t)}V^{(t)\top}]_{u,i} - M_{u,i} \right)^2 . \tag{6.24}$$

By construction, both objectives are minimized with respect to $\{U^{(t)}, V^{(t)}\}_{t=1}^{q}$ using the squared deviation from M. The difference between the two models is that (6.23) has a square that encompasses a sum over anchor points. When expanded the term includes the individual squared terms that appear in (6.24) as well as additional interaction terms. Namely, parallel LLORMA minimizes the sum of square deviations while global LLORMA minimizes the square deviation of sums. In Algorithm 6.2, we provide the pseudocode of global LLORMA.

A priori we should expect the global version of LLORMA to result in more accurate estimates of M than parallel LLORMA described in Section 6.4.1. However, since the objective can no longer be decoupled, the run time global LLORMA is likely to be longer than its parallel counterpart. We provide experimental results which compare the two versions in terms of performance and running time in Section 6.5.2.

6.5 Experiments

We conducted several experiments with recommendation data. In Section 6.5.1, we compare LLORMA to SVD and other state-of-the-art techniques. We also examine

Algorithm 6.2 The Global LLORMA Algorithm

Input: $M \in \mathbb{R}^{n_1 \times n_2}$ whose entries are defined over Ω
Parameters: kernel function $K(\cdot)$ of widths h_1 and h_2
rank r and number of local models q
regularization values λ_U, λ_V
for all $t = 1, \ldots, q$ **do**
  Select (a_t, b_t) at random from Ω
  for all $i = 1, \ldots, n_1$ **do**
    Construct entry i: $[K^{a_t}]_i := K_{h_1}(a_t, i)$
  end for
  for all $j = 1, = \ldots, n_2$ **do**
    Construct entry j: $[K^{b_t}]_j := K_{h_2}(b_t, j)$
  end for
end for
Minimize with respect to $\{(U^{(t)}, V^{(t)})\}_{t=1}^{q}$:

$$\sum_{(i,j)\in\Omega} \left(\sum_{t=1}^{q} \frac{[K^{(a_t)}]_i \, [K^{(b_t)}]_j \, [U^{(t)} V^{(t)\top}]_{i,j}}{\sum_s [K^{(a_s)}]_i \, [K^{(b_s)}]_j} - M_{i,j} \right)^2 + \lambda_U \sum_{i,k} \left[U_{i,k}^{(t)} \right]^2 + \lambda_V \sum_{j,k} \left[V_{j,k}^{(t)} \right]^2$$

Output: $\left\{ a_t,\, b_t,\, U^{(t)}, V^{(t)} \right\}_{t=1}^{q}$

in the section dependency of LLORMA on the rank r, the number of anchor points q, and the training set size. In Section 6.5.2, we compare the parallel and global versions of LLORMA. Section 6.5.3 introduces several anchor point selection schemes and compare them experimentally. In Section 6.5.4, we apply LLORMA framework to implicit feedback data using complete SVD, followed by a comparison to an ensemble of independent models to verify whether locality plays a role in Section 6.5.5.

We used four popular recommendation systems datasets. The MovieLens[1] dataset is one of the most popular datasets in the literature. We used all versions of MovieLens dataset, namely: 100K ($1K \times 2K$ with 10^5 observations), 1M ($6K \times 4K$ with 10^6 observations), and 10M ($70K \times 10K$ with 10^7 observations). We also tested LLORMA on the Netflix dataset which is of size $480K \times 18K$ with 10^8 observations and the Bookcrossing dataset ($100K \times 300K$ with 10^6 observations). These two datasets are much larger than the MovieLens dataset. We also report results on the Yelp dataset ($40K \times 10K$ with 10^5 observations), which is a recent dataset that is part of the ACM RecSys 2013 challenge[2]. The Bookcrossing and Yelp datasets reflect a recent trend of very high sparsity, often exhibited in real-world recommendation systems.

[1] http://www.grouplens.org/
[2] http://recsys.acm.org/recsys13/recsys-2013-challenge-workshop/

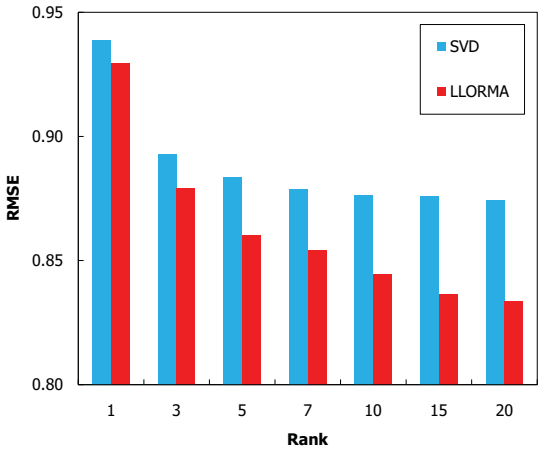

Figure 6.3: RMSE of LLORMA and SVD as a function of the rank on the Netflix dataset.

Unless stated otherwise, we randomly divided the available data into training and test sets such that the ratio of training set size to test set size was 9:1. We created five random partitions and report the average performance over the five partitions. We used a default rating of $(max + min)/2$ for test users or items which lack any rating, where max and min indicate respectively the maximum and minimum possible rating in the dataset.

In our experiments, we used the Epanechnikov kernel with $h_1 = h_2 = 0.8$, a fixed step-size for gradient descent of $\mu = 0.01$, and a 2-norm regularization value of $\lambda_U = \lambda_V = 0.001$. These values were selected using cross-validation. We set and did not attempt to optimize the parameters $T = 100$ (maximum number of iterations), $\epsilon = 0.0001$ (gradient descent convergence threshold), and $q = 50$ (number of anchor points). We selected anchor points by sampling uniformly users and items from the training points without replacement. We examine more complex anchor point selection scheme in Section 6.5.3.

6.5.1 Performance of Parallel LLORMA

Table 6.1 lists the performance of LLORMA with 50 anchor points, SVD, and two recent state-of-the-art methods based on results published in [85]. For a fixed rank r, LLORMA always outperforms SVD. Both LLORMA and SVD perform better as r increases. Both SVD and LLORMA exhibit diminishing returns as the rank increases. LLORMA with a modest rank $r = 5$ outperforms SVD of any rank. We can see that LLORMA also outperforms the APG and DFC algorithms. For a reference, the Root

Table 6.1: RMSE achieved by different algorithms on five datasets: MovieLens 1M, MovieLens 10M, Netflix, Bookcrossing, and Yelp. Results for APG [132] and DFC were taken from [85].

Method	MovieLens 1M		MovieLens 10M		Netflix		Yelp		Bookcrossing	
APG	–		0.8005		0.8433		–		–	
DFC-NYS	–		0.8085		0.8486		–		–	
DFC-PROJ	–		0.7944		0.8411		–		–	
Rank	SVD	LLORMA	SVD	LLORMA	SVD	LLORMA	SVD	LLORMA	SVD	LLORMA
Rank-1	0.9201	0.9135	0.8723	0.8650	0.9388	0.9295	1.4988	1.4490	3.3747	3.1683
Rank-3	0.8838	0.8670	0.8348	0.8189	0.8928	0.8792	1.4824	1.3133	3.3679	3.0315
Rank-5	0.8737	0.8537	0.8255	0.8049	0.8836	0.8604	1.4775	1.2358	3.3583	2.9482
Rank-7	0.8678	0.8463	0.8234	0.7950	0.8788	0.8541	1.4736	1.1905	3.3488	2.8828
Rank-10	0.8650	0.8396	0.8219	0.7889	0.8765	0.8444	1.4708	1.1526	3.3283	2.8130
Rank-15	0.8652	0.8370	0.8225	0.7830	0.8758	0.8365	1.4685	1.1317	3.3098	2.7573
Rank-20	0.8647	0.8333	0.8220	0.7815	0.8742	0.8337				

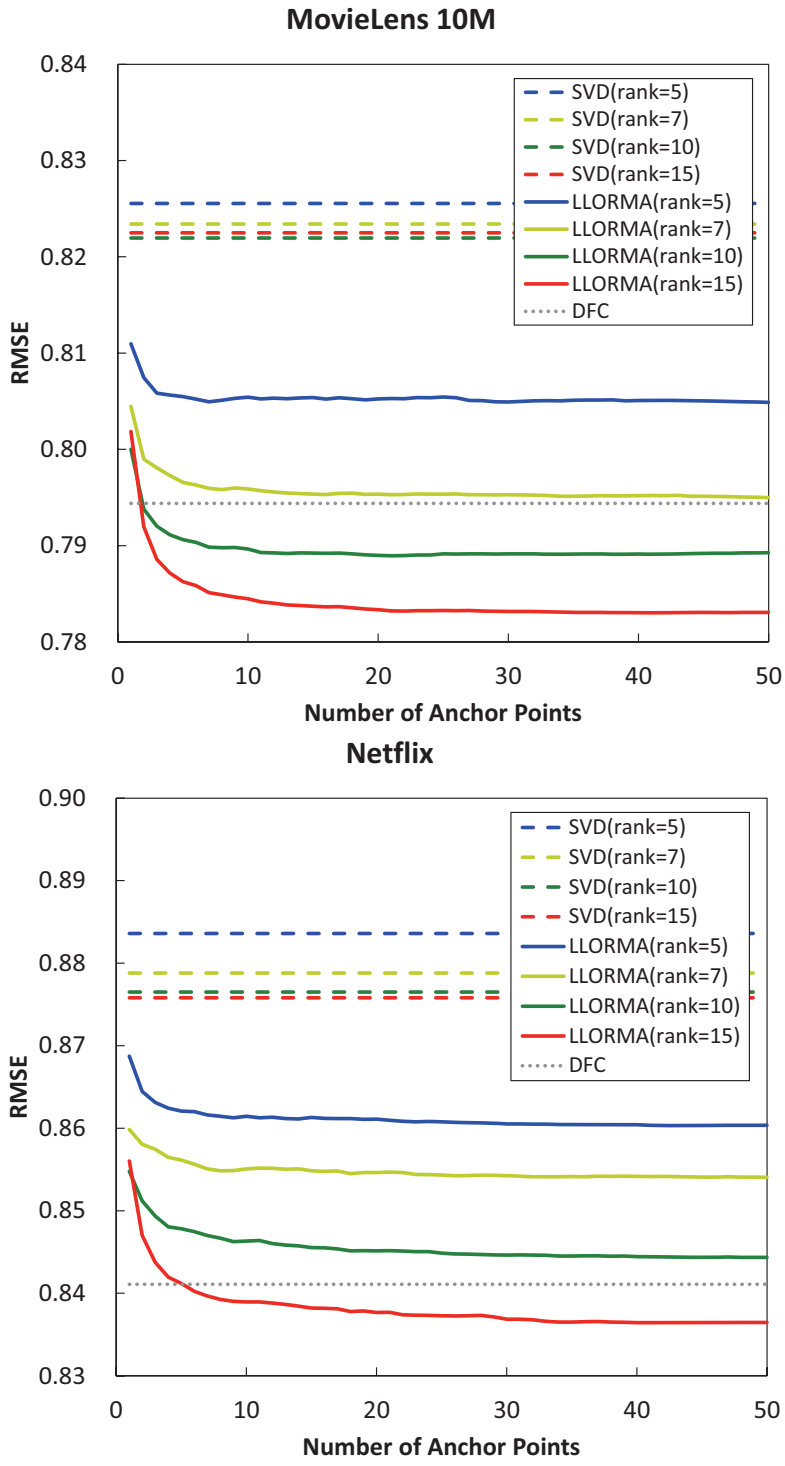

Figure 6.4: RMSE of LLORMA, SVD, and two baselines on MovieLens 10M (top) and Netflix (bottom) datasets. The results for LLORMA are depicted by thick solid lines, while for SVD with dotted lines. Models of the same rank are have identical colors.

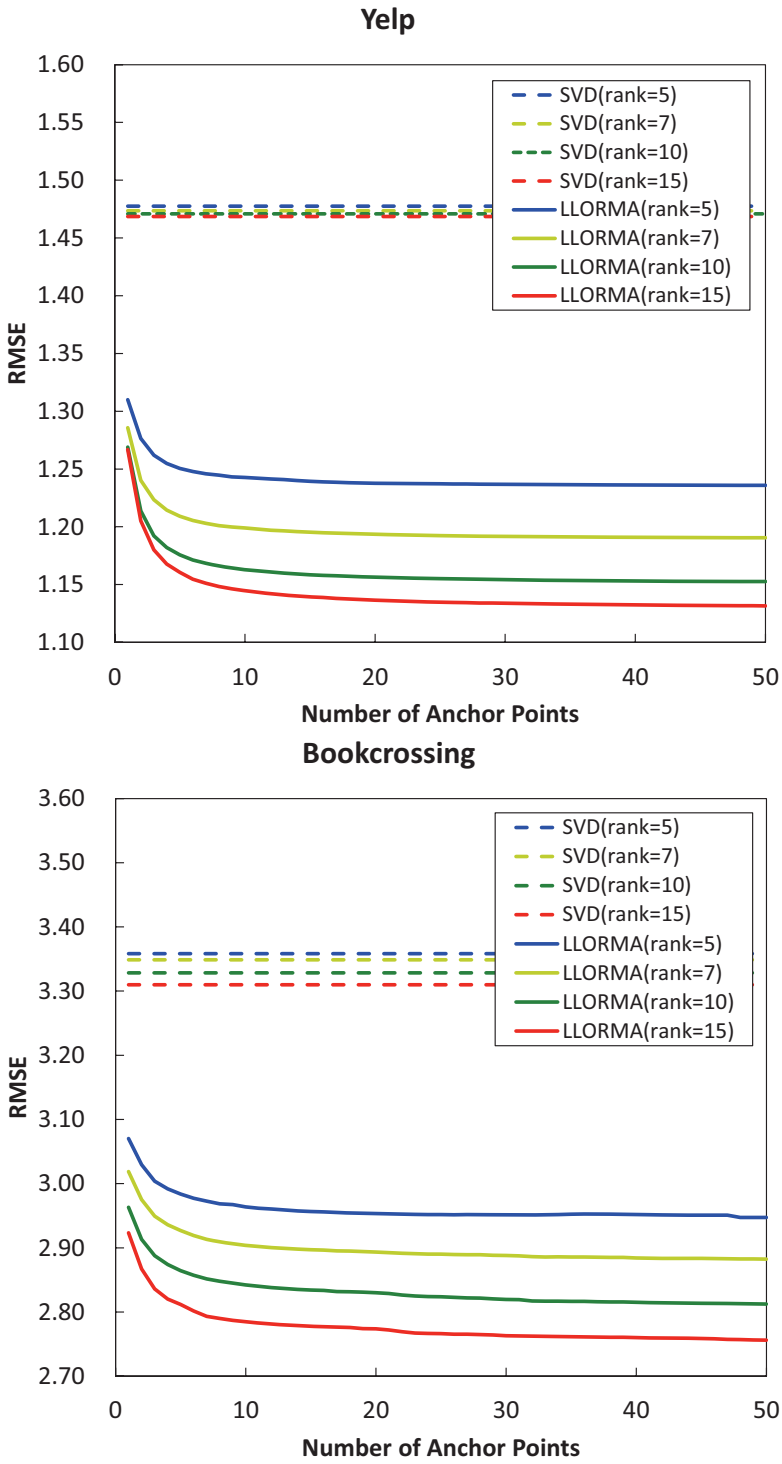

Figure 6.5: RMSE of LLORMA, SVD, and two baselines on Yelp (top), and Bookcrossing (bottom) datasets. The results for LLORMA are depicted by thick solid lines, while for SVD with dotted lines. Models of the same rank are have identical colors.

Mean Square Error (RMSE) we achieved (0.8337) is a better score than the goal of Netflix competition (0.8567).[3]

As we can see from Figure 6.3, the improvement in the performance of SVD is rather minor beyond a rank of 7 while LLORMA's improvement is still evident until a rank of about 20. Both approaches cease to make substantial improvements beyond the aforementioned ranks and exhibit diminishing returns for high ranks. As discussed in earlier sections, the diminishing returns of SVD for high ranks can be interpreted in two ways: (i) The global low-rank assumption is correct and the SVD approximates the rating matrix almost optimally. (ii) The global low-rank assumption is incorrect and the diminishing returns are due to various deficiencies such as overfitting or reaching an inferior local minimum. Since LLORMA improves beyond SVD's best performance, it deems that the second hypothesis is more plausible. The fact that LLORMA's performance asymptotes at higher ranks than SVD may indicate the first hypothesis is to a large extent correct at a local region yet not globally.

Figures 6.4 and 6.5 compares the RMSE of LLORMA, SVD, and the DFC method of [85]. We plot the RMSE of LLORMA as a function of the number of anchor points. As in the case of Table 6.1, both LLORMA and SVD improve as r increases. Here again LLORMA with local rank of at least 5 outperforms SVD of any rank. Moreover, LLORMA outperforms SVD even with only a few anchor points. Figure 6.6 shows the RMSE of LLORMA as a function of the training set size, and compares it with global SVD of 50. We also plot results for a few other methods that have shown to achieve very good approximation accuracy: non-negative matrix factorization (NMF) with rank 50 [79] and Bayesian probabilistic matrix factorization (BPMF) with rank 5 [114]. The test set size was fixed to 10% of the MovieLens 1M and the RMSE was averaged over five random train-test splits. The graph shows that all methods improve as the training set size increases while LLORMA consistently outperforms SVD and the other baselines.

To recap, the experimental results presented thus far indicate that LLORMA outperforms SVD and other state-of-the-art methods even when using relatively lower-rank approximation. Moreover, LLORMA is capable of achieving good accuracy with rather small number of anchor points and seems to perform well across a variety of training set sizes.

[3]We provide this number merely as reference since we measured our performance on a test set different from original Netflix test set, which is no longer available. Our result are based on a random sub-sampling of the Netflix training data as described above. See also the discussion in [85].

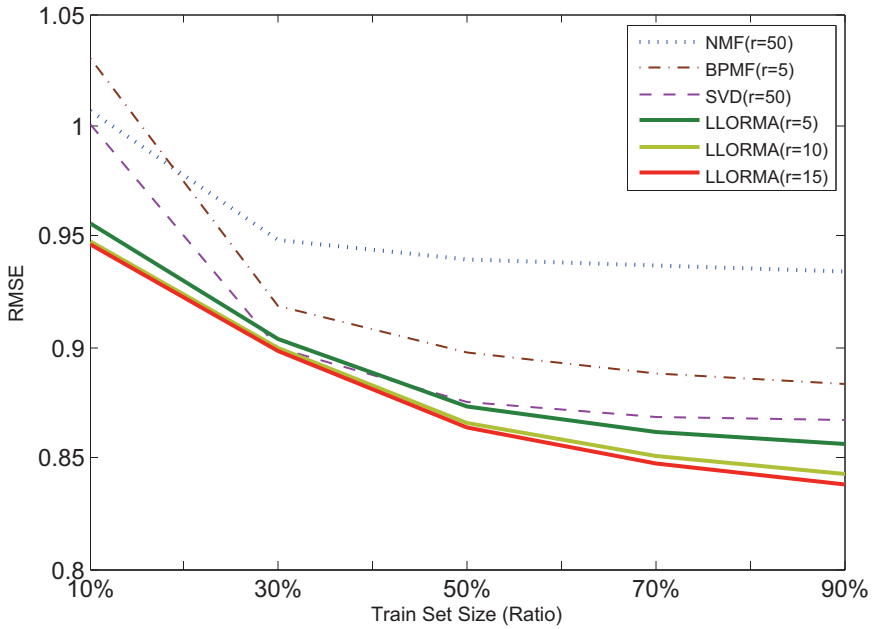

Figure 6.6: RMSE of SVD, LLORMA, NMF, and BPMF methods as a function of training set size for the MovieLens 1M dataset.

6.5.2 Comparison of Global and Parallel LLORMA

We proposed two implementations of LLORMA in this paper: a decoupled parallel approach (Section 6.4.1) and a global approximation version (Section 6.4.2). In earlier sections, we conjectured that the global version is likely to be more accurate in terms of RMSE as it directly optimizes the objective function. In terms of computational efficiency, however, we naturally expected the parallel version to be faster as it can take advantage of multicore and distributed computing architectures. In this subsection, we experimentally verify the two conjectures.

We compared the two versions of LLORMA on MovieLens 100K dataset. We tested local rank values in $\{1, 3, 5, 7, 10\}$. The experiment was conducted on a quad-core machine with 4 threads while suspending any other process. For both versions, we constructed 50 local models and repeated the experiment 5 times with different anchor points. Table 6.2 reports the average test RMSE and average elapsed time for training. As conjectured, global LLORMA results in more accurate estimations on unseen data than parallel LLORMA. However, the performance gap between the two approaches reduces as the rank increases. The parallel version LLORMA runs about 3 times faster than global LLORMA indicating that a fairly high utilization of the multicore architecture.

Table 6.2: RMSE and training time for Global and Parallel LLORMA on MovieLens 100K.

Method	Global LLORMA		Parallel LLORMA	
Rank	Test RMSE	Time	Test RMSE	Time
1	0.9072	6:04	0.9426	1:09
3	0.9020	10:27	0.9117	3:20
5	0.8990	14:40	0.9041	5:26
7	0.8986	19:43	0.9010	7:50
10	0.8975	28:59	0.8985	11:49

6.5.3 Anchor Points Selection

The method for selecting anchor points in LLORMA is important as it may affect the prediction time as well as generalization performance. If the row and column indices of the test data are provided in advance, we may choose anchor points from the test set distribution. However, in most applications the test set is not known apriori, and in fact is likely to increase and change in time. We therefore confined ourselves to three sampling methods and one clustering methods based on low-rank representation of the data. In the rest of the section we use q to denote the number of anchor points. The following are anchor point selection methods we tried.

Complete: Sample anchor points uniformly from the entire set of indices $[n_1] \times [n_2]$.

Trainset: Sample anchor points uniformly from the observed entries, Ω.

Coverage: Select anchor points such that no entry in $[n_1] \times [n_2]$ is too distant from the rest of the anchor points.

k-means: Run k-means clustering on the entries in Ω each represented using the induced d-dimensional space obtained by SVD with $k = q$.

The first two sampling methods are straightforward. The third method, termed "Coverage" was implemented by sampling aq with $a > 1$ user-item pairs and then adding an anchor point whose minimum distance to existing anchor points is the largest. The fourth selection methods that we tested is based on clustering the observed entries. It is based on the observation that an anchor point need not be an entry in the observation matrix but may rather consist of a combination of matrix entries. Recall that we weigh each local model based on user and item similarity. As explained in Section 6.4, each a user (item) is represented as a row of a low-rank matrices U (respectively, V)

attained by the global SVD, namely, $M \approx UV^\top$. Denoting the intrinsic dimension of U and V by d, each user and item can be represented as a d-dimensional vector. Instead of each row of U and V, we can generalize the space of anchor points to any point in this d-dimensional vector space. A good set of anchor points may be the q cluster centers of the d-dimensional representations of the entries of M which is computed using the k-means algorithm.

Table 6.3 provides performance results of *Global* LLORMA using different anchor point selection methods. In the experiments we used the MovieLens 100K datasets with 5 random train-test partitions. The results we report are based on averages over the 5 random splits. As one may anticipate, the different selection schemes perform similarly when q is large. For small values of q (10 or 20), we would like to underscore the following observations. The first two methods (Complete and Trainset) perform similarly. The clustering-based anchor point construction generally performs slightly better than the three other methods. Somewhat surprisingly, the Coverage method performs the worst for small values of q. Nonetheless, when q is sufficiently large all methods achieve about the same results.

Table 6.3: RMSE for various anchor point selection schemes on MovieLens 100K datset.

	10 Local models				20 Local Models			
Rank	Complete	Trainset	Coverage	k-means	Complete	Trainset	Coverage	k-means
1	0.9276	0.9296	0.9360	0.9230	0.9195	0.9206	0.9235	0.9166
3	0.9245	0.9237	0.9327	0.9208	0.9164	0.9152	0.9189	0.9134
5	0.9240	0.9221	0.9277	0.9190	0.9125	0.9128	0.9154	0.9111
7	0.9231	0.9251	0.9279	0.9202	0.9140	0.9129	0.9163	0.9103
10	0.9242	0.9271	0.9301	0.9236	0.9129	0.9125	0.9141	0.9112

6.5.4 Application to Implicit Feedback data

With explicit feedback, users actively score the items they experienced. In most cases, explicit feedback allows users to give both positive and negative reactions. On the other hand, there are many other sources of user feedback without requiring explicit rating actions from users. Even though users do not rate items they have purchased or consumed, the fact that they have chosen to experience those items also may imply some interest on them. For instance, purchase history on an online shopping mall or watching history in video streaming service can be interpreted as their interest on the items.

Implicit data can be represented as a binary matrix; 1 if observed (purchased, watched,

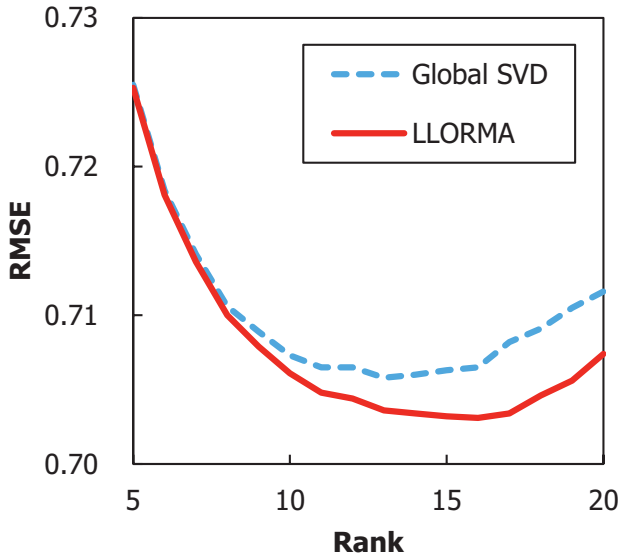

Figure 6.7: Test RMSE of global SVD and LLORMA on implicit feedback data.

and so on), and 0 otherwise. Usually '1' is interpreted as a positive feedback, while '0' as unknown (potentially positive or negative.) However, the interpretation of '1' as a positive feedback can be incorrect in some cases. Suppose a user has streamed a movie for just 5 minutes, then exited and never seen it again. This will imply that the user was disappointed on the movie though she initially started watching it with some interest. Although this interpretation is not as perfect as the explicit dataset case, however, this interpretation is widely used in real applications.

Unlike the explicit data, this matrix is complete. This implies that factorizing this matrix using SVD is very efficient. We can apply LLORMA framework to this complete SVD in the exactly same manner; we first randomly choose an anchor point, and compute user and item similarity to that anchor point. Then, we solve a weighted SVD problem, where the weight in each element is proportional to the user weight times the item weight. We repeat this process with different anchor points (in parallel), then combine them as presented earlier.

To evaluate this idea, we conducted an experiment with MovieLens 100K dataset, comparing performance of global SVD and LLORMA on implicit feedback data. We first converted all numeric ratings to 1, meaning *watched*, to simulate implicit feedback. All other unknown ratings were set to 0. We set aside 20% of watch history as the test set, and took SVD only with the remaining 80% training set. Figure 6.7 shows the test RMSE with global SVD and LLORMA. LLORMA outperforms global SVD in all cases, but the gap gets larger with higher ranks. Also, overfitting was

observed in both methods, but earlier with global SVD around rank 12, while with LLORMA around rank 17. From these observations, we conclude that the local low-rank assumption holds even with implicit feedback data, identifying sub-communities and optimizing each of them effectively.[4]

6.5.5 Comparison to Ensemble of Independent Models

The algebraic form of the parallel estimator $\hat{\hat{\mathcal{T}}}$ as given by (6.13) is a linear combination of local models, each of which focuses on a subset of user-item pairs. This algebraic form is reminiscent of ensemble methods [54] such as Bagging [13], where the final model is a linear combination of simple models, each weighed by a predefined coefficient. Ensemble methods such as boosting and Bagging have been shown to be effective tools for combining models primarily for classification tasks. In this subsection we examine the connection between LLORMA and an ensemble method based on Bagging for low-rank matrix approximation.

There are two main differences between Bagging and LLORMA. In Bagging, the dataset constructed for training each sub-model is uniformly sampled with replacements. In contrast, LLORMA's sampling is based on a non-uniform distribution respecting locality as defined by the distance metric over user-item pairs. The second difference is that Bagging assigns equal weights to each base-model. In LLORMA, each local model is associated with a weight that is proportional to the proximity of the anchor point to the test point. That is, the weights of the local models vary and are determined at inference time.

We conducted two sets of experiments comparing LLORMA with Bagging. In the first experiment, we varied the kernel widths gradually increasing the widths to the matrix dimensions, thus ending with a uniform kernel over Ω. We normalized the kernel width so that a value of 1 corresponds to the full dimension. As the width increases, the overlap between local models becomes more and more substantial. In addition, the bias of each local model increases due to the decrease in locality. Analogously, the variance of each model decreases due to the increase in the actual training set size.

Figure 6.8 shows performance of LLORMA on MovieLens 100K dataset for various kernel widths. The best performance is obtained for kernel width between 0.7 and

[4]We also tested with a real production data from the Amazon Instant Video service, from Nov 2013 to May 2014. We got similar result to the one with MovieLens, but we do not present exact figures as the data is confidential in Amazon.

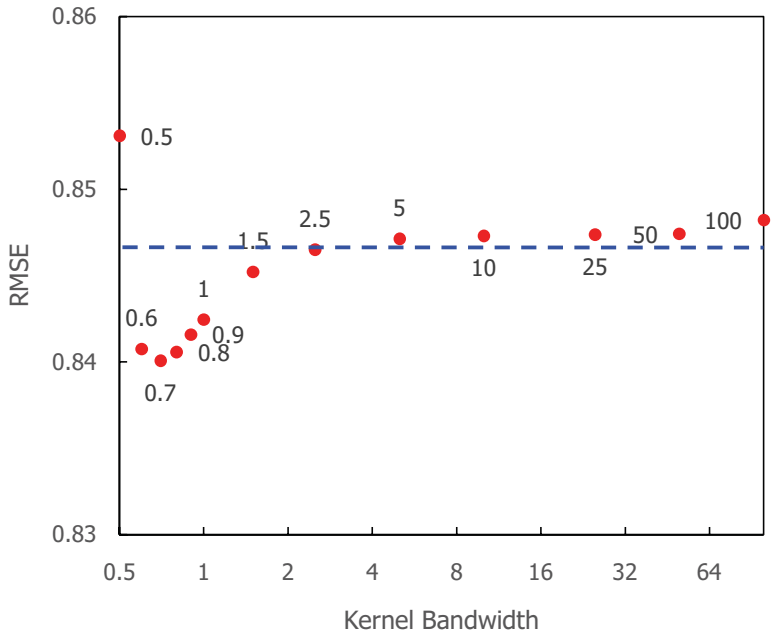

Figure 6.8: Performance of LLORMA as a function of the kernel width. The vertical axis indicates the approximations' root mean squared error (RMSE) and the horizontal axis the kernel width ($h_1 = h_2$) in log scale. The dotted blue line shows the average RMSE of Bagging for reference.

0.8. The performance rapidly deteriorates as the width decreases and slowly degrades as the width increases with an optimal width close to the middle of the range.

In our second experiment, we compared LLORMA with Bagging by taking $|\Omega|$ samples with replacements, which in expectation covers two thirds of the observed entries. Table 6.4 compares the performance of global SVD of rank 10 and 15, LLORMA with 100 local models, and Bagging with 100 models. Each result is the average of 5 random splits of the dataset. It is apparent from the table that both LLORMA and Bagging outperform global SVD. Further, LLORMA achieves lower RMSE than Bagging. The improvement of LLORMA over Bagging is statistically significant based on a paired t-test with p-values of 0.0022 for MovieLens 100K and 0.0014 for MovieLens 1M. These p-values correspond to a confidence level over 99%. LLORMA also outperforms Bagging with respect to the mean absolute error (MAE).

6.6 Nuclear Norm Minimization

The motivation for performing the experiments in the previous section with the Frobenius norm minimization framework (incomplete SVD in Equation (6.11)) rather than the nuclear norm minimization framework (Equation (6.12)) is due to the better

Table 6.4: Comparison of the mean absolute error (MAE) and root mean squared error (RMSE) for SVD, Bagging, and LLORMA on MovieLens dataset.

Dataset	MovieLens 100K		MovieLens 1M	
Method	MAE	RMSE	MAE	RMSE
SVD (rank=10)	0.7189	0.9108	0.6922	0.8683
SVD (rank=15)	0.7170	0.9094	0.6913	0.8676
Bagging	0.6985	0.8930	0.6620	0.8481
LLORMA	0.6936	0.8881	0.6577	0.8423

computational scalability of the Frobenius norm approach and its huge popularity. Nevertheless, it is worth to see whether the local low-rank assumption and LLORMA framework is still useful in case of nuclear norm minimization problem as well. In this section, we briefly introduce nuclear norm minimization problem in Section 6.6.1 and how to adapt a basic algorithm into LLORMA framework in Section 6.6.2, followed by experiments evaluating the performance of nuclear norm minimization LLORMA in Section 6.6.3.

6.6.1 Nuclear Norm Minimization Problem

Matrix completion in compressed sensing is usually stated as the following optimization problem:

$$\hat{M} = \arg\min_{X} \operatorname{rank}(X) \quad \text{s.t.} \quad X_{i,j} = M_{i,j} \quad (i,j) \in \Omega \tag{6.25}$$

where M is the partially observed rating matrix and X is the output matrix with all missing values filled. This optimization problem aims to find a matrix with as low rank as possible after filling the missing entries in M. As the objective function in (6.25) is our ultimate goal, it is unfortunately non-convex optimization problem. As an alternative, analogous to the convex relaxation from L_0 norm minimization to L_1 norm minimization problem, it is common to minimize the nuclear (trace) norm of X instead:

$$\hat{M} = \arg\min_{X} \|X\|_* \quad \text{s.t.} \quad X_{i,j} = M_{i,j} \quad (i,j) \in \Omega \tag{6.26}$$

where $\|X\|_*$ is the sum of singular values of X. It is known that this is the tightest convext relaxation of (6.25).

There are many approaches solving this optimization problem in literature. Subgradient descent approaches were proposed in [58, 2]. Formulating nuclear norm minimization problem as a semi-definite programming [35, 124, 55, 45] is another

widely-studied area. Singular Value Thresholding [16] is a simple and fast method, iteratively cutting singular values down to achieve minimal rank. Active subspace selection [51] approached this problem in a divide-and-conquer manner, iteratively selecting an active subspace using the proximal gradient, finding a smooth, convex relaxation of the smaller sub-space problem, and solving these using second order methods. Other recent improvements in this area include [33, 149, 91, 84, 19, 87, 25].

In this thesis, we adapted the Singular Value Thresholding and the Active Subspace Selection methods to our locally weighted variation. We compare the performance of those methods learned in a global manner, and those based on combination of local models. For this, we first explain how we modify these methods to LLORMA version in the next subsection.

6.6.2 Adapted SVT Algorithm

We rewrite the constrained nuclear norm minimization problem

$$\hat{M} = \arg\min_X \|X\|_* \quad \text{s.t.} \quad \|\mathcal{P}_\Omega(X - M)\|_F < \epsilon \ .$$

in its Lagrangian form:

$$\hat{M} = \arg\min_X \|\mathcal{P}_\Omega(X - M)\|_F^2 + \tau\|X\|_* \quad \text{where } \tau > 0. \tag{6.27}$$

We first repeat the definition of the following soft-thresholding operator $\mathcal{D}_\tau(Y)$ from [16]: for $\tau > 0$ and $Y \in \mathbb{R}^{n_1 \times n_2}$,

$$\mathcal{D}_\tau(Y) \stackrel{\text{def}}{=} U \text{ diag } \{(\sigma_i - \tau)_+\} V^\top \tag{6.28}$$

where $t_+ = \max(t, 0)$ and the SVD of Y is given by $U\Sigma V^\top$ with $\Sigma = \text{diag } (\sigma_1, ..., \sigma_r)$.

The SVT algorithm iteratively applies this operator:

$$\begin{cases} X^k = \mathcal{D}_\tau(Y^{k-1}) \\ Y^k = Y^{k-1} + \delta_k \mathcal{P}_\Omega(M - X^k) \end{cases} \tag{6.29}$$

until a stopping condition is reached, where $Y^0 = 0$, $k = 1, 2, 3, ...$, and $\{\delta_k\}$ is a sequence of positive step sizes ($\mathcal{P}_\Omega$ is the projection operator defined in Section 6.1).

Theorem 2 below, which is originally found in [16], argues that the iterative algorithm above (6.29) converges to the unique minimizer of the nuclear norm regularized minimization problem.

Theorem 2. For each $\tau \geq 0$ and $Y \in \mathbb{R}^{n_1 \times n_2}$, the singular value shrinkage operator (6.28) satisfies $\mathcal{D}_\tau(Y) = \arg\min_X \frac{1}{2}\|X - Y\|_F^2 + \tau\|X\|_*$.

Proof of the theorem can be found in Theorem 2.1 of [16].

Using Theorem 2, we propose an iterative algorithm similar to (6.29) for learning each local model in the LLORMA framework. Recall that we want to solve (6.12):

$$\hat{M} = \arg\min_X \|X\|_* \quad \text{s.t.} \quad \|K_h^{(a,b)} \odot \mathcal{P}_\Omega(X - M)\|_F < \epsilon$$

or equivalently

$$\hat{M} = \arg\min_X \frac{1}{2}\|K_h^{(a,b)} \odot \mathcal{P}_\Omega(X - M)\|_F^2 + \tau\|X\|_*. \tag{6.30}$$

Assuming that we use a multiplicative kernel $K_h^{(a,b)} = K_h^a \cdot K_h^b$ (where K_h^a reflects similarity between the anchor user a and all other users, and K_h^b between the anchor item b and all other items) (6.30) can be rewritten as

$$\hat{M} = \arg\min_X \frac{1}{2}\|\mathcal{P}_\Omega(A(X - M)B)\|_F^2 + \tau\|X\|_*. \tag{6.31}$$

where A and B are diagonal matrices containing $K_h(a, i)$ in $A_{i,i}$ and $K_h(b, j)$ in $B_{j,j}$. In other words, $A = \text{diag}(K_h^a)$ and $B = \text{diag}(K_h^b)$.

We propose the following adaptation of the SVT algorithm for solving the above problem (6.31):

$$\begin{aligned} X^k &= A^{-1}\mathcal{D}_\tau(AY^{k-1}B)B^{-1} \\ Y^k &= Y^{k-1} + \delta_k\mathcal{P}_\Omega(M - X^k). \end{aligned} \tag{6.32}$$

We prove that this algorithm correctly solves the optimization problem (6.31) in a similar way to Theorem 2.

Theorem 3. For each $\tau \geq 0$ and $Y \in \mathbb{R}^{n_1 \times n_2}$, the singular value shrinkage operator (6.28) satisfies $A^{-1}\mathcal{D}_\tau(AY^{k-1}B)B^{-1} = \arg\min_X \frac{1}{2}\|A(X - Y)B\|_F^2 + \tau\|X\|_*$, where $A = \text{diag}(K_h^a)$ and $B = \text{diag}(K_h^b)$.

Proof. Let $h_1(X) = \frac{1}{2}\|A(X - Y)B\|_F^2 + \tau\|X\|_*$. As A and B are just linear transformations, $h_1(X)$ is still strictly convex. We need to show that its minimizer is equal to $A^{-1}\mathcal{D}_\tau(AY^{k-1}B)B^{-1}$.

Similarly to Theorem 2, $\hat{X}$ minimizes h_1 if and only if $\mathbf{0}$ is a subgradient of h_1 at $\hat{X}$, that is,

$$\mathbf{0} \in A(\hat{X} - Y)B + \tau\partial\|\hat{X}\|_* \quad \Leftrightarrow \quad A(\hat{X} - Y)B \in \tau\partial\|\hat{X}\|_*.$$

Denoting the SVD of AYB by $U_0\Sigma_0V_0^\top + U_1\Sigma_1V_1^\top$ where the first term (subscripted with 0) corresponds to singular values smaller than τ, while the second term (subscripted with 1) the others. Assuming that $\hat{X} = A^{-1}\mathcal{D}_\tau(AY^{k-1}B)B^{-1}$, we have $A\hat{X}B = U_0(\Sigma_0 - \tau I)V_0^\top$ by definition of the soft-thresholding operator. Then,

1. $A(Y - \hat{X})B = \tau(U_0V_0^\top + W)$ where $W = \tau^{-1}U_1\Sigma_1V_1^\top$,
2. $U_0^\top W = U_0^\top(\tau^{-1}U_1\Sigma_1V_1^\top = \tau^{-1}(U_0^\top U_1)V_1^\top = \mathbf{0}$,
3. $WV_0 = (\tau^{-1}U_1\Sigma_1V_1^\top)V_0 = \tau^{-1}U_1\Sigma_1(V_1^\top V_0) = \mathbf{0}$,
4. $\|W\|_2 = \tau^{-1}\|U_1\Sigma_1V_1^\top\|_2 = \tau^{-1}\|\Sigma_1\|_2 \leq 1$, as Σ_1 is a diagonal matrix with $\sigma_i < \tau$ for all i.

By using the known fact $\partial\|X\|_* = \{UV^\top + W : W \in \mathbb{R}^{n_1\times n_2}, U^\top W = 0, WV = 0, \|W\|_2 \leq 1\}$, we conclude that $A(Y - \hat{X})B \in \tau\partial\|\hat{X}\|_*$, so $\hat{X} = A^{-1}\mathcal{D}_\tau(AY^{k-1}B)B^{-1}$ is the unique minimizer of the optimization problem $\arg\min_X \frac{1}{2}\|A(X - Y)B\|_F^2 + \tau\|X\|_*$. □

We note that the optimization problem above is a special case of "general convex constraints" in Section 3.2 of [16] corresponding to

$$\begin{aligned} X^k &= \mathcal{D}_\tau(L^\top y^{k-1}) \\ y^k &= Y^{k-1} + \delta_k(b - L(X^k)) \end{aligned} \tag{6.33}$$

where L is a linear transformation mapping.

We modified the Active Subspace Selection method in a similar manner. To simplify things, we used uniform kernel, which takes training examples with sufficiently similar ones to the anchor points with full weights, while drops otherwise.

6.6.3 Experimental Result and Discussion

Since nuclear norm minimization does not scale to large matrix sizes as incomplete SVD, we describe in this section an experiment with the relatively small MovieLens 100K dataset. We used parallel LLORMA with uniform sampling of anchor

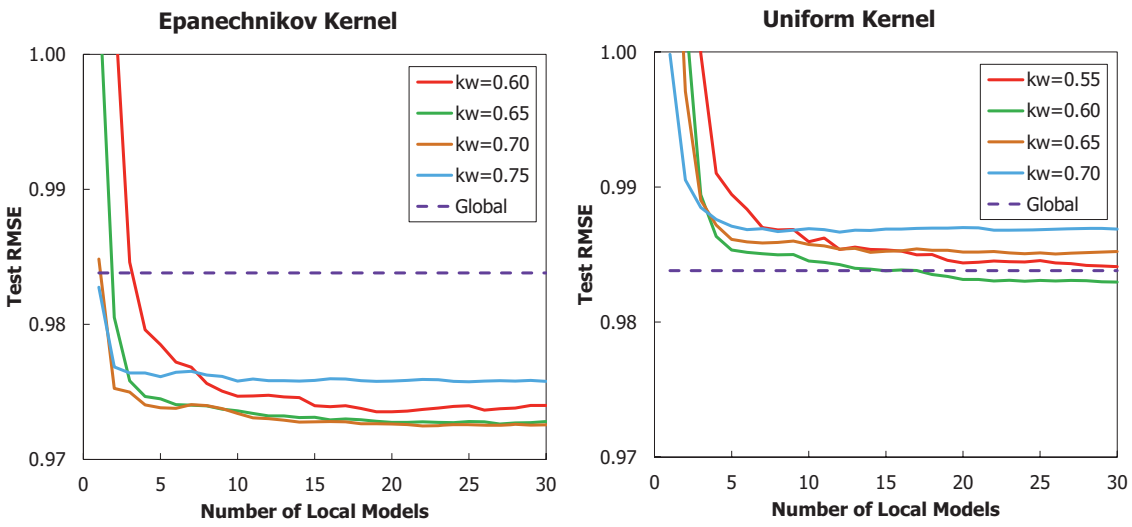

Figure 6.9: Experimental results with nuclear norm minimization using Singular Value Thresholding.

point selection from the training data. We tried two types of kernels, uniform and Epanechnikov with different bandwidths as shown in Figure 6.9. (Bandwidths outside of this range such as 0.8 did not perform as well.) The distance function was identical to the distance function in the Frobenius norm case (see Section 6.5). The learning rate was determined by cross validation. We split the dataset into train and test sets randomly with a 9:1 ratio and repeated 5 times with different random train/test splits and anchor point selections and averaged the result.

The experimental results are shown in Figure 6.9 (using Singular Value Thresholding) and in Figure 6.10 (using Active Subspace Selection). We observe the following points:

1. LLORMA performs better than the global version of nuclear norm minimization with both SVT and Active Subspace Selection. In case of SVT, LLORMA reached an RMSE of 0.9725 with Epanechnikov kernel and the best kernel width, while the global version reached 0.9838. With Active Subspace Selection, LLORMA achieved RMSE of 0.9583, while the global (original) version did 0.9611.

2. LLORMA works better with Epanechnikov kernel (left panel of Figure 6.9) than uniform kernel (right panel of Figure 6.9) in nuclear norm minimization problem. This agrees with the experimental results in Section 6.5 as well. In case of the Active Subspace Selection experiment, LLORMA with uniform kernel outperformed the global counterpart.

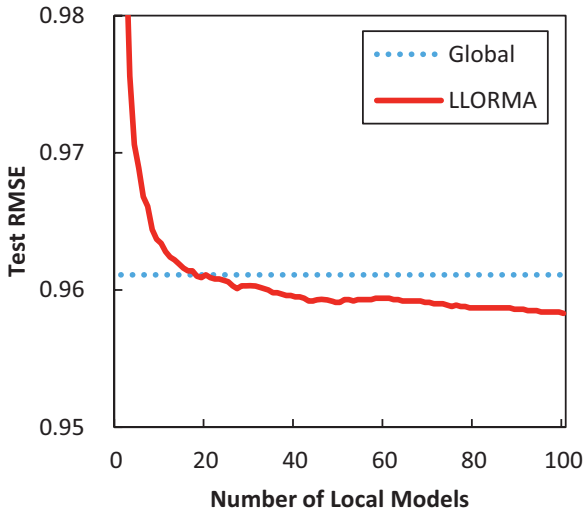

Figure 6.10: Experimental results with nuclear norm minimization using Active Subspace Selection.

3. The best kernel widths for nuclear norm minimization (0.7 for Epanechnikov kernel and 0.6 for uniform kernel) were different from the Frobenius norm case (0.8). This is an expected result, as kernel width is an intrinsic parameter of each kernel.

4. The improvement from global to local approaches is smaller than in the case of the Frobenius norm minimization. In the case of the Frobenius norm minimization, the improvement was from 3.6% to 4.9%, while it was lower (0.9%) in the case of the nuclear norm.

5. Nuclear norm minimization does not perform as well as Frobenius norm minimization (as measured by test RMSE). This confirms other results in the literature: 0.95 in [69] and around 0.96 in [51] on the same MovieLens 100K dataset. There are more reports on Netflix dataset [55, 87] with comparable performance to the baseline RMSE score (0.9525) of the Netflix competition.

Overall, we conclude that local low-rank assumption is still valid in nuclear norm minimization problem, but it is less effective than the Frobenius norm case.

6.6.4 Computational Issues

Although nuclear norm minimization is known as the tightest convex relaxation to the low-rank matrix factorization problem, it is limitedly used in real-world large-scale applications due to its heavy computational overhead. Fortunately, efficiency

and scalability have been improved in addition to prediction accuracy in this area. For example, the Active Subset Selection method [51] is a recent algorithm, which significantly improved the running time of nuclear norm minimization over classical methods such as Singular Value Thresholding.

Using more efficient method for base models also improves performance of LLORMA. In an ideal setting where we have sufficient number of machines to solely run each local model, the overall running time will be determined by the performance of the base model. We observed the time taken to run LLORMA minimization nuclear norm is XXX times faster with Active Subspace Selection than with Singular Value Thresholding. Also, we may be able to further improve performance by optimizing each base model. For instance, SVT-based LLORMA may run slightly faster when we do partial SVD rather than full SVD, stopping to compute SVD when we encounter zeros in singular values.

6.7 Summary

In this chapter, we presented a new approach for low-rank matrix approximation based on the assumption that the matrix is locally low-rank. LLORMA is highly parallelizable and thus scales well with the number of observations and the dimension of the problem. Our experiments indicate that LLORMA outperforms several state-of-the-art methods without a significant computational overhead. We also presented a formal analysis of LLORMA by deriving bounds that generalize standard compressed sensing results and express the dependency of the modeling accuracy on the matrix size, training set size, and locality (kernel bandwidth parameter). Our method is applicable beyond recommendation systems so long as the locality assumption holds and a reasonable metric space can be identified.

Chapter VII

LOCAL COLLABORATIVE RANKING

In contrast to standard collaborative filtering models, LLORMA in Chapter 6 does not assume that the matrix M of user-item ratings is low-rank. Instead, we assume that M is locally low-rank where locality is defined by a neighborhood with respect to a given metric on pairs of (row, column) indices. In this chapter, instead of squared loss reconstruction in LLORMA, we consider a ranked loss minimization setting that is more appropriate for real world recommendation systems. The resulting method, Local Collaborative Ranking (LCR), outperforms standard ranked collaborative filtering models as well as more traditional CF methods. Furthermore, due to the locality assumption, this approach is highly parallelizable and scales up to large scale datasets.

We first formalize the ranking problem in Section 7.1. In Section 7.2, we describe our model in detail followed by experimental analysis in Section 7.3.

7.1 Label Ranking

A general label ranking problem is defined as follows. Let X be a domain of instances and Y be a set of labels, possibly of infinite cardinality. A *label ranking* for an instance $x \in X$ is a total ordering over Y, where $y_1 \succ y_2$ implies that y_1 is preferred to y_2 for x. A *label ranking function* $f : X \times Y \to \mathbb{R}$ returns a score indicating how much a label y is relevant to an instance x. We learn f so as to preserve preference order as much as possible on training data. That is, given x, for all $y_1 \succ y_2$ relationship, we fit f to satisfy $f(x, y_1) > f(x, y_2)$ (if possible).

In the context of recommendation systems, X is a set of users $\mathcal{U}$ and Y is a set of items $\mathcal{I}$. The function $f(u, i)$ estimates a preference score for a user $u \in \mathcal{U}$ on an item $i \in \mathcal{I}$. We fit f such that $f(u, i_1) > f(u, i_2)$ if user u prefers item i_1 to item i_2.

One way to learn this ranking function is to sort the output of a rating predictor using the approach in Section 6.1. With this approach, f directly approximate $f(u, i) \approx M_{u,i}$ for all (u, i) pairs on the training set Ω. With this ranking function f, the ordered sequence of $f(u, 1), \ldots, f(u, n)$ gives the total order over the labels for user u. (Recall

that n is the number of items.) This approach is called a "point-wise" method.

Although it is straightforward to learn to approximate ratings with point-wise techniques, doing so suffers from calibration problem [42], or in other words, the interpretation of scores for one user may not be consistent with the interpretation of the same scores by another user. Directly approximating rating values (rather than the ordering of the ratings) misses this effect and may thus lead to lower accuracy.

However, for both users an ordered pair of items means the same thing: one item is preferred to another. The "pair-wise" approach avoids these drawbacks by considering the preference order seen in training data, rather than directly estimating the values themselves. With this approach, $f(u, i)$ does not necessarily fit to $M_{u,i}$. Instead, it tries to preserve the relative order of preferences between two ratings from the same user. That is, it minimizes a risk function

$$\mathcal{E}(f) = \sum_{u \in \mathcal{U}} \sum_{i,j \in M_u} \mathcal{L}(f(u,i) - f(u,j), M_{u,i} - M_{u,j}) \tag{7.1}$$

where M_u is the set of items rated by the user u. One choice of loss function $\mathcal{L}(x, y)$ is the zero-one loss, penalizing only when the signs of x and y disagrees. A differentiable approximation for this function will be introduced later.

One drawback of the pair-wise method is its computational complexity. As the pair-wise method sums over all pairs of ratings by the same user, its worst-case complexity is $O(k^2)$ for k ratings or per rating average of $O(k^2/m)$. Obviously, it takes much longer than a point-wise method, whose complexity is $O(k)$. Fortunately, however, rating matrices are usually very sparse, so the number of pairs may not be intractably large. Also, since many recommendation ratings lie in a finite set of scores (for example one to five stars), many ties are bound to occur and dropping these ties leads to manageable complexity.

7.2 Collaborative Ranking

We now turn our attention to the main thrust of the paper where we present our learning-to-rank algorithm by minimizing a pair-wise loss function. With a pair-wise loss, we are not interested in the absolute ratings but rather in the *difference* in preference values. Let (i, j) denote the indices of two items rated by a user u. Without loss of generality we assume that item i is preferred over item j, namely $M_{u,i} > M_{u,j}$. The training set for pair-wise losses is constructed using observed entries from Ω as

Table 7.1: Pair-based loss functions. The scalar γ is a free parameter which designate a target margin value. [M] stands for a multiplicative version, and [A] for an additive version.

Loss Name	Loss Function
Log-loss[M]	$\mathcal{L}_{\text{Log[M]}}(\Delta M, \Delta f) = \Delta M \log(1 + \exp\{\gamma - \Delta f\})$
Log-loss[A]	$\mathcal{L}_{\text{Log[A]}}(\Delta M, \Delta f) = \log(1 + \exp\{\gamma + \Delta M - \Delta f\})$
Exp-loss[M]	$\mathcal{L}_{\text{Exp[M]}}(\Delta M, \Delta f) = \Delta M \exp\{\gamma - \Delta f\}$
Exp-loss[A]	$\mathcal{L}_{\text{Exp[A]}}(\Delta M, \Delta f) = \exp\{\gamma + \Delta M - \Delta f\}$
Hinge-loss[M]	$\mathcal{L}_{\text{Hinge[M]}}(\Delta M, \Delta f) = \Delta M[\gamma - \Delta f]_+$
Hinge-loss[A]	$\mathcal{L}_{\text{Hinge[A]}}(\Delta M, \Delta f) = [\gamma + \Delta M - \Delta f]_+$

follows

$$\{(u, i, j, M_{u,i} - M_{u,j}) : (u, i), (u, j) \in \Omega, M_{u,i} > M_{u,j}\}. \tag{7.2}$$

We use the same factorization form $UV^\top$ for our model with $U \in \mathbb{R}^{m \times r}$ and $V \in \mathbb{R}^{n \times r}$ where $r \ll \min(m, n)$. In contrast to the single rating approach presented in Section 6.1, $\hat{M}_{u,i}$ would not necessarily be an approximation to $M_{u,i}$ in value. That is, for a pair $M_{u,i} > M_{u,j}$, our low-rank estimation should conform with the relative order and difference in values of $\hat{M}_{u,i}$ and $\hat{M}_{u,j}$.

In the sequel, we discuss pair-based loss functions. Then, we describe a natural adaptation of global SVD from Section 6.1 to pair-wise loss functions. Finally, we describe the *Local Collaborative Ranking (LCR)* method.

7.2.1 Pair-wise loss functions

A natural loss we consider in ranking is the rate of preference disagreement, namely, how many pairs are mis-ordered by the ranking model. We refer to this loss as the zero-one error for pairs,

$$\mathcal{E}(f) = \sum_{u \in \mathcal{U}} \frac{1}{s_u} \sum_{k=1}^{s_u} \mathbf{1}[\Delta M_k \cdot \Delta f(x_k) < 0], \tag{7.3}$$

where x_k designates an ordered pair rated by user u. ΔM_k and $\Delta f(x_k)$ denote the difference in observed and predicted rating for the pair. s_u is the number of ordered items rated by the user u. We normalize each user's pair disagreement number by s_u so that each user is weighed equally. The operator $\mathbf{1}$ designates the indicator function, $\mathbf{1}[\texttt{true}] = 1$ and $\mathbf{1}[\texttt{false}] = 0$. The error described by (7.3) amounts to the average number of pairs for which the predicted preference disagrees with the observed one and can be viewed as an instance of the generalized ranked loss in [30].

The zero-one loss (7.3) is not differentiable and finding an f that minimizes it is computationally intractable. Instead, we use a smooth surrogate loss function $\mathcal{L}$ that forms a convex upper bound on the zero-one loss function

$$\mathcal{E}(f) = \sum_{u \in \mathcal{U}} \frac{1}{s_u} \sum_{k=1}^{s_u} \mathcal{L}(\Delta M_k, \Delta f(x_k)). \tag{7.4}$$

We describe and experiment with two different families of margin-based loss functions, provided in Table 7.1. The first scales the loss by ΔM, while the second constructs an additive margin using ΔM. These losses are analogous to similar relaxations of the zero-one loss in classification.

7.2.2 A Global Approximation

Similarly to the incomplete SVD in Section 6.1, we can learn a global model by minimizing the loss described in (7.4). The minimization produces two matrices U, V that when multiplied UV^T they provide an estimate for ratings whose ordering conforms to the ordering in the training data. To simplify our derivation we define a ternary function g as follows:

$$\begin{aligned} g(u,i,j) &\stackrel{def}{=} f(u,i) - f(u,j) \\ &= [UV^\top]_{u,i} - [UV^\top]_{u,j} \\ &= \sum_{k=1}^{r} U_{u,k} V_{i,k} - \sum_{k=1}^{r} U_{u,k} V_{j,k}. \end{aligned} \tag{7.5}$$

Substituting g in (7.4) we get

$$\mathcal{E}(U,V) = \sum_{u \in \mathcal{U}} \frac{1}{s_u} \sum_{(i,j) \in M_u} \mathbf{1}(\Delta M_{u,i,j}, g(u,i,j)), \tag{7.6}$$

where M_u is the list of items rated by the user u and $\Delta M_{u,i,j} = M_{u,i} - M_{u,j}$. Each pair (i,j) is considered once when $M_{u,i} > M_{u,j}$. Since $\Delta M_{u,i,j}$ does not depend on U and V, it is viewed as a constant in the optimization process. U and V are fitted by gradient-based updates:

$$U_{u,k} \leftarrow U_{u,k} - \nu \frac{\partial \mathcal{E}(U,V)}{\partial U_{u,k}}, \quad V_{i,k} \leftarrow V_{i,k} - \nu \frac{\partial \mathcal{E}(U,V)}{\partial V_{i,k}}$$

where ν is the learning rate.

We calculate below the derivative of $\mathcal{E}(U,V)$ with respect to the entries of U and V using the chain rule. To obtain the partial derivatives with respect to $U_{u,k}$, we sum

over all pairs of two items i and j rated by the user u. In this case, we can still assume without loss of generality that $M_{u,i} > M_{u,j}$. When calculating the partial derivatives with respect to $V_{i,k}$ we note that there are two possible cases: item i is preferred to j or vice versa. The two cases contribute to the two inner summations in (7.8).

$$\frac{\partial \mathcal{E}(U,V)}{\partial U_{u,k}} = \frac{1}{s_u} \sum_{(i,j)\in M_u} \frac{\partial \mathcal{L}(\Delta M_{u,i,j}, g)}{\partial g(u,i,j)} \frac{\partial g(u,i,j)}{\partial U_{u,k}} \tag{7.7}$$

$$\begin{aligned}\frac{\partial \mathcal{E}(U,V)}{\partial V_{i,k}} = \sum_{u\in\mathcal{U}} \frac{1}{s_u} \Bigg[& \sum_{j:M_{u,i}>M_{u,j}} \frac{\partial \mathcal{L}(\Delta M_{u,i,j}, g)}{\partial g(u,i,j)} \frac{\partial g(u,i,j)}{\partial V_{i,k}} \\ & + \sum_{j:M_{u,i}<M_{u,j}} \frac{\partial \mathcal{L}(\Delta M_{u,i,j}, g)}{\partial g(u,i,j)} \frac{\partial g(u,i,j)}{\partial V_{j,k}} \Bigg].\end{aligned} \tag{7.8}$$

The partial derivatives of g are given by

$$\frac{\partial g(u,i,j)}{\partial U_{u,k}} = V_{i,k} - V_{j,k} \tag{7.9}$$

$$\frac{\partial g(u,i,j)}{\partial V_{a,k}} = \begin{cases} U_{u,k} & \text{if } a = i \\ -U_{u,k} & \text{if } a = j \end{cases}. \tag{7.10}$$

The last step is the calculation of the partial derivatives of the loss functions in Table 7.1 with respect to g. For example in the case of the multiplicative log-loss, we have

$$\frac{\partial \mathcal{L}_{\text{Log[M]}}(\Delta M, g)}{\partial g} = \frac{\partial}{\partial g} \Delta M \log(1 + e^{\gamma - g}) = \frac{-\Delta M e^{\gamma - g}}{1 + e^{\gamma - g}}.$$

The partial derivatives need to be adjusted if regularization is used. For example, L_2 regularization ($\Omega(U) = \sum_{u,k} U_{u,k}^2$ and $\Omega(V) = \sum_{i,k} V_{i,k}^2$) adds a simple linear term to the partial derivatives.

7.2.3 Local Collaborative Ranking (LCR)

In LCR, we apply the ideas of local low-rank matrix approximation to the ranked loss minimization framework. Each local low-rank model covers a different subgroup of users and items, which are combined in prediction time using locally constant regression.

Extending LLORMA to ranking loss minimization is not straightforward as LLORMA uses two kernel functions (one for users $K(u_t, u)$ and the other for items $K(i_t, i)$)

as described in Section 6.2.1, while pair-wise losses are formed based on a single user and two items in each term $g(u, i, j)$. This leads to three kernel functions: $K(u_t, u), K(i_t, i)$, and $K(i_t, j)$.

To estimate the local models, we minimize the sum of ranked losses $\mathcal{L}$ (as in (7.6)) for all possible item pairs of the same user. Formally, we need to solve

$$\arg\min_{U_t, V_t} \sum_{u \in \mathcal{U}} \frac{1}{s_u} \sum_{(i,j) \in M_u} \mathcal{L}(\Delta M_{u,i,j}, g(u, i, j)) \tag{7.11}$$

where $t = 1, \ldots, q$ and $\Delta M_{u,i,j} = M_{u,i} - M_{u,j}$. The main difference from (7.6) is the definition of $g(u, i, j)$. Instead of the difference of predictions of a single global model $(f(u, i) - f(u, j))$, it is now a combination of local models:

$$\begin{aligned} g(u, i, j) &\equiv \hat{M}_{u,i} - \hat{M}_{u,j} \\ &= \sum_{t=1}^{q} \frac{K((u_t, i_t), (u, i))}{\sum_{s=1}^{q} K((u_s, i_s), (u, i))} [U_t V_t^\top]_{u,i} \\ &\quad - \sum_{t=1}^{q} \frac{K((u_t, i_t), (u, j))}{\sum_{s=1}^{q} K((u_s, i_s), (u, j))} [U_t V_t^\top]_{u,j}. \end{aligned} \tag{7.12}$$

After estimating the local models, we combine them at test time using locally constant regression (as in the case of LLORMA)

$$\hat{M}_{u,i} = \sum_{t=1}^{q} \frac{K((u_t, i_t), (u, i))}{\sum_{s=1}^{q} K((u_s, i_s), (u, i))} [U_t V_t^\top]_{u,i}, \tag{7.13}$$

where (u_t, i_t) is the anchor point of local model t. Note that this amounts to a convex combination of local models, weighted by the proximity of the local anchor point to the predicted user-item pair.

We can calculate the gradients similarly to (7.7) and (7.8), so long as the loss function $\mathcal{L}$ is differentiable. Specifically, we have

$$\frac{\partial g(u, i, j)}{\partial [U_t]_{u,k}} = [V_t]_{i,k} - [V_t]_{j,k} \tag{7.14}$$

$$\frac{\partial g(u, i, j)}{\partial [V_t]_{a,k}} = \begin{cases} [U_t]_{u,k} & \text{if } a = i \\ -[U_t]_{u,k} & \text{if } a = j \end{cases}. \tag{7.15}$$

7.2.3.1 Parallelism

Though we did not explicitly present the full description of the gradient calculation, it is evident that it is no longer possible to parallelize the gradient computation

as in the case of LLORMA. In LLORMA, each local model does not interact with other local models during training, so each local model can be learned in parallel (asynchronously). However, it is difficult to break the objective function (7.11) with (7.12) into independent optimization problems since the estimate of each local model influences the other local models.

Nevertheless, we can still take advantage of parallelism in a slightly different and more complex way. As long as all local models are at the same phase of the learning process, we only need the current global estimation $\hat{M}_{u,i}$ and not the estimation of each individual local model. Since the global estimate is the same for every local model so long as we establish a *synchronization* mechanism, we can compute combined predictions at the beginning of each iteration, freeze the values, and let each local model use these values during the gradient estimation. We can thus still update each local model in parallel, as long as the current estimation $\hat{M}_{u,i}$ is synchronized at each iteration.

To recap, the learning process consists of the following steps: we first initialize each model at random; then, we repeatedly alternate between the following two steps until a convergence criterion is met: (i) we calculate current estimations of each local model *(Estimation step)*, and (ii) we update each local model concurrently using the estimations *(Update step)*. While the estimation step requires synchronization, the update step can be done asynchronously. Since the update step is typically the most computationally expensive, the above computational scheme can significantly reduce run time.

7.2.3.2 Distance and Smoothing Kernel

In order to complete the derivation of the algorithm, we need to endow the space of recommendations with a metric between users and analogously a metric between items. Such metrics can be constructed using side information, for example, users' age, gender, and so on (either using standard distances or using metric learning techniques, for example [143] or [76, 77, 31]) However, many datasets (including most public datasets) do not include such data, in which case d can be based on the partially observed matrix M.

One possible approach is to construct the distance between two users based on the correlation of the two users (excluding unobserved ratings), and construct similarly the distance between two items. This approach does not perform well since the

partially observed matrix is very sparse, leading to poor estimates of correlation and distance. Instead, we factorize M using incomplete SVD (6.4) and obtain two latent representation matrices U and V of users and items respectively. We then proceed to construct the distance between two users based on the normalized inner-product of

Algorithm 7.1 The LCR Learning Algorithm

Input: $M \in \mathbb{R}^{m \times n}$
Parameters: number of local models q, local rank r, learning rate ν, regularization coefficient λ
Define: Ω as the set of observed entries in M
for all $t \in \{1, ..., q\}$ **do**
  Initialize $U_t \in \mathbb{R}^{m \times r}, V_t \in \mathbb{R}^{n \times r}$ randomly.
  Pick an observed pair (u_t, i_t) from M at random.
end for
while not-converged **do**
  // Estimation step (Synchronization)
  for all $(u, i) \in \Omega$ **do**
    $w_{u,i} \leftarrow \sum_{t=1}^{q} K(u_t, u)K(i_t, i)$
    $f_{u,i} = \sum_{t=1}^{q} \frac{K(u_t,u)K(i_t,i)}{w_{u,i}} \left[U_t V_t^\top\right]_{u,i}$
  end for
  for all $(u, i) \in \Omega$ and $(u, j) \in \Omega$ **do**
    $\ell_{u,i,j} = \frac{\partial E}{\partial g}\Big|_{g=f_{u,i}-f_{u,j}}$
  end for
  // Update step
  for all $t \in \{1, ..., q\}$ **in parallel do**
    $\forall u \in \{1, ..., m\} : [\Delta U]_u \leftarrow 0$
    $\forall i \in \{1, ..., n\} : [\Delta V]_i \leftarrow 0$
    for all $(u, i) \in \Omega$ and $(u, j) \in \Omega$ **do**
      if $M_{u,i} > M_{u,j}$ **then**
        $[\Delta U]_u \leftarrow [\Delta U]_u + (\frac{K(u_t,u)\cdot K(i_t,i)}{w_{u,i}} \cdot [V_t]_i$
        $\quad - \frac{K(u_t,u)\cdot K(i_t,j)}{w_{u,j}} \cdot [V_t]_j) \cdot \ell_{u,i,j}$
        $[\Delta V]_i \leftarrow [\Delta V]_i + \frac{K(u_t,u)\cdot K(i_t,i)}{w_{u,i}} \cdot [U_t]_u \cdot \ell_{u,i,j}$
        $[\Delta V]_j \leftarrow [\Delta V]_j - \frac{K(u_t,u)\cdot K(i_t,j)}{w_{u,j}} \cdot [U_t]_u \cdot \ell_{u,j,i}$
      end if
    end for
    $\forall u \in \{1, ..., m\} : \ [U_t]_u \leftarrow [U_t]_u - \nu(\frac{[\Delta U]_u}{|\mathcal{U}|\cdot s_u} + \lambda[U_t]_u)$
    $\forall i \in \{1, ..., n\} : \ [V_t]_i \leftarrow [V_t]_i - \nu(\frac{[\Delta V]_i}{|\mathcal{U}|\cdot s_u} + \lambda[V_t]_i)$
  end for
end while
output: $U_t V_t^\top, \ t = 1, \ldots, q$

Algorithm 7.2 The LCR Prediction

1: **Input:** learned local models $U_t V_t^\top (t = 1, \ldots, q)$, test point (u^*, i^*)
2: **return** $\hat{M}_{u,i} = \sum_{t=1}^{q} \frac{K(u_t,u^*)K(i_t,i^*)}{\sum_{s=1}^{q} K(u_s,u^*)K(i_s,i^*)} \left[U_t V_t^\top\right]_{u^*,i^*}$

row i and row j of the matrix U,

$$\arccos\left(\frac{\langle u_i, u_j\rangle}{\|u_i\| \cdot \|u_j\|}\right),$$

where we denote by u_i and u_j the i and j rows of U. The analogous expression with V replacing U may be used to construct the distance between two items.

We used a product kernel $K((u_t, i_t), (u, i)) = K(u_t, u)K(i_t, i)$, where each component kernel is an Epanechnikov kernel ($K(s_1, s_2) = 3/4(1 - d(s_1, s_2)^2)\mathbf{1}[d(s_1, s_2) < h]$) with distances computed as described above. We also tried uniform and triangular kernel, but the performance was worse than Epanechnikov kernel, in agreement with the theory of kernel smoothing [137].

7.2.3.3 The Algorithm

The pseudo-code of the resulting learning algorithm is provided in Algorithm 7.1. We use an L_2 regularization scheme, but other kinds of regularization can be used instead. Prediction of an unseen test example (u^*, i^*) is described in Algorithm 7.2.

In line 2–5, we randomly initialize q local models and their anchor points. In the main loop, lines 6 through 30, we repeat synchronizing current estimation and updating each local model in parallel. In **synchronization step** (line 7–14), we prepare intermediate values which are independent of local models. This includes normalization constant for each weight ($w_{u,i}$), current estimation on each observed points ($f_{u,i}$), and the gradient $\frac{\partial E}{\partial g}$ for all training pairs (u, i, j). These are referred by each local model in **gradient update step**, in line 15–29. Using all training examples, we update each local model by using (7.7) and (7.8). Instead of updating with each training example, we accumulate gradients for each user, and update at once. Note that Δ_u and Δ_i are vectors of size r, the rank of latent matrices U and V. Each local model can proceed this step in parallel, as they share only current estimation and gradient information we got in synchronization step. When the estimation converges to ground truth, we stop iterations and output q local models.

7.3 Experiments

We conducted experiments to evaluate the performance of the proposed LCR approach on real world datasets. In the first set of experiments, we varied the LCR hyper-parameters in order to better understand the dependency of LCR on them. In the second set of experiments, we compared the performance of LCR method with other well-known ranking methods for recommendation systems.

7.3.1 Experimental Setting

7.3.1.1 Data Split

We evaluate the performance of the learning algorithm by separating the data into a training part, used to train the model, and a test part, used to evaluate it. In order to evaluate the sensitivity of the algorithm to the size of the training set, one can vary either the number of available training ratings per user, or the proportion of training ratings per user, keeping the rest of the ratings for the test set. Since users have a variable number of ratings [130, 101], the number of training and test ratings per user can vary significantly depending on this choice.

In the literature, CofiRank [140] introduced an experimental setting which fixes the number of training ratings per user. In this setting, we randomly choose a constant number N of training ratings per user, and all other ratings are kept in the test set. Users without enough ratings are dropped from the test set. For example, for a user with 100 ratings and $N = 10$, we use 10 ratings for training and 90 for test. Another user with 8 ratings is dropped as this user does not satisfy the minimum requirement. This evaluation procedure has low variance since users in the test set typically have a large number of ratings. The evaluation is biased towards frequent raters, as we drop users with few ratings. This evaluation method has recently became standard practice [140, 3, 135, 34] and thus we adopt it in our experiments where we compare the performance of LCR with others methods (Section 7.3.3).

The alternative evaluation method keeps a fixed proportion of ratings for each user in the training set and moves the rest of the observations to the test set (regardless of the number of available ratings for each user). For example, if the ratio is 0.5, a user with 100 ratings will have 50 of its ratings kept for training and the remaining 50 used for testing. A user with 8 ratings is not dropped, but trained with 4 ratings and tested with the remaining 4 ratings. This scheme is less biased since it contains

users with a few ratings as well as users with many ratings, but it may have higher variance due to the inclusion of users with a small number of ratings.

We experiment with both approaches. In the first set of experiments (Section 7.3.2) where we investigate the dependency of LCR on its hyper-parameters, we used the fixed ratio setting. In the second set of experiments (Section 7.3.3), where we investigate the performance of LCR relative to other recommendation systems, we report results with a fixed number of ratings (as in CofiRank). This choice was made to facilitate a comparison with perviously published results using the fixed number of ratings setting.

7.3.1.2 Evaluation Metrics

We use three evaluation measures, which are widely used to evaluate ranking approaches. *Zero-one Error* is the average ratio of correctly ordered test pairs, averaged over all users and is based on the zero-one ranking loss (7.3) (giving constant loss when the relative order of preferences contradicts the ground truth):

$$\mathcal{E}_{0/1} = \sum_{u \in \mathcal{U}} \sum_{i \in T_u} \sum_{j \in M_u \cup T_u \setminus \{i\}} \frac{1}{Z_u} \mathbf{1}[\Delta M_{u,i,j} \cdot g(u,i,j) < 0],$$

where $Z_u = |U| \cdot |T_u| \cdot (|M_u \cup T_u| - 1)$, and M_u and T_u are the set of available ratings for user u in the train and test sets respectively.

Note that we compare the relative order of an item in the test set with all other known ratings, including those in training set. In other words, pairs of items belonging to the train and test sets are used as well as pairs of items that belong only to the test set. Pairs of items only belonging to the train set are not used for testing.

Average Precision is defined as the area under the precision-recall curve, and is given by

$$AvgP = \frac{1}{|U|} \sum_{u \in U} \int_0^1 P(r) dr,$$

where the integration variable r ranges over all possible recall levels and $P(r)$ is the precision at recall r. In practice, the integral is replaced with a finite sum over every position in the ranked sequence of recommendations.

DCG@k takes into account the order of recommended items in the list by discounting the importance and is formally given by

$$DCG@k = \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{\log_2(i+1)},$$

where i ranges over positions in the recommendation list and rel_i indicates how much the ith item is relevant to the user (we use the observed rating score for this quantity). *NDCG* is the ratio of DCG to the maximum possible DCG for that user. This maximum occurs when the recommended items are presented in decreasing order of user preference. NDCG is perhaps the most popular evaluation metric, reflecting the importance of retrieving good items at the top of the ranking. In our experiments we use NDCG with $k = 10$.

7.3.1.3 Datasets

We used three real-world datasets. MovieLens 100K[1] contains 943 users and 1,682 items with 6.3% (user, item) pairs rated (or 6.3% dense). This small dataset is ideal for quickly benchmarking several models, so we used it to explore the importance of various hyper-parameters. We also used the bigger EachMovie dataset, with 61,265 users, 1,648 items, and a density 2.8% ratings. These two datasets are widely used in the literature. In addition, we used the Yelp[2] dataset (43,873 users and 11,537 items, with 0.04% ratings), which was released for the Yelp business rating prediction challenge at ACM RecSys 2013. This dataset is the most recent one, reflecting a recent trend of extreme sparsity.

7.3.1.4 Model Parameters

We varied the number of local models in the range {10, 20, 30, 40, 50}, the rank or latent space dimension in the set {5, 10, 15, 20}, and the loss functions {$\mathcal{L}_{\text{Log[M]}}$, $\mathcal{L}_{\text{Log[A]}}$, $\mathcal{L}_{\text{Exp[M]}}$, $\mathcal{L}_{\text{Hinge[A]}}$} in Table 7.1 with $\gamma = 0$. We used Epanechnikov kernel as a smoothing kernel as it achieves the lowest integrated squared error [137] (with kernel bandwidth of 0.8). We also used L_2-regularization, where the regularization coefficient was determined by cross-validation. Cross validation was also used to determine early stopping. In the cross validation procedure, we set aside 20% of the training data for validation purpose. Note that different evaluation metrics typically lead to different stopping times. In our experiments, we stopped either when the improvement in training error got smaller than some threshold (0.001) or when the algorithm reached 200 iterations. This is to show the speed of convergence as well as the degree of overfitting. For real use, we stop for each particular error measure

[1] http://www.grouplens.org/

[2] http://recsys.acm.org/recsys13/recsys-2013-challenge-workshop/

according to the separate validation set.

7.3.2 Trend Analysis

We first show how our LCR model behaves with different combinations of parameters. Specifically, we vary the rank of each local model in $\{5, 10, 15, 20\}$, and the number of local models in $\{10, 20, 30, 40, 50\}$. We used the MovieLens 100K dataset with fixed ratio split (ratio = 0.5) for this part of the experiments.

7.3.2.1 Effect of the Rank

Figure 7.1 compares the performance of LCR when varying the rank of the local models. The zero-one error (left) decreases monotonically as expected, since we minimize a smooth surrogate $\mathcal{L}_{\text{Log[M]}}$ loss. When the rank gets higher, the final performance gets better, but it takes more iterations until convergence. In other words, higher ranks achieve better performance, but take longer to converge. Average precision and NDCG@10 show a different trend. Unlike the zero-one error, they overfit at some point, implying that it is important to use with these evaluation metrics cross validation for determining early stopping. With higher ranks, however, overfitting seems less of a problem; when the rank is sufficiently large, it achieves its best performance at the end. Note that similar results are obtained for other loss functions and number of local models.

7.3.2.2 Effect of the Number of Local Models

Intuitively, one can imagine that increasing the number of local models in LCR would yield a better performance, because it would reduce the variance of prediction. Figure 7.2 compares the performance of LCR when varying the number of local models. As in Figure 7.1, the zero-one error monotonically decreases and converges to an improved score as the number of local models increases. Average precision and NDCG@10 tend to overfit, but the tendency to overfit diminishes as the number of local models increases. In contrast to Figure 7.1, however, more local models tend to result in almost always better performance than less local models. This shows that increasing the number of local models is relatively resistant to overfitting, unlike increasing the decomposition rank (compare the right two panels of Figure 7.1 and 7.2).

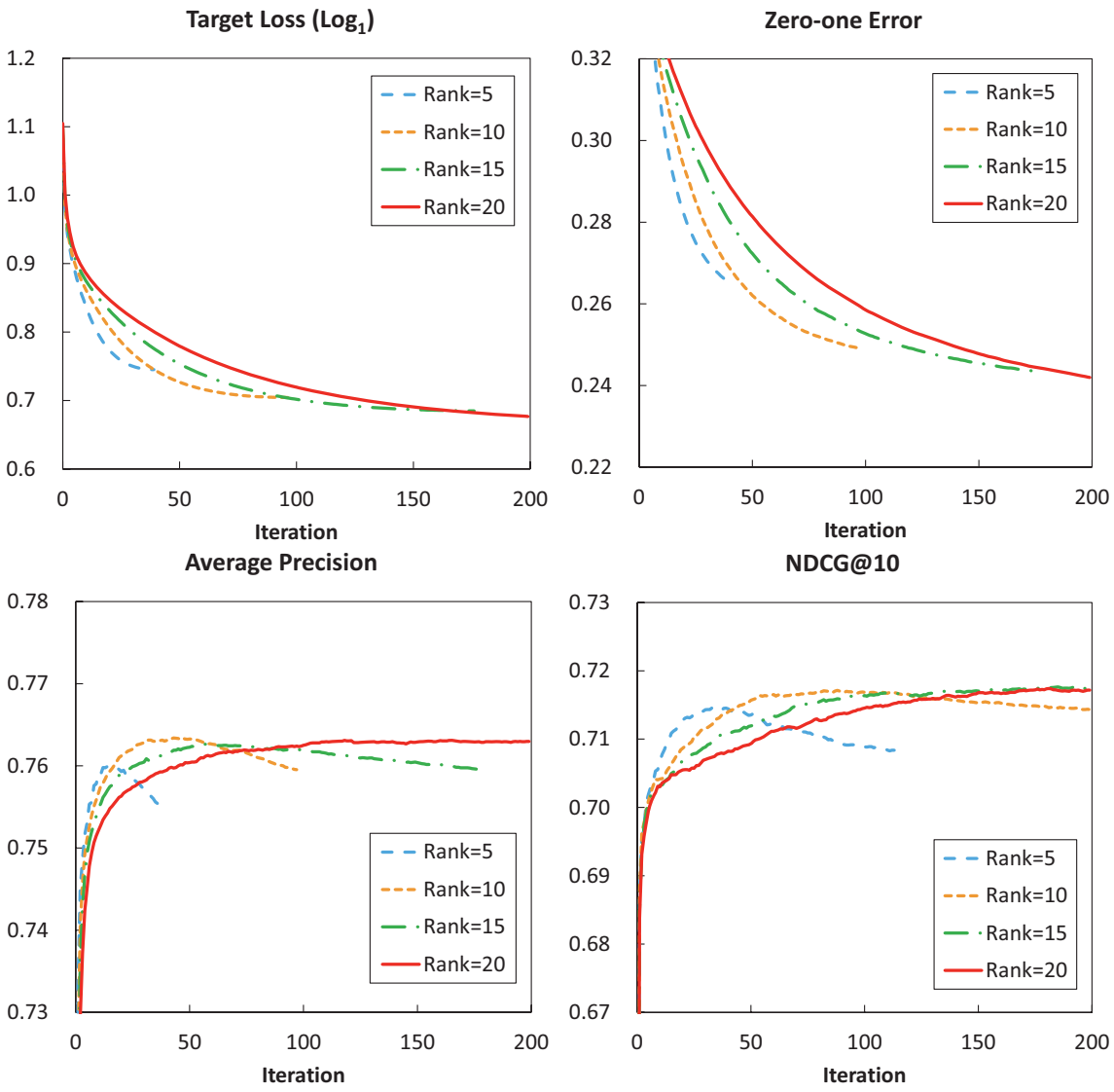

Figure 7.1: Effect of the model rank on the performance of LCR, measured by the target error (top-left), zero-one error (top-right), average precision (bottom-left), and NDCG@10 (bottom-right). Ranks vary in $\{5, 10, 15, 20\}$, while the number of local models is fixed to 10. The optimized loss in the training procedure is $\mathcal{L}_{\text{Log[M]}}$ in Table 7.1 with $\gamma = 0$.

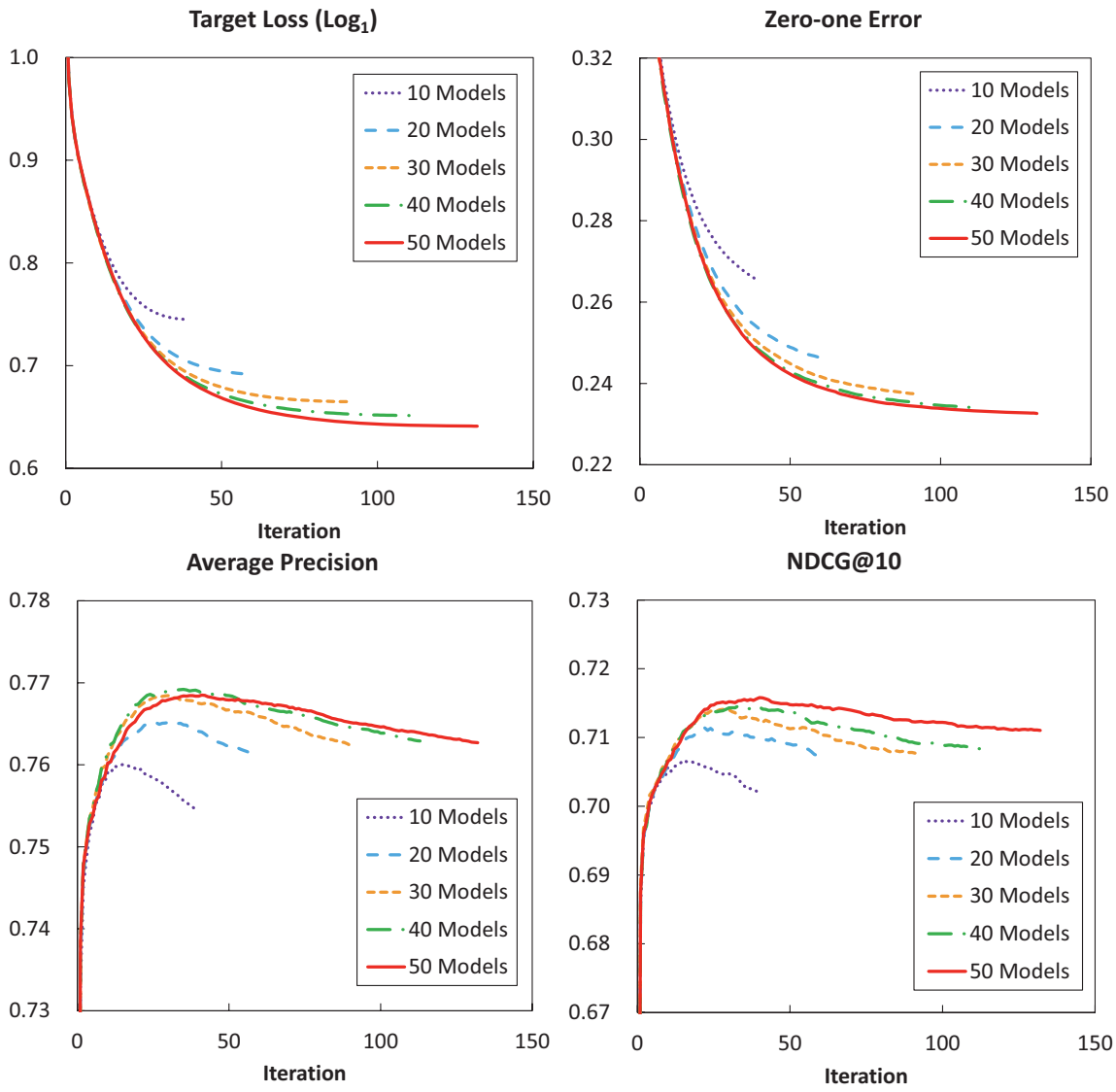

Figure 7.2: Effect of the number of local models on the performance of LCR, measured by the target error (top-left), zero-one error (top-right), average precision (bottom-left), and NDCG@10 (bottom-right). The number of local models vary in $\{10, 20, 30, 40, 50\}$, while the rank of each local model is fixed to 5. The optimized loss in the training procedure is $\mathcal{L}_{\text{Log[M]}}$ in Table 7.1 with $\gamma = 0$.

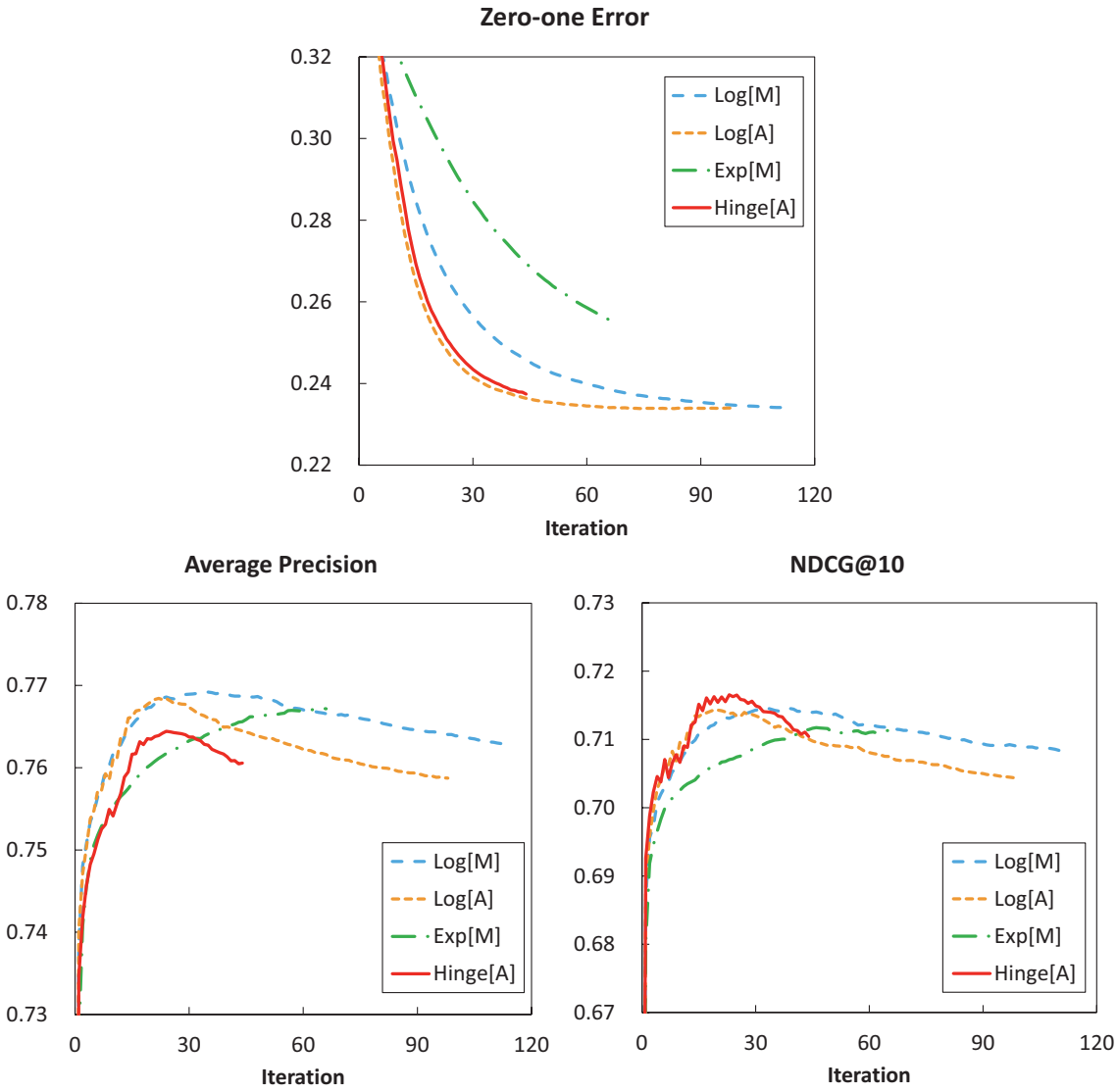

Figure 7.3: Effect of the optimized loss on the performance of LCR, measured in zero-one error (top), average precision (bottom-left), and NDCG@10 (bottom-right). We tried $\{\mathcal{L}_{\text{Log[M]}}, \mathcal{L}_{\text{Log[A]}}, \mathcal{L}_{\text{Exp[M]}}, \mathcal{L}_{\text{Hinge[A]}}\}$ in Table 7.1 with $\gamma = 0$, while the rank of each local model and the number of local models is fixed to 5 and 40, respectively.

7.3.2.3 *Effect of the Optimized Loss Function*

We also compared how LCR performs when changing the surrogate loss function that is used in the optimization procedure during the training stage. Figure 7.3 compares the performance of LCR optimized with different surrogate losses. Training on log-losses, in general, shows outstanding performances, as evaluated by zero-one error. Note that $\mathcal{L}_{\text{Log[A]}}$ achieves its best performance earlier than $\mathcal{L}_{\text{Log[M]}}$, but tends to overfit more. On the other hand, training by minimizing $\mathcal{L}_{\text{Exp[M]}}$ overfits much less as evaluated by both average precision and NDCG. Training by minimizing $\mathcal{L}_{\text{Hinge[A]}}$ was the quickest to converge among the four and performed especially well when evaluated in terms of the zero-one error. A drawback of $\mathcal{L}_{\text{Hinge[A]}}$ is its significant overfitting when evaluated by average precision and NDCG.

Figure 7.4 illustrates the behavior of the different loss functions. $\mathcal{L}_{\text{Log[M]}}$ loss (left-most) assigns almost no penalty when the estimation is correct, while relatively high penalty for wrong prediction. This means a model minimizing $\mathcal{L}_{\text{Log[M]}}$ loss focuses on pairs which are currently incorrectly estimated, while almost ignoring other pairs. $\mathcal{L}_{\text{Exp[M]}}$ loss (third) is an even more extreme loss function. These two loss functions tend to overfit less, conjecturally because they do not fit to fine details. In other words, they do not touch training pairs as long as they are correctly ordered with current model. This would be advantageous to avoid overfitting to fine details of the training examples. On the other hand, $\mathcal{L}_{\text{Log[A]}}$ and $\mathcal{L}_{\text{Hinge[A]}}$ losses assign non-zero penalty for correctly ordered pairs if the degree does not match precisely. As a result, they Thus, these models would estimate more pairs correctly, leading to lower zero-one error. However, they are subject to overfitting as they fit to fine details during this process.

We also tried $\mathcal{L}_{\text{Exp[A]}}$ and $\mathcal{L}_{\text{Hinge[M]}}$ in Table 7.1, but we excluded them for the following reasons. $\mathcal{L}_{\text{Exp[A]}}$ loss was too sensitive to hyper-parameters such as learning rate, making it difficult to use in practical settings. $\mathcal{L}_{\text{Exp[A]}}$ is more extreme than $\mathcal{L}_{\text{Exp[M]}}$, and thus is highly influenced by a very small portion of training examples. Models trained on $\mathcal{L}_{\text{Hinge[M]}}$ achieve performance similar to but slightly worse than models trained on $\mathcal{L}_{\text{Hinge[A]}}$.

7.3.2.4 *Overall Comparison*

As a summary, Figure 7.5 compares the final performance when varying the rank and the number of local models. Note that higher rank generally leads to better

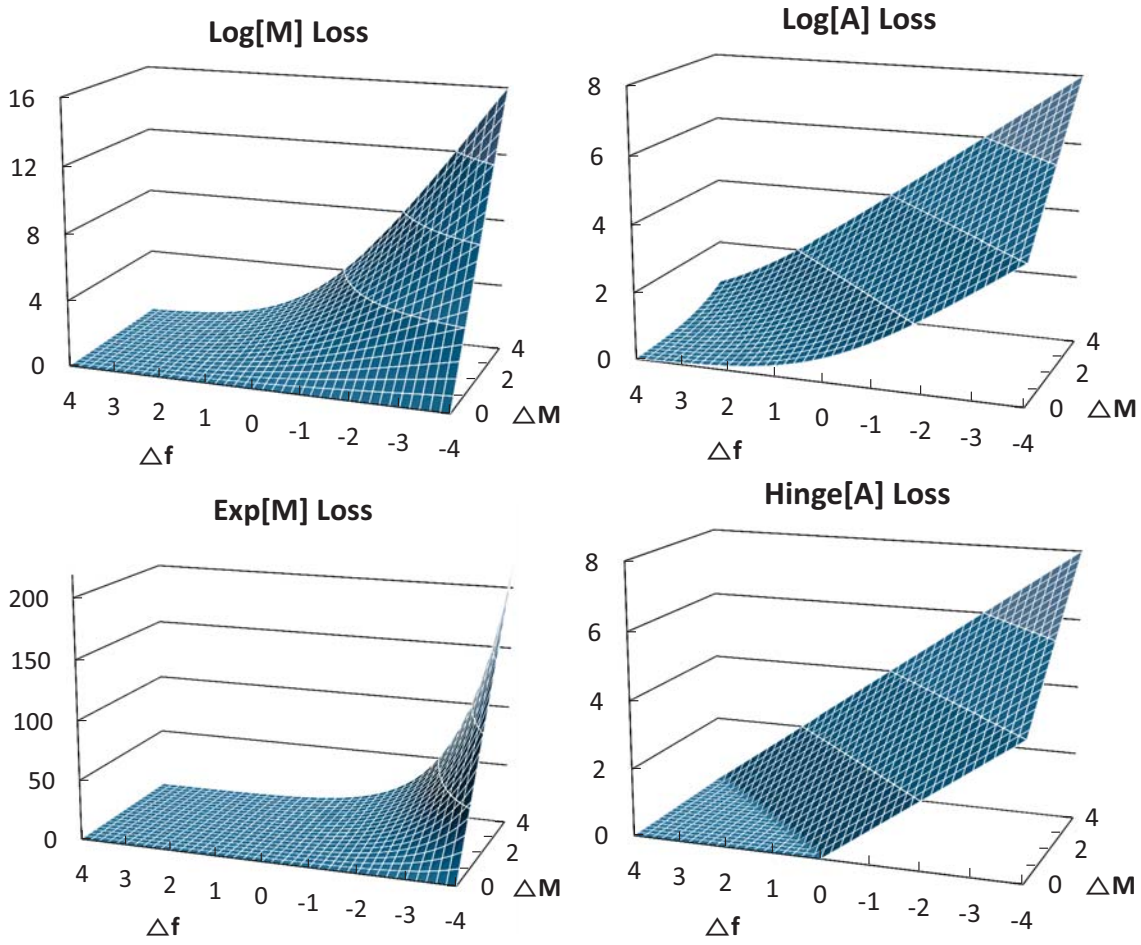

Figure 7.4: Shape of loss functions used in our experiments. ΔM is assumed to be always positive by arranging two items such that $M_{u,i} > M_{u,j}$. Δf may be either positive (left-half of each plot) or negative (right-half of each plot); positive Δf means the estimation is concordant with the real preference, while negative Δf implies the preference order is reversed in our estimation. Thus, as we expect, each loss function assigns higher penalty for more significant mistakes (in this figure, larger ΔM for negative Δf).

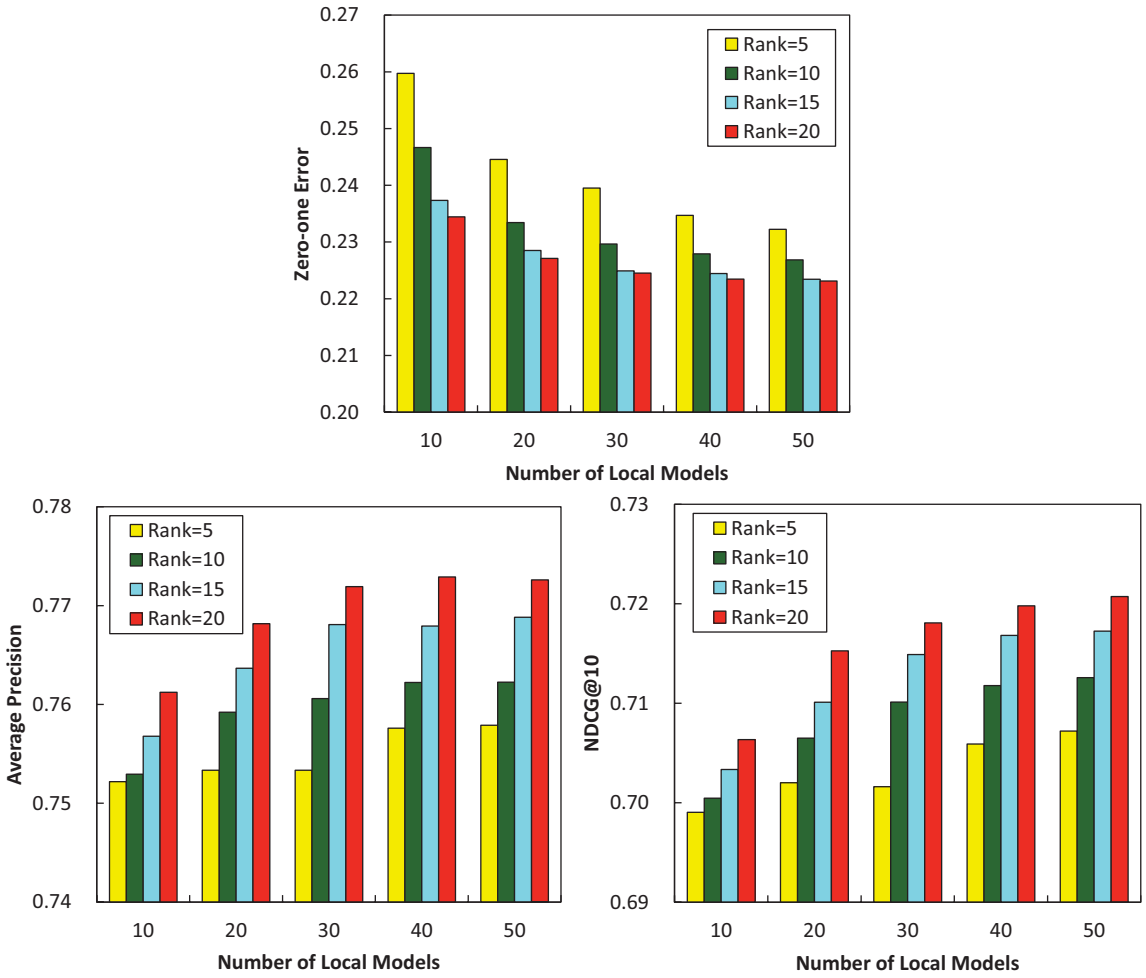

Figure 7.5: Performance trend in zero-one error (Top; the lower the better), average precision (Bottom-left; the higher the better), and NDCG@10 (Bottom-right; the higher the better), with various ranks and number of local models. The presented results are with $\mathcal{L}_{\text{Log[A]}}$. Models with other losses show similar trend.

performance with any evaluation metric, and more local models also leads to better performance with all evaluation metrics. Increasing the rank and number of local models, however, also increases computation time. (Note however that the local models may be trained in parallel, which facilitates deployment on multi-threaded computers and on clusters).

7.3.3 Comparison with Other Methods

7.3.3.1 Experimental Setup

In this section, we compare the performance of LCR with other models. Specifically, we compare with standard matrix factorization models (Regularized SVD [11],

Table 7.2: Test performance in terms of average precision and NDCG@10. Higher values mean better performance. Bold faces mean that the method performs statistically significantly better in the setting, at the level of 95% confidence level.

Metric		Average Precision			NDCG@10		
Data \ Method		$N = 10$	$N = 20$	$N = 50$	$N = 10$	$N = 20$	$N = 50$
MovieLens	CofiRank	0.6632 ± 0.0020	0.6825 ± 0.0034	0.6915 ± 0.0021	0.6502 ± 0.0021	0.6629 ± 0.0040	0.6912 ± 0.0009
	RegSVD	0.7204 ± 0.0021	0.7321 ± 0.0047	0.7501 ± 0.0045	0.6425 ± 0.0073	0.6510 ± 0.0066	0.6778 ± 0.0072
	NMF	0.7107 ± 0.0017	0.7234 ± 0.0041	0.7328 ± 0.0052	0.6285 ± 0.0069	0.6336 ± 0.0097	0.6471 ± 0.0064
	BPMF	0.6376 ± 0.0115	0.6517 ± 0.6517	0.6915 ± 0.0064	0.5636 ± 0.0119	0.5513 ± 0.0184	0.5956 ± 0.0035
	LLORMA	0.7341 ± 0.0016	0.7424 ± 0.0015	0.7490 ± 0.0053	0.6712 ± 0.0027	0.6682 ± 0.0061	0.6746 ± 0.0063
	GCR	0.7209 ± 0.0011	0.7356 ± 0.0010	0.7534 ± 0.0021	0.6990 ± 0.0011	0.6908 ± 0.0050	0.6932 ± 0.0043
	LCR	$\mathbf{0.7406 \pm 0.0019}$	$\mathbf{0.7503 \pm 0.0022}$	$\mathbf{0.7626 \pm 0.0017}$	$\mathbf{0.7152 \pm 0.0056}$	$\mathbf{0.7022 \pm 0.0049}$	$\mathbf{0.7039 \pm 0.0038}$
EachMovie	CofiRank	$\mathbf{0.7491 \pm 0.0056}$	$\mathbf{0.7476 \pm 0.0054}$	0.7231 ± 0.0033	0.6635 ± 0.0058	0.6985 ± 0.0031	$\mathbf{0.7158 \pm 0.0038}$
	RegSVD	0.6979 ± 0.0011	0.7159 ± 0.0009	$\mathbf{0.7307 \pm 0.0020}$	0.6632 ± 0.0009	0.6784 ± 0.0015	0.6981 ± 0.0012
	NMF	0.6904 ± 0.0011	0.6895 ± 0.0002	0.6769 ± 0.0015	0.6555 ± 0.0014	0.6542 ± 0.0007	0.6385 ± 0.0022
	BPMF	0.6678 ± 0.0143	0.7024 ± 0.0054	0.7172 ± 0.0012	0.6261 ± 0.0170	0.6615 ± 0.0053	0.6742 ± 0.0007
	LLORMA	0.7151 ± 0.0040	0.7116 ± 0.0032	0.7258 ± 0.0020	0.6820 ± 0.0033	0.6778 ± 0.0035	0.6863 ± 0.0028
	GCR	0.7088 ± 0.0006	0.7106 ± 0.0010	0.6958 ± 0.0006	0.6998 ± 0.0008	0.6979 ± 0.0010	0.6779 ± 0.0012
	LCR	0.7307 ± 0.0010	0.7372 ± 0.0018	$\mathbf{0.7330 \pm 0.0045}$	$\mathbf{0.7166 \pm 0.0017}$	$\mathbf{0.7201 \pm 0.0010}$	0.6988 ± 0.0046
Yelp	CofiRank	0.7246 ± 0.0018	0.7273 ± 0.0020	0.7235 ± 0.0035	0.6997 ± 0.0026	0.6842 ± 0.0029	0.6680 ± 0.0028
	RegSVD	0.7799 ± 0.0013	0.7829 ± 0.0010	$\mathbf{0.7774 \pm 0.0034}$	0.7021 ± 0.0019	0.6953 ± 0.0021	0.6823 ± 0.0035
	NMF	0.7757 ± 0.0026	0.7799 ± 0.0012	0.7750 ± 0.0031	0.6926 ± 0.0032	0.6871 ± 0.0037	0.6705 ± 0.0051
	BPMF	0.7063 ± 0.0041	0.7035 ± 0.0020	0.7035 ± 0.0061	0.6191 ± 0.0022	0.5983 ± 0.0032	0.5838 ± 0.0029
	LLORMA	0.7844 ± 0.0011	$\mathbf{0.7882 \pm 0.0012}$	0.7822 ± 0.0024	0.7065 ± 0.0023	0.7019 ± 0.0023	0.6842 ± 0.0039
	GCR	0.7754 ± 0.0012	0.7797 ± 0.0034	0.7428 ± 0.0047	0.7465 ± 0.0023	0.7332 ± 0.0026	0.6582 ± 0.0072
	LCR	$\mathbf{0.7903 \pm 0.0021}$	$\mathbf{0.7901 \pm 0.0040}$	$\mathbf{0.7791 \pm 0.0043}$	$\mathbf{0.7575 \pm 0.0024}$	$\mathbf{0.7425 \pm 0.0018}$	$\mathbf{0.7212 \pm 0.0051}$

NMF [79], Bayesian PMF [115]), with least squares local matrix approximation (LLORMA, Chapter 6), and with the ranked loss minimization methods CofiRank [140] and GCR (matrix approximation minimizing ranked loss in Section 7.2.2).

Among the different baselines above, LLORMA is notable since it is a local matrix approximation approach (though based on least squares minimization), and GCR is notable since it is a global matrix approximation based on ranked loss minimization. CofiRank is notable as it is considered a very strong baseline in recent literature. We implemented all of the methods above within the PREA toolkit [81], with the exception of CofiRank that made its code publicly available.

In order to compare with previous published results, we adopt here the CofiRank weak generalization setup (see Section 6 of [140]), predicting the rank of unrated items for users known at training time. For each user, N randomly chosen items, with $N = 10, 20, 50$, are used for training. Another 10 items are set aside for validation set, and all other items are used for testing. Users with less than $N + 20$ ratings are dropped to guarantee at least 10 items can be used for testing. This is a slight modification of the original CofiRank experimental setup, where no extra validation set was used. This modification was also proposed in [135, 34].

7.3.3.2 Parameter Setting

For LCR, we consider all options including rank, number of local models, and optimized loss function as hyper-parameters. That is, we select the best model for each dataset and for each evaluation metric on the separate validation set. For CofiRank, we use the same parameter values (100 dimensions and $\lambda = 10$) provided in the original paper [140], and default values provided in the source code for unstated parameters (such as the maximum number of iterations and BMRM parameters). For LLORMA, the rank of local models was chosen among $\{5, 10, 15, 20\}$, and the number of local models among $\{10, 20, 30, 40, 50\}$. We used the Epanechnikov kernel with width 0.8. All other parameters were set to default values in their implementation. Ranks of other matrix factorization methods were set to $\{10, 20, 30, 40, 50\}$ (SVD), $\{20, 40, 60, 80, 100\}$ (NMF), and $\{2, 3, 4, 5\}$ (BPMF). Regularization coefficients and learning rates were chosen by cross validation.

7.3.3.3 Result and Discussion

Table 7.2 compares LCR with competing approaches in terms of average precision and NDCG@10.

In terms of average precision, LCR outperforms all other approaches on MovieLens and Yelp datasets. With EachMovie, it lags behind CofiRank in some cases ($N = 10, 20$), but is better than others. In terms of NDCG, LCR performs better for all datasets as well, with the exception of EachMovie with $N = 50$. We conclude that LCR generally outperforms other approaches in terms of ranking metrics, such as average precision and NDCG, which are motivated by the ranking scenario of modern recommendation systems.

7.4 Summary

We presented in this chapter a novel collaborative ranking method based on the assumption that the rating matrix is globally high-rank but locally low-rank. The LCR approach generalizes LLORMA in Chapter 6 from least squares to ranked loss minimization, and it outperforms LLORMA and state-of-the-art ranked loss minimization methods for collaborative filtering.

It was previously observed that the local low-rank assumption applies to recommendation systems in the context of least squares rating prediction. In this chapter, we verified that the same conclusion holds when performance is evaluated using a variety of ranked loss functions. Besides providing improved ranking accuracy, our model also scales up to large datasets due to our proposed parallel training scheme. Another computational advantage stems from the fact that each local model may be restricted to a rank that is lower than the rank selected for a global model. Indeed, replacing a single rank l matrix factorization with multiple rank l' (local) matrix factorizations with $l' < l$ may lead to a win-win situation in terms of both computational efficiency and ranking accuracy.

Chapter VIII

LITERATURE SURVEY

8.1 Surveys and Experimental Studies

Recommendation systems emerged in 1990s in order to provide personalized services in E-commerce. They have been popularized by the very successful Netflix competition[1] held between 2006 and 2009, and since then many algorithms were proposed for predicting ratings and constructing recommendations.

Several well-written surveys on recommendation systems and collaborative filtering are available. Adomavicius and Tuzhilin [1] categorized CF algorithms available as of 2006 into content-based, collaborative, and hybrid and summarized possible extensions. Su and Khoshgoftaar [126] concentrated more on CF methods, including memory-based, model-based, and hybrid methods. This survey contains most state-of-the-art algorithms available as of 2009, including Netflix prize competitors. A recent textbook on recommender systems introduces traditional techniques and explores additional issues like privacy concerns [57].

There are a couple of experimental studies available. The first study by Breese et al. [12] compared two popular memory-based methods (Pearson correlation and vector similarity) and two classical model-based methods (clustering and Bayesian network) on three different dataset. A more recent experimental comparison of CF algorithms [53] compares user-based CF, item-based CF, SVD, and several other model-based methods, focusing on e-commerce applications. It considers precision, recall, F1-measure and rank score as evaluation measures, with comments about the computational complexity issue. This however ignores some standard evaluation measures such as MAE or RMSE.

8.2 Collaborative Filtering

Many memory-based CF methods predict the rating of items based on the similarity of the test user and the training users [109, 12, 48]. Similarity measures include

[1] http://www.netflixprize.com/

Pearson correlation [109] and Vector cosine similarity [12, 48]. Other memory-based CF methods includes item-based CF [117] and a non-parametric probabilistic model based on ranking preference similarities [128].

Model-based CF includes user and item clustering [12, 134, 146], Bayesian networks [12], dependence network [46] and probabilistic latent variable models [104, 86, 147]. Slope-one [82] achieved fast and reasonably accurate prediction. Matrix factorization for recommender systems have been the focus of voluminous amount of research especially since the Netflix Prize competition. It is clearly impossible to review all of the existing approaches. We review here a few of the notable approaches. [11] initially proposed applying SVD for collaborative filtering problems. [115] presented probabilistic matrix factorization (PMF) and later [114] extended matrix factorization to fully Bayesian approach. [75] proposed a non-linear version of PMF. [108] proposed a maximum-margin approach. Other examples include [71, 79, 74].

Further algorithmic improvements in matrix completion were demonstrated in [132, 65]. The work that is perhaps the most similar in to this thesis is Divide-and-Conquer Matrix Factorization (DFC) [85]. DFC also divides the completion problems into a set of smaller matrix factorization problems. Our approach in Chapter 6 differs DFC in that we use a metric structure on $[n_1] \times [n_2]$ and use overlapping partitions. Another matrix approximation by sub-division based on clustering was reported in [90] for the task of seeking user and item communities.

Lastly, there is a line of research theoretically analyzing the performance of low-rank matrix completion. [18] derived a bound on the performance of low-rank matrix completion under noise-free assumption, and [17] adapted the analysis of [18] to noisy settings. Some more remote related results were presented in [120, 36, 37].

8.3 Ensemble Models and Local Approaches

In addition to monolithic matrix factorization scheme, several ensemble methods have also been proposed for collaborative filtering. DeCoste [29] suggested ensembles of maximum margin matrix factorization (MMMF). A mixture of experts model is proposed in [125] to linearly combine the prediction results of more than two models. The Netflix Prize winner [8, 70] used combination of memory-based and matrix factorization methods. The Netflix Prize runner-up [123] devised Feature-Weighted Least Square (FWLS) solver, using a linear ensemble of learners with dynamic weights. [73] and [85] applied ensembles to Nystrom method and DFC, respectively.

Feature-weighted linear stacking [123] is the ensemble method most closely related to our approach in Chapter 5. The primary difference is the manual selection of features in [123] as opposed to automatic induction of local features in our paper that leads to a significant improvement in prediction quality. Model combination based on locality has been proposed in other machine learning topics, such as classification [142, 89] or sensitivity estimation [10].

Other local learning paradigms were suggested in the context dimensionality such as local principal component analysis [64] and local linear embedding (LLE) [112]. A relatively recent paper on matrix completion [139] applies low-rank factorization to clusters of points.

8.4 Collaborative Ranking

In the early 2000s, machine learning techniques were developed specifically for ranking problems (see for instance [59]). While early approaches focused on minimizing the pairwise zero-one loss, more recent approaches were developed to learn objectives that better follow metrics like NDCG or average precision [140, 27], which are more in line with real applications like product recommendation. Recent examples include [3, 135], which proposed collaborative ranking algorithms to approximately optimize NDCG directly; EigenRank [83], which optimizes a preference function using Kendall's Tau rank correlation. Additional examples are [44, 100, 138, 34, 122, 21, 131, 107].

The idea of collaborative ranking was developed and much more widely-used for web search and label ranking. Minimizing pair-wise ranking loss was proposed in Ranking SVM [47], RankBoost [38], RankNet [15], and FRank [133]. Wsabie [141] used weighted approximate-rank pairwise loss in label ranking for image annotation problem. Direct list-wise optimization over NDCG or average precision was also intensely investigated, with a few examples being LambdaRank [32], SVM^{RANK} [148], AdaRank [144], and [20, 145]. Other examples of collaborative ranking in web search are [39, 24, 23, 94].

Chapter IX

CONCLUSIONS AND FUTURE DIRECTIONS

Recall that we posed four research questions in Chapter 1. In this chapter, we summarize our answers to those questions with the proposed local approaches for collaborative filtering-based recommendation systems. We then conclude with contributions and remarks on future directions.

9.1 Conclusions

We posed four research questions in introduction, and we provided answers throughout this thesis. We conclude with summary of those answers in this section.

Q1. What are the existing CF models for rating prediction, and what is the best?

A1. With the toolkit PREA (Chapter 3) and experimental study (Chapter 4) based on it, we reconfirmed that the matrix factorization and its variations are generally performing the best for the rating prediction problem. More importantly, however, we discovered that there is no single CF method outperforming all other CF models to every user and for every item. Instead, we observed that some model can outperform another model for some particular set of users and items, but it may perform poorly for some other subset of users and items.

Q2. If one is not always the best, how can we combine multiple CF models?

A2. Starting from the observation that the relative performance of different candidate recommendation systems depends on the target user and item, we developed an ensemble model of recommendation systems with non-constant weights that emphasize different candidates based on their relative strengths in the feature space. In contrast to the previous work (FWLS [123]) that focused on manual construction of features, we automated this process by developing a feature induction algorithm that works in conjunction with stagewise least-squares. Specifically, we formulated a family of feature functions, based on the discrete analog of triangular kernel smoothing. This family captures a wide variety of local information and is thus able to model the

relative strengths of the different CF methods.

The combination with induced features outperformed any of the base candidates as well as other combination methods in literature. Our method scales up nicely to large data which is often present in real world recommendation systems. As our candidates included many of the recently proposed state-of-the-art recommendation systems, our conclusions are significant for the engineering community as well as recommendation system scientists.

Q3. Instead of combining existing models, can we learn local models as well as weights?

A3. In Chapter 6, we presented a new approach for low-rank matrix approximation based on the assumption that the matrix is locally low-rank. We proposed the LLORMA algorithms (parallel and global), which is highly parallelizable and thus scales well with the number of observations and the dimension of the problem. Both LLORMA and CF ensembles combine individual models based on locality using kernel smoothing, predict unseen ratings via weighted sum of individual models, and those weights are dependent on inputs. However, LLORMA learns both local models and their weights at the same time, while we assumed the individuals models are given in advance and learned only weights with CF ensembles.

Our experiments indicate that LLORMA outperforms several state-of-the-art methods without a significant computational overhead. We also presented a formal analysis of LLORMA by deriving bounds that generalize standard compressed sensing results and express the dependency of the modeling accuracy on the matrix size, training set size, and locality (kernel bandwidth parameter). Our method is applicable beyond recommendation systems so long as the locality assumption holds and a reasonable metric space can be identified.

Q4. Can we use local models to improve quality of ranking as well?

A4. With the previous question, it was observed that the local low-rank assumption applies to recommendation systems in the context of least squares rating prediction. In Chapter 7, we verified that the same conclusion holds when performance is evaluated using a variety of ranked loss functions. Specifically, we presented a novel collaborative ranking method called LCR (Local Collaborative Ranking) based on the local low-rank assumption, that is, the rating matrix is globally high-rank but locally low-rank. The LCR approach generalizes LLORMA in Chapter 6 from least

squares to ranked loss minimization, and it outperforms LLORMA and state-of-the-art ranked loss minimization methods for collaborative filtering.

Besides providing improved ranking accuracy, the LCR model also scales up to large datasets due to our proposed parallel training scheme. Another computational advantage stems from the fact that each local model may be restricted to a rank that is lower than the rank selected for a global model. Indeed, replacing a single rank l matrix factorization with multiple rank l' (local) matrix factorizations with $l' < l$ may lead to a win-win situation in terms of both computational efficiency and ranking accuracy.

9.2 Contributions

With the proposed local approaches for collaborative filtering and open source toolkit, we contribute research community and even wider audiences with better recommendation methods both theoretically and practically. We summarize our contributions in this section.

9.2.1 Better Quality of Recommendations

First of all, we present several local approaches that significantly improve the accuracy of recommendations. Ensembles of CF with non-constants weights in Chapter 5 and local low-rank matrix approximation in Chapter 6 lower root-mean squared error (RMSE) significantly. Although the improvement in RMSE looks small (less than 0.001), it has been shown since the Netflix competition that lowering the RMSE by 0.0001 is extremely difficult. Local collaborative ranking in Chapter 7 also improves ranking quality in terms of average precision and NDCG.

Local low-rank matrix approximation was not just verified by offline tests with standard datasets, but also was implemented and tested in industry including Google and Amazon during the author's internship. It was verified that local approaches in this thesis were effective to improve recommendation quality in real production, though we cannot provide exact figures publicly.

9.2.2 Theoretical Aspects

In addition to the improvement shown with experiments, we present important fundamental observations for collaborating filtering. The global low-rank assumption

has been prevailing in most previous works, but the local low-rank assumption presented in Chapter 6 fundamentally changes the underlying assumption for recommendation systems using matrix factorization. We show that the diminishing returns phenomenon we underwent with higher rank under the global low-rank assumption is mainly because we are stuck at local optimum, not because it is the best solution. We also provide theoretical analysis of the proposed local low-rank matrix approximation method.

9.2.3 Computational Aspects

The algorithms presented in this thesis are highly efficient and scalable for large scale dataset. Local low-rank matrix approximation (Chapter 6) and local collaborative ranking (Chapter 7) can run either in clusters or in a multiprocessor machine in parallel. As the learning process takes less time with a local model than with a global one, and as every local model can be learned at the same time (at least in theory, assuming that we have enough machines), the overall computation time for learning process is even shorter than previous global models despite even superior accuracy. We verify efficiency of our proposed algorithm with large scale data including Netflix dataset as well as real production data from Amazon Instant Video service. With sufficient computing resources, we proved that local approaches in this thesis can run fast enough for real production use.

9.2.4 Open Source Software

We distribute local approaches in this thesis (LLORMA and LCR) with many other baselines in the toolkit PREA presented in Chapter 3. (`http://prea.gatech.edu`) Researchers both in academia and industry will take advantage of this open source toolkit. For research scientists, state-of-the-art algorithms implemented in PREA will be great baselines to compare with their new methods. In industry, engineers can easily compare many candidate recommendation algorithms to find the one best fitting to their needs in applications. In some cases, simple and fast algorithm will be the best, while some other applications may allow heavy offline computation to get more accurate estimation.

PREA toolkit (Chapter 3) has been widely used in various areas of research communities, including recommendation systems [99, 95, 96, 97, 98, 14, 63, 62, 116, 113], natural language processing [68], crowd-sourcing [61, 60], optimization [111, 106],

computer vision [127], and computer networks [5, 6, 4, 43]. Even outside of computer science and engineering, PREA is used as a tool for matrix factorization in Material Science and Engineering, Aerospace Engineering, and so on.

9.3 Future Research Directions

This dissertation advances collaborative filtering-based recommendation systems with local approaches. Although they are effective and efficient, we might be able to further improve them. We briefly introduce a couple of possible future research directions in this section.

The LLORMA and LCR, the two methods under local low-rank assumption in Chapters 6 and 7, might be extended in two ways:

- **Better way of anchor point selection**: In current LLORMA, the center point of each local model is just randomly chosen. Although we have shown that a random choice works well with sufficiently large number of local models, it is still possible to waste computing power and memory, for example, if we select very similar anchor points more than once by coincidence. Extending the concept of anchor points from a user and item pair to any point in latent space representing users and items may improve the prediction accuracy with smaller number of local models.
- **LLORMA on contents-based recommender systems**: With user and item features, we can deal with cold-start users and items by embedding the feature vector to latent space. We may be able to achieve better performance by applying the local low-rank assumption for feature embeddings as well.

Also, local approaches for collaborative filtering may be useful for applications other than recommendation systems:

- **Crowd-sourcing recommendation**: One major problem with crowd-sourcing-based evaluation is coming from inadequate assignment of tasks to workers. By recording quality of work from user feedback [60], we can build a collaborative filtering based recommender systems to connect crowd workers to tasks. The local low-rank assumption might be beneficial for this application as well.
- **Real-time Advertising**: For the perspective of websites with ad slots, real-time ad selection is a kind of item (ad) recommendation problem for a given

user. On the other hand, for the perspective of advertisers, they have to choose several users to show their ads within their budget. This is more like a user recommendation for a given item. Collaborative filtering approaches are widely used in this area, and our local approaches might be adapted to solve key problems such as bidding price estimation or conversion rate estimation.

REFERENCES

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

[2] Haim Avron, Satyen Kale, Shiva Kasiviswanathan, and Vikas Sindhwani. Efficient and practical stochastic subgradient descent for nuclear norm regularization. In *Proc. of the International Conference on Machine Learning*, 2012.

[3] Suhrid Balakrishnan and Sumit Chopra. Collaborative ranking. In *Proc. of the ACM International Conference on Web Search and Data Mining*, 2012.

[4] Ejder Bastug, Mehdi Bennis, and Mérouane Debbah. Living on the edge: The role of proactive caching in 5g wireless networks. *IEEE Communications Magazine*, 52(8):82–89, 2014.

[5] Ejder Bastug, Mehdi Bennis, and Mérouane Debbah. Social and spatial proactive caching for mobile data offloading. In *IEEE International Conference on Communications Workshops (ICC)*, 2014.

[6] Ejder Bastug, Mehdi Bennis, and Mérouane Debbah. Think before reacting: Proactive caching in 5g small cell networks. 2014.

[7] C. Basu, H. Hirsh, and W. Cohen. Recommendation as classification: Using social and content-based information in recommendation. In *Proceedings of the National Conference on Artificial Intelligence*, pages 714–720, 1998.

[8] R. Bell, Y. Koren, and C. Volinsky. Modeling relationships at multiple scales to improve accuracy of large recommender systems. In *Proc. of the ACM SIGKDD*, 2007.

[9] R. Bell, Y. Koren, and C. Volinsky. All together now: A perspective on the netflix prize. *Chance*, 23(1):24–29, 2010.

[10] P. Bennett. Neighborhood-based local sensitivity. In *Proc. of the ECML*, 2007.

[11] D. Billsus and M. J. Pazzani. Learning collaborative information filters. In *Proc. of the International Conference on Machine Learning*, 1998.

[12] J. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proc. of Uncertainty in Artificial Intelligence*, 1998.

[13] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[14] Supaporn Bundasak and Krisana Chinnasarn. emenu recommender system using collaborative filtering and slope one predictor. In *Proc. of the International Joint Conference on Computer Science and Software Engineering (JCSSE)*. IEEE, 2013.

[15] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proc. of the International Conference on Machine Learning*, 2005.

[16] Jian-Feng Cai, Emmanuel J Candès, and Zuowei Shen. A singular value thresholding algorithm for matrix completion. *SIAM Journal on Optimization*, 20(4):1956–1982, 2010.

[17] E.J. Candès and Y. Plan. Matrix completion with noise. *Proc. of the IEEE*, 98(6):925–936, 2010.

[18] E.J. Candès and T. Tao. The power of convex relaxation: Near-optimal matrix completion. *IEEE Transactions on Information Theory*, 56(5):2053–2080, 2010.

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

[20] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *Proc. of the International Conference on Machine Learning*, 2007.

[21] Soumen Chakrabarti, Rajiv Khanna, Uma Sawant, and Chiru Bhattacharyya. Structured learning for non-smooth ranking losses. In *Proc. of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2008.

[22] S. H. S. Chee, J. Han, and K. Wang. Rectree: An efficient collaborative filtering method. In *Lecture Notes in Computer Science*, pages 141–151. Springer Verlag, 2001.

[23] Zheng Chen and Heng Ji. Collaborative ranking: a case study on entity linking. In *Proc. of the Conference on Empirical Methods in Natural Language Processing*, 2011.

[24] Boris Chidlovskii, Natalie S Glance, and M Antonietta Grasso. Collaborative re-ranking of search results. In *Proc. of AAAI-2000 Workshop on AI for Web Search*, 2000.

[25] Patrick L Combettes and Valérie R Wajs. Signal recovery by proximal forward-backward splitting. *Multiscale Modeling & Simulation*, 4(4):1168–1200, 2005.

[26] M. Connor and J. Herlocker. Clustering items for collaborative filtering. In *Proceedings of ACM SIGIR Workshop on Recommender Systems*, 2001.

[27] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proc. of the ACM Conference on Recommender Systems*, 2010.

[28] Souvik Debnath, Niloy Ganguly, and Pabitra Mitra. Feature weighting in content based recommendation system using social network analysis. In *Proc. of the International Conference on World Wide Web*, 2008.

[29] D. DeCoste. Collaborative prediction using ensembles of maximum margin matrix factorizations. In *Proc. of the International Conference on Machine Learning*, 2006.

[30] O. Dekel, C. Manning, and Y. Singer. Log-linear models for label ranking. 2003.

[31] J. Dillon, Y. Mao, G. Lebanon, and J. Zhang. Statistical translation, heat kernels, and expected distances. In *Uncertainty in Artificial Intelligence*, pages 93–100. AUAI Press, 2007.

[32] Pinar Donmez, Krysta M Svore, and Christopher JC Burges. On the local optimality of lambdarank. In *Proc. of International ACM SIGIR Conference*, 2009.

[33] Miro Dudik, Zaid Harchaoui, and Jérôme Malick. Lifted coordinate descent for learning with trace-norm regularization. In *Proc. of the International Conference on Artificial Intelligence and Statistics*, 2012.

[34] Chaosheng Fan, Yanyan Lan, Jiafeng Guo, Zuoquan Lin, and Xueqi Cheng. Collaborative factorization for recommender systems. In *Proc. of the International ACM SIGIR Conference*, 2013.

[35] Maryam Fazel, Haitham Hindi, and Stephen P Boyd. A rank minimization heuristic with application to minimum order system approximation. In *Proc. of the American Control Conference*, volume 6, 2001.

[36] R. Foygel and N. Srebro. Concentration-based guarantees for low-rank matrix reconstruction. *ArXiv Report arXiv:1102.3923*, 2011.

[37] R. Foygel, N. Srebro, and R. Salakhutdinov. Matrix reconstruction with the local max norm. *ArXiv Report arXiv:1210.5196*, 2012.

[38] Yoav Freund, Raj Iyer, Robert E Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4:933–969, 2003.

[39] Jill Freyne, Barry Smyth, Maurice Coyle, Evelyn Balfe, and Peter Briggs. Further experiments on collaborative ranking in community-based web search. *Artificial Intelligence Review*, 21(3-4):229–252, 2004.

[40] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

[41] A. Gunawardana and G. Shani. A survey of accuracy evaluation metrics of recommendation tasks. *Journal of Machine Learning Research*, 10:2935–2962, 2009.

[42] Severin Hacker and Luis von Ahn. Matchin: eliciting user preferences with an online game. In *Proc. of the SIGCHI Conference on Human Factors in Computing Systems*, 2009.

[43] Kenza Hamidouche, Ejder Batu, Mehdi Bennis, and Mérouane Debbah. Le caching proactif dans les réseaux cellulaires 5g. *La Revue de l'Electricité et de l'Electronique (REE). Dossier: Propagation et Télédetection*, 2014.

[44] Edward Harrington. Online ranking/collaborative filtering using the perceptron algorithm. In *Proc. of the International Conference on Machine Learning*, 2003.

[45] Elad Hazan. Sparse approximate solutions to semidefinite programs. In *LATIN 2008: Theoretical Informatics*, pages 306–316. Springer, 2008.

[46] D. Heckerman, D. Maxwell Chickering, C. Meek, R. Rounthwaite, and C. Kadie. Dependency networks for inference, collaborative filtering, and data visualization. *Journal of Machine Learning Research*, 1, 2000.

[47] R. Herbrich, T. Graepel, and K. Obermayer. Large margin rank boundaries for ordinal regression. In *Advances in Neural Information Processing Systems*, 1999.

[48] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *Proc. of ACM SIGIR Conference*, 1999.

[49] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22(1):5–53, January 2004.

[50] W. Hill, L. Stead, M. Rosenstein, and G. Furnas. Recommending and evaluating choices in a virtual community of use. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1995.

[51] Cho-Jui Hsieh and Peder Olsen. Nuclear norm minimization via active subspace selection. In *Proc. of the International Conference on Machine Learning*, 2014.

[52] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Proc. of the IEEE International Conference on Data Mining*, 2008.

[53] Z. Huang, D. Zeng, and H. Chen. A comparison of collaborative-filtering recommendation algorithms for e-commerce. *IEEE Intelligent Systems*, 22:68–78, 2007.

[54] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

[55] Martin Jaggi, Marek Sulovsk, et al. A simple algorithm for nuclear norm regularized problems. In *Proc. of the International Conference on Machine Learning*, 2010.

[56] M. Jahrer, A. Töscher, and R. Legenstein. Combining predictions for accurate recommender systems. In *Proc. of the ACM SIGKDD*, 2010.

[57] D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich. *Recommender Systems - An Introduction*. Cambridge, 2011.

[58] Shuiwang Ji and Jieping Ye. An accelerated gradient method for trace norm minimization. In *Proc. of the International Conference on Machine Learning*, 2009.

[59] T. Joachims. Optimizing search engines using clickthrough data. In *Proc. of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2002.

[60] Hyun Joon Jung. Quality assurance in crowdsourcing via matrix factorization based task routing. In *Proceedings of the companion publication of the 23rd international conference on World wide web companion*, pages 3–8. International World Wide Web Conferences Steering Committee, 2014.

[61] Hyun Joon Jung and Matthew Lease. Crowdsourced task routing via matrix factorization. *arXiv preprint arXiv:1310.5142*, 2013.

[62] Abhishek Kaleroun. *Hybrid Bee Colony Trust Mechanism in Recommender System*. PhD thesis, Thapar University Patiala, 2014.

[63] Abhishek Kaleroun and Shalini Batra. Collaborating trust and item-prediction with ant colony for recommendation. In *Proc. of the IEEE International Conference on Contemporary Computing (IC3)*, 2014.

[64] Nandakishore Kambhatla and Todd K Leen. Dimension reduction by local principal component analysis. *Neural Computation*, 9(7):1493–1516, 1997.

[65] R. H. Keshavan, A. Montanari, and S. Oh. Matrix completion from noisy entries. *Journal of Machine Learning Research*, 99:2057–2078, 2010.

[66] B. M. Kim and Q. Li. Probabilistic model estimation for collaborative filtering based on items attributes. In *Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 185–191, 2004.

[67] J. Kim. *Nonnegative matrix and tensor factorizations, least squares problems, and applications*. PhD thesis, 2011.

[68] S. Kim, F. Li, G. Lebanon, and I. Essa. Beyond sentiment: The manifold of human emotions. In *Proc. of the International Conference on Artificial Intelligence and Statistics*, 2013.

[69] Alex Kleeman, Nick Hendersen, and Sylvie Denuit. Matrix factorization for collaborative prediction. In *ICME*, 2005.

[70] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proc. of the ACM SIGKDD*, 2008.

[71] Y. Koren. Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM Transactions on Knowledge Discovery from Data*, 4(1):1–24, 2010.

[72] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[73] S. Kumar, M. Mohri, and A. Talwalkar. Ensemble nystrom method. In *Advances in Neural Information Processing Systems*, 2009.

[74] B. Lakshminarayanan, G. Bouchard, and C. Archambeau. Robust bayesian matrix factorisation. In *Proc. of the International Conference on Artificial Intelligence and Statistics*, 2011.

[75] N. D. Lawrence and R. Urtasun. Non-linear matrix factorization with gaussian processes. In *Proc. of the International Conference on Machine Learning*, 2009.

[76] G. Lebanon. Learning Riemannian metrics. In *Proc. of the 19th Conference on Uncertainty in Artificial Intelligence*. AUAI Press, 2003.

[77] G. Lebanon. Axiomatic geometry of conditional models. *IEEE Transactions on Information Theory*, 51(4):1283–1294, 2005.

[78] D. Lee and H. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791, 1999.

[79] D. Lee and H. Seung. Algorithms for non-negative matrix factorization. In *Advances in Neural Information Processing Systems*, 2001.

[80] J. Lee, M. Sun, and G. Lebanon. A comparative study of collaborative filtering algorithms. *ArXiv Report 1205.3193*, 2012.

[81] J. Lee, M. Sun, and G. Lebanon. Prea: Personalized recommendation algorithms toolkit. *Journal of Machine Learning Research*, 13:2699–2703, 2012.

[82] D. Lemire and A. Maclachlan. Slope one predictors for online rating-based collaborative filtering. *Society for Industrial Mathematics*, 5:471–480, 2005.

[83] Nathan N Liu and Qiang Yang. Eigenrank: a ranking-oriented approach to collaborative filtering. In *Proc. of the International ACM SIGIR Conference*, 2008.

[84] S. Ma, D. Goldfarb, and L. Chen. Fixed point and bregman iterative methods for matrix rank minimization. *Mathematical Programming*, 128:321–353, 2011.

[85] L. W. Mackey, A. S. Talwalkar, and M. I. Jordan. Divide-and-conquer matrix factorization. In *Advances in Neural Information Processing Systems*, 2011.

[86] B. Marlin. Modeling user rating profiles for collaborative filtering. In *Advances in Neural Information Processing Systems*, 2004.

[87] Rahul Mazumder, Trevor Hastie, and Robert Tibshirani. Spectral regularization algorithms for learning large incomplete matrices. *The Journal of Machine Learning Research*, 99:2287–2322, 2010.

[88] P. Melville, R. J. Mooney, and R. Nagarajan. Content-boosted collaborative filtering. In *In Proceedings of the 2001 SIGIR Workshop on Recommender Systems*, 2001.

[89] C. J. Merz. Dynamical selection of learning algorithms. *Lecture Notes in Statistics*, pages 281–290, 1996.

[90] N. Mirbakhsh and C. X. Ling. Clustering-based matrix factorization. *ArXiv Report arXiv:1301.6659*, 2013.

[91] Bamdev Mishra, Gilles Meyer, Francis Bach, and Rodolphe Sepulchre. Low-rank optimization with trace norm penalty. *SIAM Journal on Optimization*, 23(4):2124–2149, 2013.

[92] K. Miyahara and M. J. Pazzani. Collaborative filtering with the simple bayesian classifier. In *Proceedings of the 6th Pacific Rim International Conference on Artificial Intelligence*, pages 679–689, 2000.

[93] K. Miyahara and M. J. Pazzani. Improvement of collaborative filtering with the simple bayesian classifier 1. (11), 2002.

[94] Ananth Mohan, Zheng Chen, and Kilian Q Weinberger. Web-search ranking with initialized gradient boosted regression trees. *Journal of Machine Learning Research-Proceedings Track*, 14:77–89, 2011.

[95] Katja Niemann and Martin Wolpers. A new collaborative filtering approach for increasing the aggregate diversity of recommender systems. In *Proc. of the ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2013.

[96] Katja Niemann and Martin Wolpers. Usage context-boosted filtering for recommender systems in tel. In *Scaling up Learning for Sustained Impact*, pages 246–259. Springer, 2013.

[97] Katja Niemann and Martin Wolpers. Creating usage context-based object similarities to boost recommender systems in technology enhanced learning. 2014.

[98] Katja Niemann and Martin Wolpers. Usage-based clustering of learning resources to improve recommendations. In *Open Learning and Teaching in Educational Communities*, pages 317–330. Springer, 2014.

[99] Kyung-Wha Park, Byoung-Hee Kim, Tae-Suh Park, and Byoung-Tak Zhang. Uncovering response biases in recommendation. In *Workshops at the AAAI Conference on Artificial Intelligence*, 2014.

[100] Seungtaek Park and David M Pennock. Applying collaborative filtering techniques to movie search for better ranking and browsing. In *Proc. of the ACM SIGKDD International Conference*, 2007.

[101] Yoonjoo Park and Alexander Tuzhilin. The long tail of recommender systems and how to leverage it. In *Proc. of the ACM Conference on Recommender Systems*, 2008.

[102] A. Paterek. Improving regularized singular value decomposition for collaborative filtering. *Statistics*, 2007:2–5, 2007.

[103] D. Y. Pavlov and D. M. Pennock. A maximum entropy approach to collaborative filtering in dynamic, sparse, high-dimensional domains. In *Proceedings of Neural Information Processing Systems*, pages 1441–1448. MIT Press, 2002.

[104] D. M. Pennock, E. Horvitz, S. Lawrence, and C. L. Giles. Collaborative filtering by personality diagnosis: A hybrid memory- and model-based approach. In *Proc. of the Conference on UAI*, 2000.

[105] R. Polikar. Ensemble learning. 3(12):2776, 2008.

[106] Pitaya Poompuang and Wichian Premchaiswadi. Adaptive similarity interval for collaborative filtering optimization. *International Journal of Digital Content Technology & its Applications*, 7(14), 2013.

[107] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proc. of the Conference on Uncertainty in Artificial Intelligence*, 2009.

[108] J.D.M. Rennie and N. Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *Proc. of the International Conference on Machine Learning*, 2005.

[109] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *Proc. of the Conference on CSCW*, 1994.

[110] L. Reyzin and R. E. Schapire. How boosting the margin can also boost classifier complexity. In *Proceedings of the ICML*, pages 753–760. Association for Computational Linguistics, 2006.

[111] Fabio Roda. Integrating high-level requirements in optimization problems: theory and applications. *4OR*, 12(2):199–200, 2014.

[112] S. Roweis and L. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290:2323–2326, 2000.

[113] Almudena Ruiz Iniesta. Estrategias de recomendación basadas en conocimiento para la localización personalizada de recursos en repositorios educativos. 2014.

[114] R. Salakhutdinov and A. Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *Proc. of the International Conference on Machine Learning*, 2008.

[115] R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. In *Advances in Neural Information Processing Systems*, 2008.

[116] Bhavya Sanghavi, Rishabh Rathod, and Dharmeshkumar Mistry. Recommender systems-comparison of content-based filtering and collaborative filtering. *International Journal of Current Engineering and Technology*, 2014.

[117] B. Sarwar, G. Karypis, J. Konstan, and J. Reidl. Item-based collaborative filtering recommendation algorithms. In *Proc. of the International Conference on World Wide Web*, 2001.

[118] B. M. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering. In *Proceedings of the 5th International Conference on Computer and Information Technology*, 2002.

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

[120] S. Shalev-Shwartz, A. Gonen, and O. Shamir. Large-scale convex minimization with a low-rank constraint. In *Proc. of the International Conference on Machine Learning*, 2011.

[121] U. Shardanand and P. Maes. Social information filtering: algorithms for automating word of mouth. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1995.

[122] Y. Shi, A. Karatzoglou, L. Baltrunas, M. Larson, N. Oliver, and A. Hanjalic. Climf: learning to maximize reciprocal rank with collaborative less-is-more filtering. In *Proc. of the ACM Conference on Recommender Systems*, 2012.

[123] J. Sill, G. Takacs, L. Mackey, and D. Lin. Feature-weighted linear stacking. *Arxiv preprint arXiv:0911.0460*, 2009.

[124] N. Srebro, J. D. M. Rennie, and T. S. Jaakola. Maximum-margin matrix factorization. In *Advances in Neural Information Processing Systems*, 2005.

[125] X. Su, R. Greiner, T. M. Khoshgoftaar, and X. Zhu. Hybrid collaborative filtering algorithms using a mixture of experts. In *Proc. of the IEEE/WIC/ACM International Conference on Web Intelligence*, 2007.

[126] X. Su and T. M. Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, 2009:4:2–4:2, 2009.

[127] J Sun, Dhruv Parthasarathy, and K Varshney. Collaborative kalman filtering for dynamic matrix factorization. 2014.

[128] M. Sun, G. Lebanon, and P. Kidwell. Estimating probabilities in recommendation systems. In *Proc. of the International Conference on Artificial Intelligence and Statistics*, 2011.

[129] P. Symeonidis, A. Nanopoulos, and Y. Manolopoulos. Providing justifications in recommender systems. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 38(6):1262–1272, 2008.

[130] Tom F Tan and Serguei Netessine. Is tom cruise threatened? using netflix prize data to examine the long tail of electronic commerce. *Wharton Business School, University of Pennsylvania, Philadelphia*, 2009.

[131] Michael Taylor, John Guiver, Stephen Robertson, and Tom Minka. Softrank: optimizing non-smooth rank metrics. In *Proc. of the International Conference on Web Search and Web Data Mining*, 2008.

[132] K.C. Toh and S. Yun. An accelerated proximal gradient algorithm for nuclear norm regularized linear least squares problems. *Pacific Journal of Optimization*, 6(15):615–640, 2010.

[133] Ming-Feng Tsai, Tie-Yan Liu, Tao Qin, Hsin-Hsi Chen, and Wei-Ying Ma. Frank: a ranking method with fidelity loss. In *Proc. of International ACM SIGIR Conference*, 2007.

[134] L. H. Ungar and D. P. Foster. Clustering methods for collaborative filtering. In *AAAI Workshop on Recommendation Systems*, 1998.

[135] Maksims Volkovs and Richard S Zemel. Collaborative ranking with 17 parameters. In *Advances in Neural Information Processing Systems*, 2012.

[136] S. Vucetic and Z. Obradovic. Collaborative filtering using a regression-based approach. *Knowledge and Information Systems*, 7:1–22, 2005.

[137] M. P. Wand and M. C. Jones. *Kernel Smoothing*. Chapman and Hall/CRC, 1995.

[138] Jun Wang, Stephen Robertson, Arjen P de Vries, and Marcel JT Reinders. Probabilistic relevance ranking for collaborative filtering. *Information Retrieval*, 11(6):477–497, 2008.

[139] Yi Wang, Arthur Szlam, and Gilad Lerman. Robust locally linear analysis with applications to image denoising and blind inpainting. *SIAM Journal on Imaging Sciences*, 6(1):526–562, 2013.

[140] Markus Weimer, Alexandros Karatzoglou, Quoc V. Le, and Alex Smola. Cofi rank: Maximum margin matrix factorization for collaborative ranking. In *Advances in Neural Information Processing Systems*, 2007.

[141] J. Weston, S. Bengio, and N. Usunier. Wsabie: Scaling up to large vocabulary image annotation. In *Proc. of the International Joint Conference on Artificial Intelligence*, 2011.

[142] K. Woods, W.P. Kegelmeyer Jr, and K. Bowyer. Combination of multiple classifiers using local accuracy estimates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):405–410, 1997.

[143] E. Xing, A. Ng, M. Jordan, and S. Russel. Distance metric learning with applications to clustering with side information. In *Advances in Neural Information Processing Systems*, 2003.

[144] Jun Xu and Hang Li. Adarank: a boosting algorithm for information retrieval. In *Proc. of International ACM SIGIR Conference*, 2007.

[145] Jun Xu, Tie-Yan Liu, Min Lu, Hang Li, and Wei-Ying Ma. Directly optimizing evaluation measures in learning to rank. In *Proc. of International ACM SIGIR Conference*, 2008.

[146] G. R. Xue, C. Lin, Q. Yang, W. S. Xi, H. J. Zeng, Y. Yu, and Z. Chen. Scalable collaborative filtering using cluster-based smoothing. In *Proc. of ACM SIGIR Conference*, 2005.

[147] K. Yu, S. Zhu, J. Lafferty, and Y. Gong. Fast nonparametric matrix factorization for large-scale collaborative filtering. In *Proc. of ACM SIGIR Conference*, 2009.

[148] Yisong Yue, Thomas Finley, Filip Radlinski, and Thorsten Joachims. A support vector method for optimizing average precision. In *Proc. of International ACM SIGIR Conference*, 2007.

[149] Xinhua Zhang, Dale Schuurmans, and Yao-liang Yu. Accelerated training for matrix-norm regularization: A boosting approach. In *Advances in Neural Information Processing Systems*, 2012.

[150] C.-N. Ziegler, G. Lausen, and Schmidt-Thie L. Taxonomy-driven computation of product recommendations. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 406–415, 2004.