# Improving the Quality of Top-$N$ Recommendation

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

**Evangelia Christakopoulou**

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

**Dr. George Karypis, Advisor**

**February, 2018**

# Acknowledgements

This thesis was accomplished with the tremendous help of many amazing people, who I would like to thank from the bottom of my heart.

First and foremost, I would like to thank my advisor George Karypis. He is a great teacher, and he has taught me so many skills in so many areas: data mining, machine learning, software development, scientific writing and critical thinking. I would like to thank him for believing in me and for giving me the great opportunity of coming to the States. He is a truly amazing mentor, and I could not thank him enough for everything he has done. I feel very lucky to have such an extremely intelligent and talented, but also so good-hearted advisor guiding me through the PhD, and through life. I would not be where I am today without him.

I would like to thank Professors Arindam Banerjee, Joseph Konstan, Gedas Adomavicius, and Jaideep Srivastava for serving on my thesis and preliminary committees. I feel very fortunate for having a committee comprising of such intelligent, insightful, and accomplished professors with deep knowledge in the field. Their comments and suggestions have shaped my research and their guidance has been invaluable.

A very special heartfelt and huge thanks to my amazing sister Konstantina. She has helped me in countless ways, from our discussions on machine learning and data mining as she is such an amazing researcher, to providing tremendous emotional support throughout good and hard times. She is always next to me and the US and grad school journey has been no exception. She inspires me, and helps me grow every day and I would not be the person I am without her. I would like to thank her for being such an amazing sister, friend and the best roommate anyone could possibly have. I am now looking forward to our future collaboration together- it has been long awaited for!

I would also like to thank very much my parents, Eleni and Andreas. They are

always my rock and their continuous love and support is one of the main sources of my energy and well-being. I could not have done this without them standing by me in all of my decisions, even if these made things harder for them. I would like to thank them for a myriad things they are doing for me and have done so in all my life, but we do not have enough space. I would like to especially thank my amazing mum, who is also my role model and my inspiration, who has taught me to be strong and who has gotten on the plane a few more times than she would like to, just to be there for me. She is my first teacher/mentor and the person who taught me to believe in me.

I would like to thank my friends back home who are so loving and understanding, especially Anastasia, Maria, Kostas and Maria. Also, the friends I made in the States who make me feel so at home and happy, especially Agi, Iwanna, Maria, Andreas, Nikos, Nikos, Panos and Vasilis. I would also like to thank all of my friends on the fourth floor of DTC, our talks have made my days and I am so glad we went through all of this together. I would also like to thank my family for their unconditional support; especially my uncle Yorgos. A very warm and special thanks to my boyfriend who has helped and supported me so much, and who is one of the closest people to me; without him things would be so very different.

I have been very lucky spending my days (and often evenings) with intelligent and truly good people - my labmates at Karypis Lab. They have helped me in my bad times and they have been very happy with me during the good times and throughout all of the times they have helped me learn a lot. So, huge thanks to my girls: Agi, Ancy, Asmaa, Maria, Sara, Shalini, and Xia for everything; I feel grateful. Also, thanks to Dominique, Shaden, Saurav, Mohit, Jeremy, David, Santosh, Rezwan, and Haoji for teaching me so much and for being friends whose company has given me great joy.

Furthermore, a great thanks to the staff at the Department of Computer Science, the Digital Technology Center, and the Minnesota Supercomputing Institute at the University of Minnesota for providing the resources which were crucial for my research and for helping me on many things. Last, I would like to thank all of the great mentors I was fortunate to have: my mentors at the internships Shipeng Yu, Abhishek Gupta, Ajay Bangla and Shilpa Arora, and my advisors in undergraduate degree Nikolaos Avouris and Sophia Daskalaki for teaching me, and helping me achieve my goals.

# Dedication

To my parents, Eleni and Andreas.

For being the most supportive and loving parents in the world, and helping me live my dreams. Without them, I would not be in the position of writing this thesis today.

## Abstract

Top-$N$ recommenders are systems that provide a ranked list of $N$ products to every user; the recommendations are of items that the user will potentially like. Top-$N$ recommendation systems are present everywhere and used by millions of users, as they enable them to quickly find items they are interested in, without having to browse or search through big datasets; an often impossible task. The quality of the recommendations is crucial, as it determines the usefulness of the recommender to the users. So, how do we decide which products should be recommended? Also, how do we address the limitations of current approaches, in order to achieve better quality?

In order to provide insight into these problems, this thesis focuses on developing *novel, scalable algorithms that improve the state-of-the-art* top-$N$ recommendation quality, while providing insight into the top-$N$ recommendation task. The developed algorithms address some of the limitations of existent top-$N$ recommendation approaches and can be applied to real-world problems and datasets. The main areas of our contributions are the following:

1. **Exploiting higher-order sets of items**: We investigate to what extent higher-order sets of items are present in real-world datasets, beyond pairs of items. We also show how to best utilize them to improve the top-$N$ recommendation quality.
2. **Estimating a global and multiple local models**: We show that estimating multiple user-subset specific local models, beyond a global model significantly improves the top-$N$ recommendation quality. We demonstrate this with both item-item models and latent space models.
3. **Investigating and using the error**: We investigate what are the properties of the error and how they correlate with the top-$N$ recommendation quality, in methods that treat the missing entries as zeros. Then, we utilize the learned insights to develop a method, which explicitly uses the error.

We have applied our algorithms to big datasets, with *millions* of ratings, that span different areas, such as grocery transactions, movie ratings, and retail transactions, showing significant improvements over the state-of-the-art.

# Contents

**Acknowledgements** **i**

**Dedication** **iii**

**Abstract** **iv**

**List of Tables** **viii**

**List of Figures** **x**

**1 Introduction** **1**

1.1 Contributions . . . . . 1

1.2 Outline . . . . . 4

1.3 Related publications . . . . . 5

**2 Definitions & Notation** **7**

2.1 Mathematical notations . . . . . 7

2.2 Recommender systems notation . . . . . 7

2.3 Common notations reference . . . . . 8

**3 Related Work** **10**

3.1 Neighborhood-based top-$N$ recommendation approaches . . . . . 11

3.1.1 SLIM . . . . . 11

3.2 Latent space top-$N$ recommendation approaches . . . . . 13

3.2.1 PureSVD . . . . . 13

3.3 Ranking-based top-$N$ recommendation approaches . . . . . 14

3.4 Higher-order methods for top-$N$ recommendation . . . . . 14
3.5 Local models for top-$N$ recommendation . . . . . 14

**4 Datasets and Evaluation Methodology 17**
4.1 Datasets . . . . . 17
4.2 Evaluation methodology . . . . . 19
4.3 Performance metrics . . . . . 19
4.4 Comparison algorithms . . . . . 20

**5 Higher-Order Sparse LInear Method for Top-$N$ Recommendation 22**
5.1 Introduction . . . . . 23
5.2 Proposed approach . . . . . 24
5.2.1 Overview . . . . . 24
5.2.2 Estimation of the sparse aggregation coefficient matrices . . . . . 25
5.3 Experimental results . . . . . 27
5.3.1 Verifying the existence of higher-order relations . . . . . 28
5.3.2 Performance comparison . . . . . 30
5.4 Conclusion . . . . . 35

**6 Local Item-Item Models for Top-N Recommendation 36**
6.1 Introduction . . . . . 36
6.2 Proposed approach . . . . . 37
6.2.1 Motivation . . . . . 37
6.2.2 Overview . . . . . 39
6.2.3 Estimating the item-item models . . . . . 40
6.2.4 Finding the optimal assignment of users to subsets . . . . . 41
6.3 Experimental results . . . . . 42
6.3.1 Performance of the proposed methods . . . . . 45
6.3.2 Performance against competing approaches . . . . . 51
6.3.3 Time complexity . . . . . 52
6.4 Conclusion . . . . . 54

**7 Local Latent Space Models for Top-$N$ Recommendation** **55**
7.1 Introduction . . . 56
7.2 Proposed approach . . . 57
7.2.1 Motivation . . . 57
7.2.2 Overview . . . 57
7.2.3 Estimation . . . 58
7.2.4 Prediction and recommendation . . . 61
7.3 Experimental results . . . 61
7.3.1 Performance of the proposed methods . . . 63
7.3.2 Performance against competing approaches . . . 68
7.4 Conclusion . . . 71

**8 Investigating & Using the Error in Top-$N$ Recommendation** **73**
8.1 Introduction . . . 74
8.2 Notation and definitions . . . 75
8.2.1 Error on the missing entries . . . 75
8.2.2 Similarity matrices . . . 75
8.3 Analysis of the properties of the error for SLIM and PureSVD . . . 76
8.3.1 Theoretical analysis . . . 76
8.3.2 Experimental analysis . . . 78
8.4 Proposed approach . . . 85
8.4.1 Overview . . . 85
8.5 Experimental results . . . 86
8.6 Conclusion . . . 89

**9 Conclusion** **90**
9.1 Thesis summary . . . 90
9.2 Future research directions . . . 93

**References** **95**

# List of Tables


2.1 Common notations used in this thesis. . . . . 9
4.1 Dataset characteristics. . . . . 18
5.1 The average basket size of datasets we evaluated HOSLIM on. . . . . 27
5.2 HOSLIM: Coverage by affected users. . . . . 28
5.3 HOSLIM: Coverage by non-zeros. . . . . 29
5.4 Comparison of 1st order with 2nd order models. . . . . 31
5.5 Comparison of the HR of constrained HOSLIM with unconstrained HOSLIM and SLIM. . . . . 35
6.1 Comparison between our proposed approaches for the *groceries* dataset. 46
6.2 Comparison between our proposed approaches for the *ml10m* dataset. . 46
6.3 Comparison between our proposed approaches for the *jester* dataset. . . 46
6.4 Comparison between our proposed approaches for the *flixster* dataset. . 47
6.5 Comparison between our proposed approaches for the *netflix* dataset. . 47
6.6 Comparison of GLSLIM with competing approaches in terms of HR. . . 51
6.7 Comparison of GLSLIM with competing approaches in terms of ARHR. 52
7.1 Comparison between our proposed approaches for the *groceries* dataset. 65
7.2 Comparison between our proposed approaches for the *ml10m* dataset. . 66
7.3 Comparison between our proposed approaches for the *jester* dataset. . . 66
7.4 Comparison between our proposed approaches for the *flixster* dataset. . 66
7.5 Comparison between our proposed approaches for the *netflix* dataset. . 67
7.6 Comparison with competing latent space approaches in terms of HR. . . 69
7.7 Comparison with competing latent space approaches in terms of ARHR. 70
7.8 Comparison of global approaches with global & local approaches in terms of HR. . . . . 70

7.9 Comparison of global approaches with global & local approaches in terms of ARHR. . . . . 70
7.10 The training time for *ml10m* dataset with 5 clusters. . . . . 71
8.1 Overview of the notations used in this chapter. . . . . 75
8.2 Comparison between SLIM, ESLIM-u and ESLIM-i in terms of HR. . . 87
8.3 Comparison between SLIM, ESLIM-u and ESLIM-i in terms of ARHR. 87

# List of Figures


2.1 The user-item implicit feedback matrix $\mathbf{R}$. . . . . 8
3.1 A sparse aggregation coefficient matrix $\mathbf{S}$. . . . . 12
5.1 An example of the HOSLIM matrices $\mathbf{R}'$ and $\mathbf{S}'$. . . . . 25
5.2 Varying the size of the top-$N$ list for HOSLIM. . . . . 32
5.3 Effect of $\sigma$ on the performance of HOSLIM. . . . . 34
6.1 (a) Local item-item models improve upon global item-item model. (b) Global item-item model and local models yield the same results. . . . . 38
6.2 The effect of the number of clusters on the performance of GLSLIM. . . 48
6.3 Comparing the performance of GLSLIM with CLUTO initialization versus with random initialization of user subsets. . . . . 49
6.4 How the $l_1$ norm of the global model $\mathbf{S}$ and local models $\mathbf{S}^{p_u}$ changes from the beginning of GLSLIM until convergence. . . . . 49
6.5 Varying the size of the top-$N$ list for GLSLIM. . . . . 50
6.6 The speedup achieved by GLSLIM on the *ml10m* dataset, while increasing the number of nodes. . . . . 53
6.7 The total time in mins achieved by GLSLIM with and without warm start on the *ml10m* dataset, while increasing the number of nodes. . . . 53
7.1 The performance of the proposed methods: LSVD, sLSVD, rLSVD, GLSVD, sGLSVD, and rGLSVD when varying the number of user subsets, in terms of ARHR for the *ml10m* dataset. . . . . 67
7.2 The performance of sGLSVD in terms of HR for different sizes $N$ of the recommendation list. . . . . 68

8.1 Scatterplot of rating and error similarities $a_{uv}$ and $c_{uv}$ for all pairs of users $u$ and $v$, for a good-performing SLIM model (estimated with $\beta = 1$ and resulting in HR = 0.33) and a worse-performing one (estimated with $\beta = 150$ and resulting in HR = 0.24) for the *ml100k* dataset. . . . . 79

8.2 Scatterplot of rating and error similarities $a_{uv}$ and $c_{uv}$ for all pairs of users $u$ and $v$, for a good-performing PureSVD model (estimated with $f = 50$ and resulting in HR = 0.296) and for a bad-performing PureSVD model (estimated with $f = 500$ and resulting in HR = 0.056), for the *ml100k* dataset. . . . . 80

8.3 The effect of the $l_2$ regularization $\beta$ on the performance of SLIM and on the corresponding 'User Rating.Error Similarity' and 'Item Rating.Error Similarity'. The maximum HR and ARHR are achieved for the values of $\beta$ for which the 'User Rating.Error Similarity' and 'Item Rating.Error Similarity' also obtain their local maxima. . . . . 81

8.4 The effect of the rank $f$ on the performance of PureSVD and on the corresponding 'User Rating.Error Similarity' and 'Item Rating.Error Similarity'. The maximum HR and ARHR are achieved for the values of the rank $f$ for which the 'User Rating.Error Similarity' and 'Item Rating.Error Similarity' also obtain their local maxima. . . . . 82

8.5 Examining how close the user rating-based and error-based representations are, in terms of their average cosine similarity and the frobenius norm of their difference for the *delicious* dataset, for SLIM and PureSVD models, while varying the respective parameters. The cosine similarity takes its highest values for the parameter values for which the frobenius norm of their difference takes its lowest values. . . . . 84

8.6 The performance of ESLIM-u and ESLIM-i for the tail items (50% least frequent items), while varying the $l_u/l_i$ regularization parameters. The performance of SLIM on the tail items is also shown for comparison purposes. . . . . 87

# Chapter 1

# Introduction

In today's society, consumers have a huge variety of products to choose from; the options seem limitless in many cases [1]. Choosing the product that best fits their needs is a needle in a haystack problem.

This is why a good recommendation system which takes into account their needs, as inferred from their likes, views, purchases, clicks and absence of those, while at the same time going through all possible items for them instead of them, seems promising and much needed. As a result, recommender systems have exploded in the last decade [2, 3, 4] and lots of online web portals, systems and websites heavily rely on recommendation to help users find what they want the most, in a personalized way that fits their needs.

## 1.1 Contributions

The work presented in this thesis lies in the general area of recommender systems. It is focused on *developing algorithms to improve the accuracy of top-$N$ recommendation systems, utilizing implicit feedback data.*

Top-$N$ recommenders identify a small number of $N$ items that a user will find useful to purchase, view, like, click e.t.c. among a large collection of such items by leveraging historical information from that and other users. They are wildly popular, ranging from Netflix movie recommendations, to Amazon product recommendations, to Facebook friend recommendations e.t.c.

The focus of the thesis is on top-$N$ recommenders and not on the more traditional

rating prediction systems which try to estimate the ratings for missing items as accurately as possible. The reason is that users tend to look at the top provided recommendations, without being interested in the recommended items in the middle or at the bottom of the list. It is thus a lot more important to provide good top recommendations, instead of accurately predicting the ratings for all possible items, as is the case for rating prediction.

Also, the work presented in this thesis utilizes implicit historical data, as they are more prevalent than explicit ratings or additional side information, and thus more easily available.

Within this context, this thesis has made significant contributions along the following directions.

### Higher-Order Sparse LInear Method for Top-$N$ Recommendation

Item-item approaches, that explore co-occurrence relationships between pairs of items, have been shown to be very effective for the top-$N$ recommendation task. However, in many application domains, users tend to consume items in sets, and the sets that different users consume are often overlapping. For example, in a grocery store, people tend to buy multiple items that are required to make a certain dish and in a music streaming site, users tend to listen to songs that are organized in playlists. This thesis (Chapter 5) shows that the recommendation quality can be improved by identifying and exploiting these sets of items. It also presents an approach (HOSLIM) based on structural equation modeling to generalize the item-item approaches to also incorporate itemset-item information. The experimental evaluation of this approach, performed on a variety of real-world datasets, shows that HOSLIM achieves considerable improvements of 7.86% on average over competing item-item approaches. Also, for domains that exhibit such set-based consumption characteristics, the gains can reach up to 32% over competing baselines.

### Local Item-Item Models for Top-$N$ Recommendation

The item-item approaches, although being very well-suited for the top-$N$ recommendation task, suffer from the fact that since they estimate a single model, they are not

very personalized to the individual users. For example, there could be a pair of items that are extremely similar for a specific user subset, while they have low similarity for another user subset. By using a global model, the similarity between these items will tend to be towards some average value; thus losing the high correlation of the pair for the first user subset. Building on this insight, this thesis (Chapter 6) presents an approach (GLSLIM) that combines the global model along with local item-item models estimated for different subsets of users. The assignment of the users to the subsets is not made apriori, but it is discovered as part of the optimization problem. The recommendations for a user are derived by aggregating information from a global model, which captures population-wide preferences, and a local model that captures the preferences of like-minded users. The evaluation performed on various real-world datasets shows that GLSLIM achieves better results than the standard global approach and also outperforms other state-of-the-art approaches, on average by 9.29% and up to 17.37%.

### Local Latent Space Models for Top-$N$ Recommendation

Further pursuing this research direction, this thesis (Chapter 7) also studies the benefits that such a global and local approach can provide to latent space top-$N$ recommendation approaches. Users' behaviors are driven by their preferences across various aspects. Latent space approaches model these aspects in the form of latent factors. Though such a user-model has been shown to lead to good results, the aspects that different users care about can vary. In many domains, there may be a set of aspects for which all users care about and a set of aspects that are specific to different subsets of users. Following this insight, the thesis proposes two latent space models: rGLSVD and sGLSVD, that combine such a global and user subset specific sets of latent factors. The rGLSVD model assigns the users into fixed subsets based on their rating patterns and then estimates a global and a set of user subset specific local models whose number of latent dimensions can vary. The sGLSVD model estimates both global and user subset specific local models by keeping the number of latent dimensions the same among these models but optimizes the grouping of the users in order to achieve the best approximation. The experimental evaluation shows that the proposed approaches outperform state-of-the-art latent space top-$N$ recommendation approaches on average by 13% and up to 37%.

**Investigating & Using the Error in Top-$N$ Recommendation**

Different popular top-$N$ recommendation approaches, such as SLIM [5] and PureSVD [6], treat the missing entries as zeros; in other words they make the assumption that the unconsumed items are probably disliked. However, in order to perform top-$N$ recommendation, they sort the predicted values of the missing entries and they recommend the ones with the highest values, thus the ones that have the highest error. In other words, they recommend the unconsumed items for which the underlying assumption is wrong and the corresponding items are probably liked by the user. The question then becomes: given a dataset, what are the properties of the error, how they correlate with the top-$N$ recommendation quality, and how the performance of the underlying methods can be improved by utilizing the error properties uncovered? This thesis (Chapter 8) attempts to answer this question, showing that for SLIM and PureSVD methods, users and items with similar ratings also have similar errors in their missing entries, and vice versa. Also, for the same training set, among the different models that are estimated by changing their respective hyperparameters, the ones that achieve the best recommendation performance are those that display the closest rating-based and error-based similarities. Also, utilizing this insight, the thesis proposes a method called ESLIM, which explicitly enforces users with similar rating behavior to also have similar error, and likewise for items. The method is shown to outperform SLIM, especially for items that have been rated by a few users (tail items).

## 1.2 Outline

The remainder of the thesis is organized as follows:

- Chapter 2 defines the notation used.
- Chapter 3 gives an overview of the prior related work on top-$N$ recommendation.
- Chapter 4 describes the datasets used and the evaluation methodology employed in the thesis.
- Chapter 5 presents the Higher-Order Sparse Linear Method for top-$N$ recommendation (HOSLIM), which utilizes higher order sets of items present in the data.

- Chapter 6 presents the Global and Local Sparse Linear Method for top-$N$ recommendation (GLSLIM), which estimates multiple local item-item models, beyond a global item-item model.

- Chapter 7 extends the idea of multiple local models to latent space methods and presents two methods that estimate both a global low-rank model and multiple user subset specific low-rank models: the Global and Local Singular Value Decomposition with varying ranks method (rGLSVD) and the Global and Local Singular Value Decomposition with varying subsets method (sGLSVD) for top-$N$ recommendation.

- Chapter 8 analyzes the properties of the error for SLIM and PureSVD and describes the ESLIM method.

- Chapter 9 summarizes the collective conclusions drawn by the works in the thesis and discusses future directions.

## 1.3 Related publications

The work presented in this thesis has been published in a variety of top-tier conferences and journals. The related publications are presented in the following list:

- **Evangelia Christakopoulou** and George Karypis. HOSLIM: higher-order sparse linear method for top-n recommender systems. *In Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 38–49. Springer, Cham, 2014. [7]

- **Evangelia Christakopoulou**. Moving beyond linearity and independence in top-n recommender systems. *In Proceedings of the 8th ACM Conference on Recommender systems*, pages 409–412. ACM, 2014. [8]

- **Evangelia Christakopoulou** and George Karypis. Local item-item models for top-n recommendation. *In Proceedings of the 10th ACM Conference on Recommender Systems*, pages 67–74. ACM, 2016. [9]

- David Anastasiu, **Evangelia Christakopoulou**, Shaden Smith, Mohit Sharma and George Karypis. Big data and recommender systems. *In Novatica: Journal*

*of the Spanish Computer Scientist Association*. 2016. [10]

- **Evangelia Christakopoulou**, Shaden Smith, Mohit Sharma, Alex Richards, David Anastasiu and George Karypis. Scalability and distribution of collaborative recommenders. *In Collaborative Recommendations: Algorithms, Practical Challenges and Applications*. 2018. [11]

- **Evangelia Christakopoulou** and George Karypis. Investigating & using the error in top-$N$ recommendation. *Ready for submission*.

- **Evangelia Christakopoulou** and George Karypis. Local Latent Space Models for Top-$N$ Recommendation. *Under review*.

# Chapter 2

# Definitions & Notation

This chapter introduces definitions in top-$N$ recommendation that will be useful in this thesis. It also provides the notation that will be followed in the following chapters of the thesis.

## 2.1 Mathematical notations

All *vectors* are represented by bold lower case letters and they are column vectors (e.g., $\mathbf{p}$, $\mathbf{q}$). Row vectors are represented by having the transpose superscript $^T$, (e.g., $\mathbf{p}^T$).

All *matrices* are represented by bold upper case letters (e.g., $\mathbf{R}$, $\mathbf{A}$, $\mathbf{U}$). For a given matrix $\mathbf{A}$, its $i$th row is represented by $\mathbf{a}_i^T$ and its $j$th column by $\mathbf{a}_j$. The element of matrix $\mathbf{A}$ that corresponds to the $i$th row and $j$th column is noted as $a_{ij}$.

We use calligraphic letters to denote *sets*.

A *predicted value* is denoted by having a $\sim$ over it (e.g., $\tilde{r}$).

We use the symbol $\odot$ to denote the Hadamart product (*element-wise multiplication*).

## 2.2 Recommender systems notation

The number of *users* are denoted by $n$ and the number of *items* are denoted by $m$. Symbols $u$ and $i$ will be used to denote individual users and items, respectively.

Matrix $\mathbf{R}$ is used to represent the *user-item implicit feedback* matrix of size $n \times m$, containing the items that the users have purchased/viewed/rated. So, the implicit

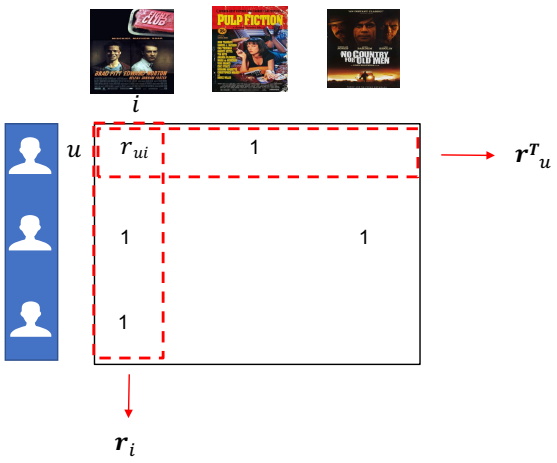



Figure 2.1: The user-item implicit feedback matrix $\mathbf{R}$.

behavior of user $u$ is presented in $\mathbf{r}_u^T$ and the implicit feedback of all users for items $i$ is shown in $\mathbf{r}_i$. If user $u$ provided feedback for item $i$, the $r_{ui}$ entry of $\mathbf{R}$ is 1, otherwise it is 0. An illustration of matrix $\mathbf{R}$ is shown in Figure 2.1.

We use the term *rating* to refer to the non-zero entries of $\mathbf{R}$, even though these entries can represent implicit feedback. We also refer to the items that the user has purchased/viewed/rated as *rated* items and to the rest as *unrated* items. The set of items that the user $u$ has rated is denoted by $\mathcal{R}_u$.

The number of items to be recommended is $N$, which is by default 10, unless stated otherwise.

## 2.3 Common notations reference

A reference table containing all the common notations is provided, for convenience of the reader (Table 2.1).

The top part of the table contains the general top-$N$ recommendation notation, introduced in this chapter. The bottom part of the table contains the notation belonging to popular top-$N$ recommendation methods (namely $k$-NN, PureSVD and SLIM) that are described in Chapter 3, and which will be used throughout the thesis. The reason why they are presented in this Table 2.1 is for convenience of the reader, who wants to look up commonly used notations.

Table 2.1: Common notations used in this thesis.

| Symbol | Definition |
|---|---|
| $n$ | number of users |
| $m$ | number of items |
| $u$ | individual user |
| $i$ | individual item |
| $\mathbf{R}$ | $n \times m$ implicit feedback matrix |
| $\mathbf{r}_u^T$ | implicit feedback of user $u$ |
| $\mathbf{r}_i$ | implicit feedback of item $i$ |
| $r_{ui}$ | feedback of user $u$ to item $i$ |
| $\mathcal{R}_u$ | set of items the user $u$ has rated |
| $N$ | the number of recommended items |
| $\mathbf{S}$ | $m \times m$ SLIM coefficient matrix |
| $k$ | number of neighbors |
| $f$ | number of latent factors |
| $\mathbf{P}$ | $n \times f$ user latent matrix |
| $\mathbf{Q}$ | $m \times f$ item latent matrix |

The rest of the notations, that are algorithm-specific, are described in each chapter, when needed.

# Chapter 3

# Related Work

There has been extensive research dedicated to the top-$N$ recommendation task [2, 4, 5, 12, 13, 14, 15, 16]. In this chapter, we will present a few of the most notable methods developed for the top-$N$ recommendation task, that use only user-item feedback data, with a special emphasis on implicit feedback data. Methods that utilize context [17, 18], or social information [19, 20] are outside the scope of this thesis. The methods shown in **bold** below are the **baselines** against which we compare our methods, which are presented in this thesis. The way we evaluate them is described in Section 4.4.

The methods developed to tackle the top-$N$ recommendation task broadly fall into two categories: the neighborhood-based (which focus either on users or items) and the latent space ones. The latent space methods [6] perform a low-rank factorization of the user-item matrix into user factor and item factor matrices, which represent both the users and the items in a common latent space. The neighborhood-based methods [12] (user-based or item-based) focus on identifying similar users/items based on the rating matrix.

The latent space methods have been shown to be superior for solving the rating prediction problem, in which every missing rating is estimated as accurately as possible [21, 22, 23, 24, 25, 26, 27]. However, it is the neighborhood methods that have been shown to be better for the top-$N$ recommendation problem [5, 12, 28, 29]. Among the latter, the item-based methods, which include item k-NN [12] and Sparse LInear Methods (SLIM) [5], have been shown to outperform the user-based schemes for the top-$N$ recommendation task. In fact, the winning method in a recent million song dataset

challenge [28] was a rather straightforward item-based neighborhood top-$N$ recommendation approach.

## 3.1 Neighborhood-based top-$N$ recommendation approaches

The neighbor methods ($k$-**NN**) consist of the user-based and the item-based ones. The user-based neighbor methods [30, 31, 32] first identify the $k$ users that are most similar to the target user. Then, they compute the union of items purchased by these users and associate a weight with each item, based on how frequently it was purchased by the $k$ most similar users. From this union, the $N$ items that have the highest weight and have not been purchased by the target user are recommended.

The traditional approaches for developing item-based top-$N$ recommendation methods ($k$-NN) [12, 33, 34, 35] use various vector-space similarity measures (e.g., cosine, extended Jaccard, Pearson correlation coefficient, etc.) to identify for each item the $k$ most similar other items based on the sets of users that co-rated these items. Then, given a set of items that have already been rated by a user, they derive their recommendations by combining the most similar unrated items to those already rated. Karypis [34] and Deshpande and Karypis [12] in particular showed that item-based models lead to better top-$N$ recommendation than user-based.

In recent years, the performance of these item-based neighborhood schemes has been significantly improved by using supervised learning methods to learn a model that both captures the similarities (or aggregation coefficients) and also identifies the sets of neighbors that lead to the best overall performance [5, 36]. One of these methods is the Sparse LInear Method for top-$N$ recommendation (SLIM), which was the first method to compute the item-item relations using statistical learning and which has been shown to be one of the best approaches for top-$N$ recommendation [5].

### 3.1.1 SLIM

**SLIM** [5] computes the item-item relations by estimating a sparse aggregation matrix coefficient matrix. As it has been shown to be very well-suited for the top-$N$ recommendation task, different top-$N$ recommendation methods use it as a building block [36, 37, 38, 39].

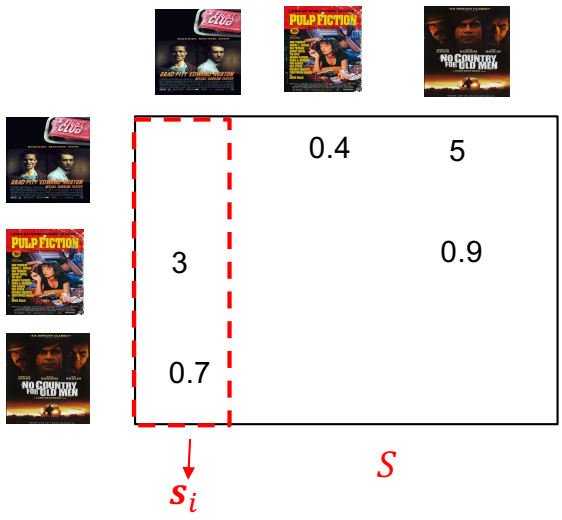



Figure 3.1: A sparse aggregation coefficient matrix $\mathbf{S}$.

SLIM treats the missing entries as zeros and estimates a sparse $m \times m$ aggregation coefficient matrix $\mathbf{S}$. An example matrix $\mathbf{S}$ is shown in Figure 3.1. The recommendation score on an unrated item $i$ for user $u$ is computed as a sparse aggregation of all the user's past rated items:

$$\tilde{r}_{ui} = \mathbf{r}_u^T \mathbf{s}_i, \tag{3.1}$$

where $\mathbf{r}_u^T$ is the row-vector of $\mathbf{R}$ corresponding to user $u$ and $\mathbf{s}_i$ is the $i$th column vector of matrix $\mathbf{S}$, that is estimated by solving the following optimization problem:

$$\begin{aligned} \underset{\mathbf{s}_i}{\text{minimize}} \quad & \tfrac{1}{2}||\mathbf{r}_i - \mathbf{R}\mathbf{s}_i||_2^2 + \tfrac{\beta}{2}||\mathbf{s}_i||_2^2 + \lambda||\mathbf{s}_i||_1, \\ \text{subject to} \quad & \mathbf{s}_i \geq 0, \text{and} \\ & s_{ii} = 0, \end{aligned} \tag{3.2}$$

where $\mathbf{r}_i$ denotes the column-vector of $\mathbf{R}$ containing the feedback associated to the item $i$, $||\mathbf{s}_i||_2^2$ is the $l_2$ norm of $\mathbf{s}_i$ and $||\mathbf{s}_i||_1$ is the entry-wise $l_1$ norm of $\mathbf{s}_i$. The optimization problem of Equation (3.2) is an elastic net problem [40, 41], which means that both $l_2$ and $l_1$ regularizations are used. The $l_1$ regularization gets used so that sparse solutions are found [42]. The $l_2$ regularization prevents over-fitting. The parameters $\beta$ and $\lambda$ are regularization parameters, controlling the strength of the $l_2$ regularization and the $l_1$ regularization, respectively.

The non-negativity constraint is applied so that the vector estimated contains positive coefficients. The $s_{ii} = 0$ constraint makes sure that when computing the weights of an item, that item itself is not used as this would lead to trivial solutions.

All the $\mathbf{s}_i$ vectors can be put together into a matrix $\mathbf{S}$, which can be thought of as an item-item similarity matrix that is learned from the data. So, the model introduced by SLIM can be presented as $\tilde{\mathbf{R}} = \mathbf{RS}$.

In order to create the top-$N$ list for user $u$, we compute $\tilde{r}_{ui}$ for every unrated item $i$, we sort these values and we recommend the $N$ items with the highest estimated ratings.

## 3.2 Latent space top-$N$ recommendation approaches

There are a lot of latent space approaches used for top-$N$ recommendation [6, 43, 44, 45, 46, 47, 48, 49] that have been shown to have good top-$N$ recommendation quality. The latent space approaches perform a low-rank factorization of the user-item feedback matrix into user factors and item-factors that represent user and item characteristics in a common latent space. Among those, the matrix factorization method by Hu et al. [45] and Pan et al. [46] is especially applied for implicit feedback datasets in which the observed and the non-observed entries of the matrix are weighted differently.

### 3.2.1 PureSVD

Among the latent space methods for the top-$N$ recommendation task, a notable one is the **PureSVD** method developed by Cremonesi et al. [6], which performs a truncated Singular Value Decomposition of the matrix $\mathbf{R}$ of rank $f$ to generate the recommendations. In order to do so, the authors proposed to treat the missing entries as zeros. PureSVD is a simple but powerful method, for generating top-$N$ recommendations and demonstrates that treating the missing entries as zero leads to better results than the matrix completion approaches. Specifically, PureSVD estimates the user–item matrix $\mathbf{R}$ by the factorization:

$$\tilde{\mathbf{R}} = \mathbf{P}\boldsymbol{\Sigma}_f\mathbf{Q}^T, \tag{3.3}$$

where $\mathbf{P}$ is an $n \times f$ orthonormal matrix, $\mathbf{Q}$ is an $m \times f$ orthonormal matrix and $\boldsymbol{\Sigma}_f$ is an $f \times f$ diagonal matrix containing the $f$ largest singular values.

## 3.3 Ranking-based top-$N$ recommendation approaches

There is also a class of methods, where the top-$N$ recommendation task has been formulated as a ranking problem [50, 51, 52, 53, 54]. These methods, instead of focusing on estimating the rating for specific unrated items, focus on estimating the rankings of the unrated items. The ranking objective can be used both in context of neighborhood-based and latent space methods.

Among these approaches, a well-known one is Bayesian Personalized Ranking - Matrix Factorization (**BPRMF**) [50]. BPRMF focuses on finding the correct personalized ranking for all items to maximize the posterior probability.

## 3.4 Higher-order methods for top-$N$ recommendation

There have been various works which utilize the concept of frequent itemsets and association rules [55, 56], for recommendation [57, 58, 59, 60].

Higher-order interactions between different features can be modeled with the popular method of Factorization Machines [61]. However, although Factorization Machines can theoretically model high-order feature interactions, typically researchers consider only second order feature interactions due to high complexity, as there is no efficient training algorithm for higher-order factorization machines [62].

The closest method to our thesis work is the one proposed by Deshpande and Karypis [12] which takes into account higher-order relations, beyond pairwise relations between items. The method called Higher Order $k$ Nearest Neighbors (**HOKNN**), incorporates combinations of items (itemsets) in the following way: The most similar items are found not for each individual item, as it is typically done in the neighborhood-based models, but for all possible itemsets up to a particular size $l$.

## 3.5 Local models for top-$N$ recommendation

The idea of using multiple local models is well researched in the literature [63, 64, 65, 66, 67, 68, 69, 70]. Among them, the approaches that are the most relevant to our work are discussed here.

The idea of estimating multiple local models has been proposed in the work by O'connor and Herlocker [67], who performed rating prediction by clustering the rating matrix item-wise and estimating a separate local model for each cluster with nearest neighbor collaborative filtering.

Xu et al. [69] developed a method that co-clusters users and items and estimates a separate local model on each cluster, by applying different collaborative filtering methods; including the item-based neighborhood method. The predicted rating for a user-item pair is the prediction from the subgroup with the largest weight for the user.

Koren [14] proposed a combined model, which estimates every user-item rating $r_{ui}$ as a combination of a global latent space model and local neighborhood interactions.

Weston et al. [47] modeled a user with $T$ latent vectors, each of dimension $m$, to model the user's latent tastes, while every item has a single latent vector of size $m$. In order to compute the prediction for each user and item, they compute the maximum possible score after multiplying each of the $T$ user latent vectors to the item one.

Lee et al. [65, 66] proposed a method called Local Low-Rank Matrix Approximation (**LLORMA**) that relies on the idea that the rating matrix $\mathbf{R}$ is locally low-rank and is approximated as a weighted sum of low-rank matrices. In their method, neighborhoods are identified surrounding different anchor points of user-item pairs, based on a function that measures distances between pairs of users and items and then a local low-rank model is estimated for every neighborhood. The estimation is done in an iterative way where first the latent factors representing the anchor points are estimated and then based on the similarities of the observed entries to the anchor points, the latent factors are re-estimated, until convergence. The predicted rating of a target user-item pair is calculated as a weighted combination of the estimated local models, where the weight is the similarity of the pair to the anchor points. Lee et al. have tested this approach with both a squared error objective [65] and a pairwise ranking objective [66].

LLORMA is the closest approach to the work presented in this thesis in Chapters 6 and 7. However, our work differs from LLORMA in multiple ways: LLORMA considers only local models; while our work also computes a global model and has a personalization factor for each user determining the interplay between the global and the local information. Also, in order to learn better local models, our work considers updating the user subsets for which the local models are estimated (Chapter 6 and Chapter 7),

and also having varying ranks among the local models (Chapter 7), which is not the case for LLORMA. Finally, the way the local models are created is different: the local models in LLORMA are based on anchor points, while in our work they correspond to different user subsets.

# Chapter 4

# Datasets and Evaluation Methodology

## 4.1 Datasets

We used multiple real-world datasets that span the movie, social-bookmarking and point-of-sales domains and one synthetic dataset to evaluate the methods in this thesis. Table 4.1 shows their characteristics. The columns corresponding to #users, #items and #non-zeros show the number of users, number of items and number of non-zeros, respectively, in each dataset. The column corresponding to density shows the density of each dataset (i.e., density=#non-zeros/(#users×#items)).

The *synthetic* dataset is generated by using the IBM synthetic dataset generator [56], which simulates the behavior of customers in a retail environment. The parameters used for generating the dataset are: average size of itemset= 4 and total number of itemsets existent= $1,200$. The *ml100k* dataset corresponds to MovieLens-100K [71], which contains user ratings on different movies. The *bms1* dataset [72] contains several months worth of clickstream data from an e-commerce website. The *delicious* dataset [46] was obtained from the social bookmarking site http://del.ici.ous. In this dataset, the items refer to tags and the non-zeros refer to posts. A non-zero entry indicates that the corresponding user wrote a post using the corresponding tag. The *ctlg3* dataset corresponds to the catalog purchasing transactions of a major mail-order catalog retailer. The *retail* dataset [73] contains the retail market basket data from a Belgian retail store. The

Table 4.1: Dataset characteristics.

| Name | #Users | #Items | #Non-zeros | Density |
|---|---|---|---|---|
| synthetic | 5,000 | 1,000 | 73,597 | 1.47% |
| ml100k | 943 | 1,681 | 100,000 | 6.30% |
| bms1 | 26,667 | 496 | 116,704 | 0.88% |
| delicious | 2,989 | 2,000 | 246,430 | 4.12% |
| ctlg3 | 56,593 | 39,079 | 451,247 | 0.02% |
| retail | 85,146 | 16,470 | 905,560 | 0.06% |
| jester | 57,732 | 150 | 1,760,039 | 20.32% |
| groceries | 63,034 | 15,846 | 2,060,719 | 0.21% |
| bms-pos | 435,319 | 1,657 | 3,286,742 | 0.46% |
| flixster | 29,828 | 10,085 | 7,356,146 | 2.45% |
| ml10m | 69,878 | 10,677 | 10,000,054 | 1.34% |
| netflix | 274,036 | 17,770 | 31,756,784 | 0.65% |

*jester* dataset [74] corresponds to an online joke recommender system and contains ratings that users gave on jokes. The *groceries* dataset corresponds to transactions of a local grocery store. Each user corresponds to a customer and the items correspond to the distinct products purchased over a period of one year. The *bms-pos* dataset [72] contains several years worth of point-of-sales data from a large electronics retailer. The *flixster* dataset is a subset of the original Flixster dataset [75], which consists of movie ratings taken from the corresponding social movie site. The subset was created by keeping the users who have rated more than thirty items and the items that have been rated by at least twenty-five users. The *ml10m* dataset corresponds to the MovieLens 10M dataset [71], and contains ratings that users gave on various movies. The *netflix* dataset is a subset of the original Netflix dataset [21], which contains anonymous movie ratings. The subset was created by keeping the users who have rated between thirty and five hundred items.

Note that some of the datasets originally have ratings, but they were converted to

implicit feedback, by transforming the rated entries to ones and the missing entries to zeros. The existence of a rating (1) indicates that the user purchased/rated the item and its absence (0) that he/she did not. For the different methods presented in this thesis, we use a subset of the listed datasets.

## 4.2 Evaluation methodology

Throughout the thesis, we employ leave-one-out cross-validation [76] to evaluate the performance of the developed and competing methods. For each user, we randomly select an item rated by him/her, and we place it in the test set. The rest of the data comprise the training set.

The reason why we hide only rated items in the test set is because an unrated item leaves doubts as to whether the item would be a good candidate for recommendation to the user.

## 4.3 Performance metrics

There are multiple possible metrics we can use to evaluate the success of a top-$N$ recommendation system and its usefulness to the user. More often than not, identifying the correct metric is a line of research on its own [4, 13], as beyond the accuracy of the recommendation, the novelty and diversity of the recommendation are very important as well [8, 77].

In this thesis, we focus on the accuracy of the top-$N$ recommenders and we follow the metrics used by Deshpande and Karypis [12] and Ning and Karypis [5], when evaluating their top-$N$ recommendation approaches.

Specifically, we measure the performance by considering the number of times the single left-out item is in the top-$N$ recommended items for this user and its position in that list. The quality measures used are the hit-rate (HR) and average-reciprocal hit rank (ARHR).

HR is defined as

$$HR = \frac{\#hits}{\#users}, \tag{4.1}$$

and ARHR is defined as

$$ARHR = \frac{1}{\#users} \sum_{i=1}^{\#hits} \frac{1}{p_i}, \tag{4.2}$$

where "#users" is the total number of users ($n$), and "#hits" is the number of users whose item in the test set is present in the size-$N$ recommendation list. The symbol $p_i$ denotes the position of the item $i$ in the list, which ranges from 1 specifying the top of the list, to $N$ specifying the bottom of the list.

The ARHR is a weighted version of HR, where the position of the test item in the top-$N$ list is taken into account. Both measures have a range from 0 to 1, with 1 being the ideal.

## 4.4 Comparison algorithms

In this thesis, we compare our methods against other competing modern top-$N$ recommendation approaches, that span both the item-item approaches: item $k$-NN [12], HOKNN [12], SLIM [5], and the latent space approaches: PureSVD [6], BPRMF [50] and LLORMA [65]. The details behind these methods are described in Chapter 3.

### Software

For SLIM, we used the SLIM package[1] . For PureSVD, we used the SVDLIBC package[2] , which is based on the SVDPACKC library [78]. For BPRMF,we used the LibRec package [79] and for LLORMA we used the PREA toolkit [80].

### Model selection

We performed an extensive search over the parameter space of the various methods, in order to find the set of parameters that gives us the best performance for all the methods. In the thesis, we only report the performance corresponding to the parameters that lead to the best results.

For item $k$-NN and HOKNN, the number of neighbors $k$ examined lie in the interval $[1 - 50, 60, 70, 80, 90, 100, 200, 300, \ldots, 900, 1000]$. For HOKNN, the support threshold

[1] `www-users.cs.umn.edu/~xning/slim/html`
[2] `https://tedlab.mit.edu/~dr/SVDLIBC/`

$\sigma$ took on values:$\{10, 15, 20, \ldots, 100, 150, 200, \ldots, 950, 1000, 1500, 2000, 2500, 3000\}$.

For SLIM, the $l_1$ and $l_2$ regularization parameters $\lambda$ and $\beta$ were chosen from the set of values: $\{0.0001, 0.001, 0.01, 0.1, 1, 2, 3, 5, 7, 10\}$. The larger the regularization parameters are, the stronger the regularization is.

For PureSVD, the number of singular values $f$ tried lie in the interval:
$\{10, 15, 20, \ldots, 95, 100, 150, 200, \ldots, 1450, 5000\}$.

For BPRMF, the number of factors used in order to get the best results lie in the interval $[1, 10000]$. The values of the learning rate that we tried are: $\{0.0001, 0.001, 0.01, 0.1\}$. The values of the regularization we tried are: $\{0.0001, 0.001, 0.01, 0.1\}$.

Finally, for LLORMA, we followed the parameter methodology of the original paper [65] and we kept fixed the number of iterations $T = 100$, the convergence threshold to $\epsilon = 0.0001$, the number of anchor points to $q = 50$, and used the Epanechnikov kernel with $h1 = h2 = 0.8$. We tried for the regularization values $\lambda_U = \lambda_V$ the values: $\{0.001, 0.01, 0.1\}$. We also tried for the rank of the models the values: $\{1, 2, 3, 5, 7, 10, 15, 20, 25, 30, 35, 40, 45, 50\}$.

As however LLORMA was developed for rating prediction, but we want to use it for top-$N$ recommendation with the evaluation methodology described in Section 4.2, we need to also utilize the unrated items feedback beyond the rated items. It is of very high complexity to introduce in LLORMA all of the unrated items, making it computationally infeasible. Thus, in this thesis, we sample the unrated items for LLORMA. After experimentation, we concluded that sampling for every user ten times the number of unrated items as the number of items he/she has rated gives overall a good approximation of the overall training matrix $\mathbf{R}$.

### Significance testing

When comparing our proposed approaches to the competing methods, we can see some performance differences. We need though a principled way to evaluate how significant the improvement of one approach versus another approach is.

To do so, we perform paired t-tests [81] and we report the performance difference to be statistically significant, if it falls within the 95% confidence interval.

# Chapter 5

# Higher-Order Sparse LInear Method for Top-$N$ Recommendation

This chapter focuses on the development of a top-$N$ recommendation method that revisits the issue of higher-order relations, in the context of modern item-item top-$N$ recommendation methods, as past attempts to incorporate them in the context of classical neighborhood-based methods did not lead to significant improvements in recommendation quality (discussed in Section 3.4). We propose a method called Higher-Order Sparse LInear Method (HOSLIM), which estimates two sparse aggregation coefficient matrices $\mathbf{S}$ and $\mathbf{S}'$ that capture the item-item and itemset-item similarities, respectively. Matrix $\mathbf{S}'$ allows HOSLIM to capture higher-order relations, whose complexity is determined by the length of the itemset. A comprehensive set of experiments is conducted which show that higher-order interactions exist in real datasets and when incorporated in the HOSLIM framework, the recommendations made are improved, in comparison to only using pairwise interactions. Also, the experimental results show that HOSLIM outperforms state-of-the-art item-item recommenders, and the greater the presence of higher-order relations, the more substantial the improvement in recommendation quality is.

## 5.1 Introduction

Item-based methods have been shown to be very well-suited for the top-$N$ recommendation problem [5, 12, 28]. In recent years, the performance of these item-based neighborhood schemes has been significantly improved by using supervised learning methods to learn a model that both captures the similarities and also identifies the sets of neighbors that lead to the best overall performance. One of these methods is SLIM [5] (discussed in Section 3.1), which learns a sparse aggregation coefficient matrix $\mathbf{S}$ from the user-item implicit feedback matrix $\mathbf{R}$, by solving an optimization problem.

However, there is an inherent limitation to both the old and the new top-$N$ recommendation methods, as they capture only pairwise relations between items and they are not capable of capturing higher-order relations. For example, in a grocery store, users tend to often buy items that form the ingredients in recipes. Similarly, the purchase of a phone is often combined with the purchase of a screen protector and a case. In both of these examples, purchasing a subset of items in the set significantly increases the likelihood of purchasing the rest. Ignoring this type of relations, when present, can lead to suboptimal recommendations.

The potential of improving the performance of top-$N$ recommendation methods was recognized by Deshpande et al. [12] (discussed in Section 3.4), who incorporated combinations of items (i.e., itemsets) in their method called HOKNN. The most similar items were identified not only for each individual item, but also for all sufficiently frequent itemsets that are present in the active user's basket. The recommendations were computed by combining itemsets of different size. However, in most datasets this method did not lead to significant improvements. We believe that the reason for this is that the recommendation score of an item was computed simply by an item-item or itemset-item similarity measure, which does not take into account the subtle relations that exist when these individual predictors are combined.

In this chapter, we revisit the issue of utilizing higher-order information, in the context of modern item-item methods. The research question answered is whether the incorporation of higher-order information in the recently developed top-$N$ recommendation methods will improve the recommendation quality further. Our contribution is twofold: First, we verify the existence of higher-order information in real-world datasets,

which suggests that higher-order relations do exist and thus if properly taken into account, they can lead to performance improvements. Second, we develop an approach referred to as Higher-Order Sparse Linear Method (HOSLIM), in which the itemsets capturing the higher-order information are treated as additional items. We conduct a comprehensive set of experiments on different datasets from various applications, which show that HOSLIM improves the recommendation quality on average by 7.86% beyond competing item-item schemes and for datasets with prevalent higher-order information up to 32%. In addition, we present the requirements that need to be satisfied, in order to ensure that HOSLIM computes the predictions in an efficient way.

## 5.2 Proposed approach

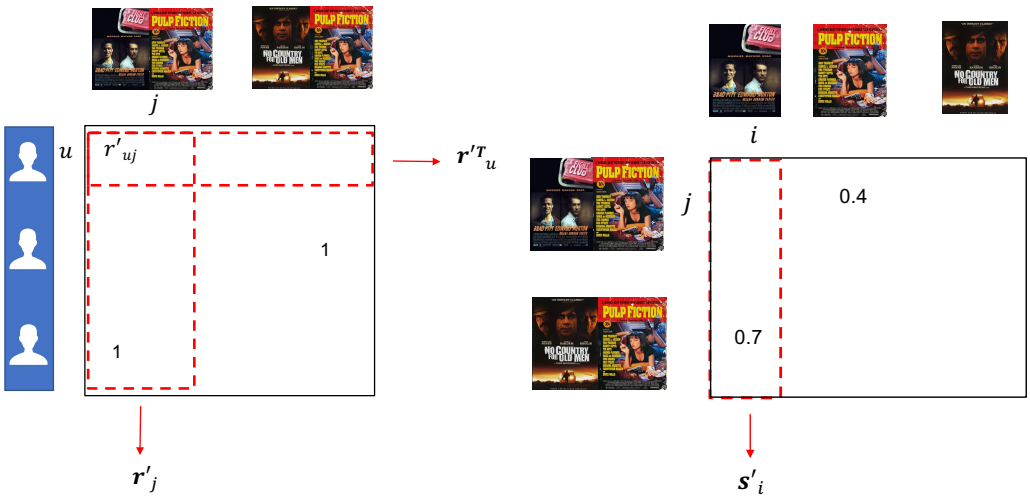
In this chapter, we present our proposed approach HOSLIM for top-$N$ recommendation, which combines the ideas of the higher-order models with the SLIM learning framework, in order to estimate the various item-item and itemset-item similarities.

For the purpose of this chapter, *itemsets* are defined as the sets of items that are co-purchased by at least $\sigma$ users in the user-item implicit feedback matrix $\mathbf{R}$, where $\sigma$ denotes the minimum support threshold [55, 56]. The set of itemsets, denoted by $\mathcal{I}$, has cardinality $p$. We use the notation $j$ to refer to an individual itemset. For the rest of this chapter, every itemset will be frequent and of size two, unless stated otherwise.

### 5.2.1 Overview

In HOSLIM, we first identify the *itemsets* with the use of the method Lpminer by Seno and Karypis [82]. We construct the $n \times p$ *user-itemset implicit feedback matrix* $\mathbf{R}'$, whose columns correspond to the different itemsets in $\mathcal{I}$. An entry $r'_{uj}$ is 1 if user $u$ has purchased *all* the items corresponding to the itemset of the $j$th column of $\mathbf{R}'$, and 0 otherwise.

Then, we estimate the sparse aggregation coefficient matrix $\mathbf{S}$ of size $m \times m$, which captures the item-item similarities and the sparse aggregation coefficient matrix $\mathbf{S}'$, of size $p \times m$ that captures the itemset-item similarities. An example of the matrices $\mathbf{R}'$ and $\mathbf{S}'$ can be shown in Figure 5.1.




(a) $\mathbf{R}'$ (b) $\mathbf{S}'$

Figure 5.1: An example of the HOSLIM matrices $\mathbf{R}'$ and $\mathbf{S}'$.

The predicted score for user $u$ on an unrated item $i$ is computed as a sparse aggregation of both the items purchased and the itemsets that the user's basket supports:

$$\tilde{r}_{ui} = \mathbf{r}_u^T \mathbf{s}_i + \mathbf{r}_u'^T \mathbf{s}'_i, \tag{5.1}$$

where $\mathbf{s}_i$ is a sparse vector of size $m$ corresponding to the $i$th column of $\mathbf{S}$, $\mathbf{s}'_i$ is sparse vector of size $p$ corresponding to the $i$th column of $\mathbf{S}'$, $\mathbf{r}_u^T$ is the $u$th row of $\mathbf{R}$ showing the item implicit feedback of user $u$, and $\mathbf{r}_u'^T$ is the $u$th row of $\mathbf{R}'$ showing the itemset implicit feedback of user $u$.

Finally, top-$N$ recommendation gets done for the $u$th user by computing the scores for all the unpurchased items, sorting them and then taking the top-$N$ values.

### 5.2.2 Estimation of the sparse aggregation coefficient matrices

The sparse matrices $\mathbf{S}$ and $\mathbf{S}'$ encode the similarities (or aggregation coefficients) between the items/itemsets and the items. The $i$th columns of $\mathbf{S}$ and $\mathbf{S}'$ can be estimated by solving the following optimization problem:

**Algorithm 1** HOSLIM

1: Compute the *itemsets* with Lpminer [82].
2: Construct the user-itemset feedback matrix $\mathbf{R}'$ (Section 5.2.1)
3: Estimate the item-item matrix $\mathbf{S}$ and the itemset-item matrix $\mathbf{S}'$, with Equation (5.2).
4: For every user $u$, estimate the predictions on all his unrated items $i$ with Equation (5.1), sort them and recommend the $N$ with the highest values.

$$\begin{aligned} \underset{\mathbf{s}_i, \mathbf{s}'_i}{\text{minimize}} \quad & \tfrac{1}{2}||\mathbf{r}_i - \mathbf{R}\mathbf{s}_i - \mathbf{R}'\mathbf{s}'_i||_2^2 \qquad + \tfrac{\beta}{2}||\mathbf{s}_i||_2^2 + \tfrac{\beta}{2}||\mathbf{s}'_i||_2^2 \\ & \qquad\qquad\qquad\qquad\quad + \lambda||\mathbf{s}_i||_1 + \lambda||\mathbf{s}'_i||_1 \\ \text{subject to} \quad & \mathbf{s}_i \geq 0 \\ & \mathbf{s}'_i \geq 0 \\ & s_{ii} = 0, \text{ and} \\ & s'_{ji} = 0, \text{ where } \{i \in \mathcal{I}_j\}, \end{aligned} \tag{5.2}$$

where $\mathcal{I}_j$ is the set of items that constitute the itemset of the $j$th column of $\mathbf{R}'$, $\mathbf{r}_i$ is the $i$th column of $\mathbf{R}$ containing the feedback of item $i$. The optimization problem of Equation (5.2) is an elastic net regularization problem. It can be solved using coordinate descent and soft thresholding [40].

The constant $\lambda$ is the $l_1$ regularization parameter, which controls the sparsity of the solutions found [42]. The constant $\beta$ is the $l_2$ regularization so that overfitting is prevented.

The non-negativity constraints are applied so that the vectors estimated contain positive coefficients. The constraint $s_{ii} = 0$ makes sure that when computing $r_{ui}$, the element $r_{ui}$ is not used. If this constraint was not enforced, then an item would recommend itself. Following the same logic, the constraint $s'_{ji} = 0$ ensures that the itemsets $j$, for which $i \in \mathcal{I}_j$ will not contribute to the computation of $r_{ui}$.

All the $\mathbf{s}_i$ vectors can be put together into a matrix $\mathbf{S}$, which can be thought of as an item-item similarity matrix that is learned from the data. All the $\mathbf{s}'_i$ vectors can be put together into a matrix $\mathbf{S}'$, which can be thought of as an itemset-item similarity matrix that is learned from the data.

Since the estimation of columns $\mathbf{s}_i$ and $\mathbf{s}'_i$ is independent from the estimation of the

Table 5.1: The average basket size of datasets we evaluated HOSLIM on.

| Name | Average Basket Size |
|---|---|
| groceries | 32.69 |
| synthetic | 14.72 |
| delicious | 82.45 |
| ml100k | 106.04 |
| retail | 10.64 |
| bms-pos | 7.55 |
| bms1 | 4.38 |
| ctlg3 | 7.97 |

rest of the columns, as shown in Equation (5.2), HOSLIM allows for *parallel estimation* of the different columns. This makes HOSLIM scalable and easy to be applied on big datasets, even though more aggregation coefficients are estimated. A continuation of the discussion of the efficiency/scalability of HOSLIM can be found in Section 5.3.2.

Overall, the model introduced by HOSLIM can be presented as $\tilde{\mathbf{R}} = \mathbf{RS} + \mathbf{R}'\mathbf{S}'$. The overview of HOSLIM can be found in Algorithm 1.

## 5.3 Experimental results

The experimental evaluation consists of two parts: First, we analyze various datasets in order to assess the extent to which higher-order relations exist in them. Second, we present the performance of HOSLIM and compare it to competing item-item top-$N$ recommender methods: item $k$-NN and SLIM, as well as the competing baseline HOKNN, which also incorporates itemset information.

Details of the datasets we used can be found in Section 4.1. Also, Table 5.1 presents the average basket size of the datasets we used. The average basket size is the average number of transactions per user. An overview of the competing methods: item $k$-NN, SLIM, and HOKNN can be found in Chapter 3. Also, details on how we ran them (parameters tried and software used) can be found in Section 4.4.

For HOSLIM as well, we performed an extensive search over the parameter space, in order to find the set of parameters that gives us the best performance. We only

Table 5.2: HOSLIM: Coverage by affected users.

| Name | Dependency | | | |
|---|---|---|---|---|
| | max$\geq$ 2 | max$\geq$ 5 | min$\geq$ 2 | min$\geq$ 5 |
| groceries | 95.17 | 88.11 | 97.53 | 96.36 |
| synthetic | 98.04 | 98.00 | 98.06 | 98.06 |
| delicious | 81.33 | 55.34 | 81.80 | 72.57 |
| ml100k | 99.47 | 28.42 | 99.89 | 63.63 |
| retail | 23.54 | 8.85 | 49.70 | 38.48 |
| bms-pos | 59.66 | 32.61 | 66.71 | 51.53 |
| bms1 | 31.52 | 29.47 | 31.55 | 31.54 |
| ctlg3 | 34.95 | 34.94 | 34.95 | 34.95 |

report the performance corresponding to the parameters that lead to the best results. For fairness of comparison with the competing baselines, the values of $\lambda$ and $\beta$ tried were from the same interval as the corresponding values of $\lambda$ and $\beta$ for SLIM: $\{0.0001, 0.001, 0.01, 0.1, 1, 2, 3, 5, 7, 10\}$. Also, the values of the support threshold $\sigma$ tried belonged to the same interval as the values of $\sigma$ for HOKNN:
$\{10, 15, 20, \ldots, 100, 150, 200, \ldots, 950, 1000, 1500, 2000, 2500, 3000\}$.

### 5.3.1 Verifying the existence of higher-order relations

We verified the existence of higher-order relations in the datasets, by measuring how prevalent are the itemsets with strong association between the items that comprise it (beyond pairwise associations). In order to identify such itemsets, (which will be referred to as "good"), we conducted the following experiment:

We found *all frequent itemsets of size* 3 *with* $\sigma$ *equal to* 10. For each of these itemsets we computed two quality metrics.

$$dependency\_max = \frac{P(ABC)}{\max(P(AB)P(C), P(AC)P(B), P(BC)P(A))}, \tag{5.3}$$

and

$$dependency\_min = \frac{P(ABC)}{\min(P(AB)P(C), P(AC)P(B), P(BC)P(A))}, \tag{5.4}$$

Table 5.3: HOSLIM: Coverage by non-zeros.

| Name | Dependency | | | |
|---|---|---|---|---|
| | max$\geq$ 2 | max$\geq$ 5 | min$\geq$ 2 | min$\geq$ 5 |
| groceries | 68.30 | 47.91 | 84.69 | 73.09 |
| synthetic | 76.50 | 75.83 | 76.80 | 76.79 |
| delicious | 59.02 | 22.88 | 59.97 | 44.14 |
| ml100k | 69.77 | 3.75 | 77.94 | 37.62 |
| retail | 13.69 | 4.10 | 40.66 | 25.63 |
| bms-pos | 81.51 | 44.77 | 91.92 | 80.09 |
| bms1 | 63.18 | 60.82 | 63.22 | 63.21 |
| ctlg3 | 24.85 | 24.81 | 24.85 | 24.85 |

where $ABC$ is such an example itemset of size 3 with support $\sigma = 10$ and $AB$, $AC$ and $BC$ are the induced pairs of items.

The metric *dependency_max* measures how much greater the probability of a purchase of all the items of an itemset is than the *maximum* probability of the purchase of an induced pair. The metric *dependency_min* measures how much greater the probability of the purchase of all the items of an itemset is than the *minimum* probability of the purchase of an induced pair.

These metrics are suited for identifying the "good" itemsets, as they discard the itemsets that are frequent just because their induced pairs are frequent. Instead, the above-mentioned metrics discover the frequent itemsets that have all or some infrequent induced pairs, meaning that these itemsets contain higher-order information.

The *dependency_max* is a stricter metric than the *dependency_min*, as an itemset needs to have a greater probability of being rated than all the associated pairs, in order to be identified as carrying higher-order information with the *dependency_max* metric. The *dependency_min* metric has a more relaxed criterion; it also captures the itemsets $ABC$ for which the existence of one pair (e.g., $AB$) increases the probability of the third item (e.g., $C$); but this property does not hold for other pairs.

Given these metrics, we then selected the itemsets of size three that have quality metrics greater than 2 and 5. The higher the quality cut-off, the more certain we are that a specific itemset is "good".

For these sets of high quality itemsets, we analyzed how well they cover the original datasets. We used two metrics of coverage. The first is the percentage of users that have at least one "good" itemset, while the second is the percentage of the non-zeros in the user-item matrix $\mathbf{R}$ covered by at least one "good" itemset. A non-zero in $\mathbf{R}$ is considered to be covered, when the corresponding item of the non-zero value participates in at least one "good" itemset supported by the associated user.

Tables 5.2 and 5.3 show the coverage of the different datasets, in terms of users and non-zeros, respectively. The itemsets considered have a support threshold of 10, except in the case of *delicious* and *ml100k*, where the support threshold is 50, (as *delicious* and *ml100k* are dense datasets and thus a large number of itemsets is induced).

We can see from Tables 5.2 and 5.3 that not all datasets have uniform coverage with respect to high quality itemsets. The *groceries* and *synthetic* datasets contain a large number of "good" itemsets that cover a large fraction of non-zeros in $R$ and nearly all the users. On the other hand, the *ml100k*, *retail* and *ctlg3* datasets contain "good" itemsets that have significantly lower coverage with respect to both coverage metrics. The coverage characteristics of the good itemsets that exist in the remaining datasets is somewhere in between these two extremes.

These results suggest that the potential gains that HOSLIM can achieve will vary across the different datasets and should perform better for the datasets with abundant "good" itemsets, such as *groceries* and *synthetic* datasets. We can observe that the stricter the quality metric and the quality cut-off, the smaller the coverage is with respect to these itemsets.

### 5.3.2 Performance comparison

Table 5.4 shows the performance achieved by HOSLIM, SLIM, $k$-NN and HOKNN. For each method, columns corresponding to the best HR (Equation (4.1)) and the set of parameters with which it is achieved are shown. For $k$-NN (1st order), the parameter used is the number of nearest neighbors (nnbrs). For HOKNN (2nd order), the parameters are the number of nearest neighbors (nnbrs) and the support threshold $\sigma$. For SLIM (1st order), the set of parameters consists of the $l2$ regularization parameter $\beta$ and the $l1$ regularization parameter $\lambda$. For HOSLIM (2nd order), the parameters are $\beta$, $\lambda$ and the support threshold $\sigma$.

Table 5.4: Comparison of 1st order with 2nd order models.

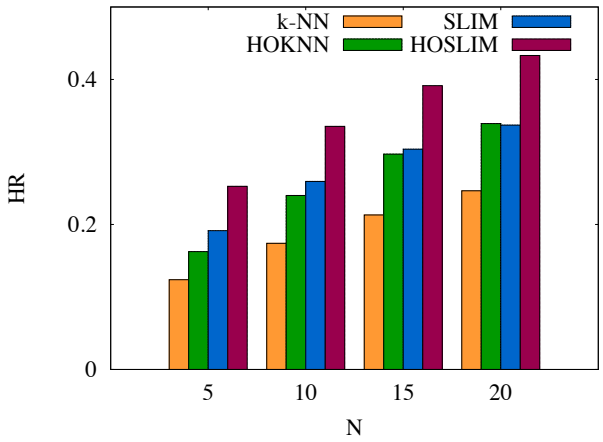
| | $k$-NN models | | | | | SLIM models | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $k$-NN | | HOKNN | | | SLIM | | | HOSLIM | | | |
| Dataset | nnbrs | HR | nnbrs | $\sigma$ | HR | $\beta$ | $\lambda$ | HR | $\sigma$ | $\beta$ | $\lambda$ | HR |
| groceries | 1000 | 0.174 | 800 | 10 | 0.240 | 5 | 0.001 | 0.259 | 10 | 10 | 0.0001 | **0.338** |
| synthetic | 41 | 0.697 | 47 | 10 | 0.769 | 0.1 | 0.1 | 0.733 | 10 | 3 | 1 | **0.860** |
| delicious | 80 | 0.134 | 80 | 10 | 0.134 | 10 | 0.01 | 0.148 | 50 | 10 | 0.01 | **0.156** |
| ml100k | 15 | 0.267 | 15 | 10 | 0.267 | 1 | 5 | 0.338 | 180 | 5 | 0.0001 | **0.349** |
| retail | 1000 | 0.281 | 1,000 | 10 | 0.282 | 10 | 0.0001 | 0.310 | 10 | 10 | 0.1 | **0.317** |
| bms-pos | 700 | 0.478 | 600 | 10 | 0.480 | 7 | 2 | 0.502 | 20 | 10 | 5 | **0.509** |
| bms1 | 200 | 0.571 | 200 | 10 | 0.571 | 15 | 0.01 | 0.588 | 10 | 10 | 0.001 | **0.594** |
| ctlg3 | 700 | 0.559 | 700 | 11 | 0.559 | 5 | 0.1 | 0.581 | 15 | 5 | 0.1 | **0.582** |

The results of Table 5.4 show that HOSLIM produces recommendations that are better than the other methods in all the datasets. We can also see that by comparing first-order with second-order models, that the incorporation of higher-order information can only improve the recommendation quality. This is the case especially in the HOSLIM framework.

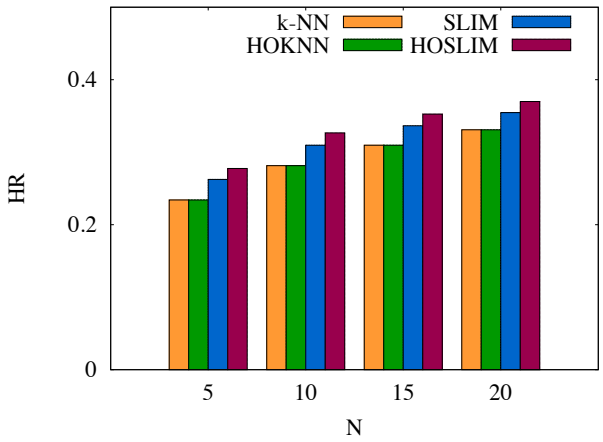
Moreover, we can observe that the greater the existence of higher-order relations in the dataset, the more significant the improvement in recommendation quality is. For example, the greater improvement happens in the *groceries* and the *synthetic* datasets, in which the higher-order relations are the greatest (as seen from Tables 5.2 and 5.3). On the other hand, the *ctlg3* dataset does not benefit from higher-order models, since there are not enough higher-order relations.

These results are to a large extent in agreement with our expectations based on the analysis presented in Section 5.3.1. The datasets for which HOSLIM achieves the highest improvement are those that contain the largest number of users and non-zeros that are covered by high-quality itemsets.

Also, in order to better understand how the existence of "good" itemsets affects the performance of HOSLIM, we computed the correlation coefficient of the percentage improvement of HOSLIM beyond SLIM (presented in Table 5.4) with the product of the affected users coverage and the number of non-zeros coverage (presented in Tables 5.2




(a) Groceries Dataset




(b) Retail Dataset

Figure 5.2: Varying the size of the top-$N$ list for HOSLIM.

and 5.3). The correlation coefficient is 0.712, indicating a strong positive correlation between the coverage (in terms of users and non-zeros) of higher-order itemsets in the dataset and the performance gains achieved by HOSLIM.

### Sensitivity to the size of the top-$N$ list

Figure 5.2 demonstrates the performance of HOSLIM and the competing baselines for different values of $N$: 5, 10, 15 and 20 for the *groceries* and the *retail* datasets. Similar

trends hold for the rest of the datasets as well. The size of the recommendation list $N$ was chosen to be quite small, as a user will not see an item that exists in the bottom of a top-100 or top-200 list.

We can see that HOSLIM outperforms the competing methods for different values of $N$, beyond the default value of 10, which was used in the rest of the chapter. We can also see that as $N$ increases, the performance of the different methods increases as well, as there is higher probability that the hidden item of our test set will be in the top-$N$ list.

**Sensitivity on the support of the itemsets**

Figure 5.3 shows the sensitivity of HOSLIM to the support threshold $\sigma$, for the *groceries* and the *retail* datasets. The trends are the same for the rest of the datasets.

We can see that there is a wide range of support thresholds for which HOSLIM outperforms SLIM. Also, a low support threshold means that more itemsets get utilized and HOSLIM benefits more from the itemsets, leading to better performance.

**Efficient recommendation by controlling the complexity**

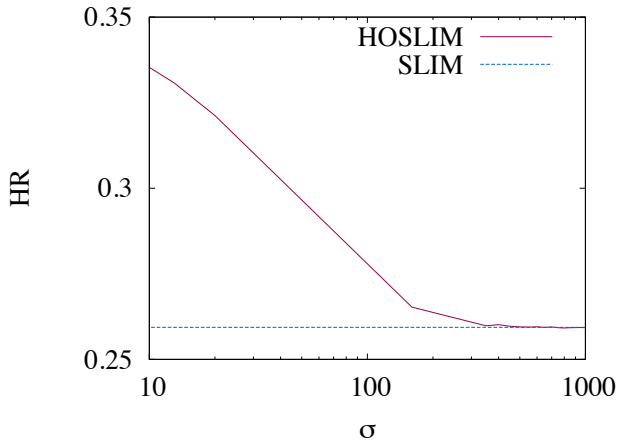
Until this point, the model selected was the one producing the best recommendations, with no further constraints. However, in order for HOSLIM to be used in real-life scenarios, it also needs to be applied fast. In other words, the model should compute the recommendations fast and this means that it should have non-prohibitive complexity.

The question that normally arises is the following: If we find a way to control the complexity, how much will the performance of HOSLIM be affected? In order to answer this question, we did the following experiment: As the cost of computing the top-$N$ recommendation list depends on the number of non-zeros in the model, we selected from all learned models the ones that satisfied the constraint:

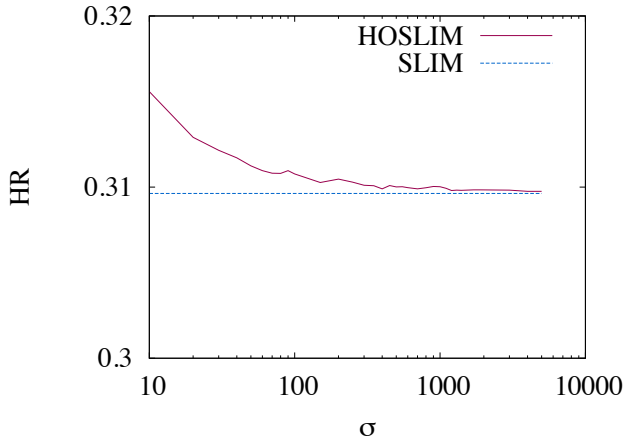
$$nnz(\mathbf{S}') + nnz(\mathbf{S}_{HOSLIM}) \leq 2nnz(\mathbf{S}_{SLIM}). \tag{5.5}$$

With this constraint, we increased the complexity of HOSLIM a little beyond the original SLIM (since the original number of non-zeros is now at most doubled).

Table 5.5 shows the HR achieved by SLIM and constrained and unconstrained HOSLIM. It can be observed that the HR of the constrained HOSLIM model is close to




(a) Groceries Dataset




(b) Retail Dataset

Figure 5.3: Effect of $\sigma$ on the performance of HOSLIM.

the HR of unconstrained HOSLIM, and always better than the HR of SLIM. This means that HOSLIM is applicable in real-world scenarios and can be scaled to big datasets, improving the top-$N$ recommendation quality efficiently.

Table 5.5: Comparison of the HR of constrained HOSLIM with unconstrained HOSLIM and SLIM.

| Dataset | constrained HOSLIM | unconstrained HOSLIM | SLIM |
|---|---|---|---|
| groceries | 0.327 | 0.338 | 0.259 |
| synthetic | 0.860 | 0.860 | 0.733 |
| delicious | 0.154 | 0.156 | 0.148 |
| ml | 0.340 | 0.349 | 0.338 |
| retail | 0.317 | 0.317 | 0.310 |
| bms-pos | 0.509 | 0.509 | 0.502 |
| bms1 | 0.594 | 0.594 | 0.588 |
| ctlg3 | 0.582 | 0.582 | 0.581 |

## 5.4 Conclusion

In this chapter, we revisited the research question of the existence of higher-order information in real-world datasets and whether its incorporation could help the recommendation quality. This was done in the light of modern top-$N$ item-item recommendation methods. The developed approach (HOSLIM) couples the incorporation of higher-order associations (beyond pairwise) with the modern top-$N$ recommendation method SLIM.

The two main take-away messages are that higher-order information exists in different real-world datasets and that its incorporation in modern top-$N$ item-based methods can help the recommendation quality, on average about 7.86% upon competing item-item approaches. Also, when the dataset in question contains abundant higher-order itemsets, the gain can reach up to 32%.

# Chapter 6

# Local Item-Item Models for Top-N Recommendation

Item-based approaches based on SLIM (described in Section 3.1.1) have demonstrated very good performance for top-$N$ recommendation; however they only estimate a single model for all the users. This work is based on the intuition that not all users behave in the same way – instead there exist subsets of like-minded users. By using different item-item models for these user subsets, we can capture differences in their preferences and this can lead to improved performance for top-$N$ recommendations.

In this chapter, we extend SLIM by combining global and local SLIM models. We present a method that computes the prediction scores as a user-specific combination of the predictions derived by a global and local item-item models. We present an approach in which the global model, the local models, their user-specific combination, and the assignment of users to the local models are jointly optimized to improve the top-$N$ recommendation performance. Our experiments show that the proposed method improves upon the standard SLIM model and outperforms competing top-$N$ recommendation approaches, both item-item based and latent space ones.

## 6.1 Introduction

Item-based methods have the drawback of estimating only a single model for all users. In many cases, there are differences in users' behavior, which cannot be captured by a

single model. For example, there could be a pair of items that are extremely similar for a specific user subset, while they have low similarity for another user subset. By using a global model, the similarity between these items will tend to be towards some average value; thus, losing the high correlation of the pair for the first user subset.

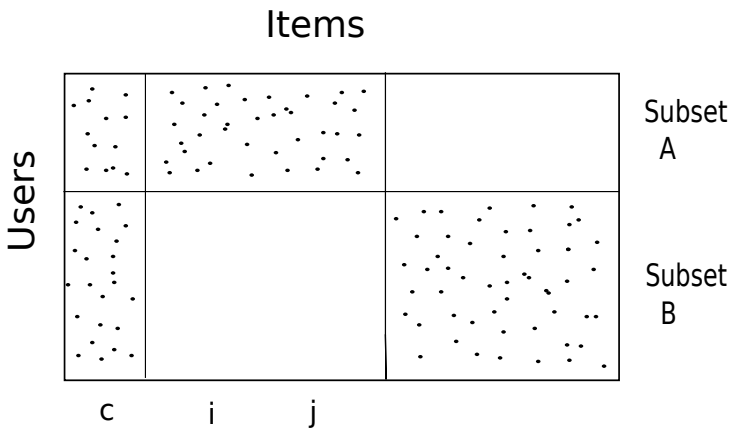
In this chapter we present a top-$N$ recommendation method that extends the SLIM model in order to capture the differences in the preferences between different user subsets. Our method, which we call GLSLIM (Global and Local SLIM), combines global and local SLIM models in a personalized way and automatically identifies the appropriate user subsets. This is done by solving a joint optimization problem that estimates the different item-item models (global and local), their user-specific combination, and the assignment of the users to these models. Our experimental evaluation shows that GLSLIM significantly outperforms competing top-$N$ recommendation methods, reaching up to 17% improvement in recommendation quality.

## 6.2 Proposed approach

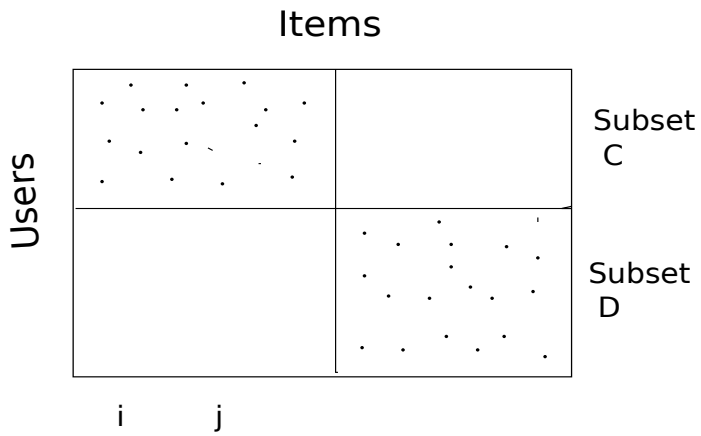
### 6.2.1 Motivation

A global item-item model may not be sufficient to capture the preferences of a set of users, especially when there are user subsets with diverse and sometimes opposing preferences. An example of when local item-item models (item-item models capturing similarities in user subsets) will be beneficial and outperform the item-item model capturing the global similarities is shown in Figure 6.1. It portrays the training matrix $\mathbf{R}$ of two different datasets that both contain two distinct user subsets. Item $i$ is the target item for which we will try to compute predictions. The predictions are computed by using an item-item cosine similarity-based method, in this motivation example.

In the left dataset, (Figure 6.1(a)) there exist some items which have been rated only by the users of one subset, but there is also a set of items which have been rated by users in both subsets. Items $c$ and $i$ will have different similarities when estimated for user-subset A, than when estimated for user-subset B, than for the overall matrix. Specifically, their similarity will be zero for the users of subset B (as item $i$ is not rated by the users of that subset), but it will be non-zero for the users of subset A – and we can further assume without loss of generality that in this example it is high. Then, the




(a) Overlapping rated items between user subsets




(b) No common rated items between user subsets

Figure 6.1: (a) Local item-item models improve upon global item-item model. (b) Global item-item model and local models yield the same results.

similarity between $i$ and $c$ will be of average value when computed in the global case. So, estimating the local item-item similarities for the user subsets of this dataset will help capture the diverse preferences of user-subsets A and B, which would otherwise be missed if we only computed them globally.

However, when using item $j$ to make predictions for item $i$, their similarity will be the same, either globally estimated, either locally for subset A, as they both have been rated only by users of subset A. The same holds for the dataset pictured in Figure 6.1(b),

**Algorithm 2** GLSLIM

Assign $g_u = 0.5$, to every user $u$.
Compute the initial clustering of users.
**while** number of users who switched clusters $> 1\%$ of the total number of users **do**
  Estimate $\mathbf{S}$ and $\mathbf{S}^{p_u}$, $\forall p_u \in \{1, \ldots, k\}$ with Equation (6.2).
  **for all** user $u$ **do**
    **for all** cluster $p_u$ **do**
      Compute $g_u$ for cluster $p_u$ with Equation (6.3).
      Compute the training error.
    **end for**
    Assign user $u$ to the cluster $p_u$ that has the smallest training error and update $g_u$ to the corresponding one for cluster $p_u$.
  **end for**
**end while**

as this dataset consists of user subsets who have no common rated items between them.

Although datasets like the one in Figure 6.1(b) cannot benefit from using local item-item similarity models, datasets such as the one pictured in Figure 6.1(a) can greatly benefit as they can capture item-item similarities, which could be missed in the case of just having a global model.

### 6.2.2 Overview

In this chapter, we present the method GLSLIM, which computes top-$N$ recommendations that utilize user–subset specific models and a global model. These models are jointly optimized along with computing the user assignments for them. We use SLIM for estimating the models. Thus, we estimate a global item-item coefficient matrix $\mathbf{S}$ and also $k$ local item-item coefficient matrices $\mathbf{S}^{p_u}$, where $k$ is the number of user subsets and $p_u \in \{1, \ldots, k\}$ is the index of the user subset, for which we estimate the local matrix $\mathbf{S}^{p_u}$. Every user can belong to one user subset.

The predicted rating of user $u$, who belongs to subset $p_u$, for item $i$ will be estimated

by:

$$\tilde{r}_{ui} = \sum_{l \in \mathcal{R}_u} g_u s_{li} + (1 - g_u) s_{li}^{p_u}. \tag{6.1}$$

The meanings of the various terms are as follows: The term $s_{li}$ shows the global item-item similarity between the $l$th item rated by $u$ and the target item $i$. The term $s_{li}^{p_u}$ depicts the item-item similarity between the $l$th item rated by $u$ and target item $i$, corresponding to the local model of the user-subset $p_u$, to which target user $u$ belongs. Finally, the term $g_u$ is the personalized weight per user, which controls the interplay between the global and the local part. It lies in the interval $[0, 1]$, with 0 showing that the recommendation is affected only by the local model and 1 showing that the user $u$ will use only the global model.

In order to perform top-$N$ recommendation for user $u$, we compute the estimated rating $\tilde{r}_{ui}$ for every unrated item $i$ with Equation (6.1). Then, we sort these values and we recommend the top-$N$ items with the highest ratings to the user.

The estimation of the item-item coefficient matrices, the user assignments and the personalized weight is done with alternating minimization, which will be further explained in the following subsections.

### 6.2.3 Estimating the item-item models

We first separate the users into subsets with either a clustering algorithm (we used CLUTO by Karypis [83]) or randomly. We initially set $g_u$ to be 0.5 for all users, in order to have equal contribution of the global and the local part and we estimate the coefficient matrices $\mathbf{S}$ and $\mathbf{S}^{p_u}$, with $p_u \in \{1, \ldots, k\}$. We use two vectors $\mathbf{g}$ and $\mathbf{g}'$ each of size $n$, where the vector $\mathbf{g}$ contains the personalized weight $g_u$ for every user $u$ and the vector $\mathbf{g}'$ contains the complement of the personalized weight $(1 - g_u)$ for every user $u$. When assigning the users into $k$ subsets, we split the training matrix $\mathbf{R}$ into $k$ training matrices $\mathbf{R}^{p_u}$ of size $n \times m$, with $p_u \in \{1, \ldots, k\}$. Every row $u$ of $\mathbf{R}^{p_u}$ will be the $u$th row of $\mathbf{R}$, if the user $u$ who corresponds to this row belongs in the $p_u$th subset. If the user $u$ does not belong to the $p_u$th subset, then the corresponding row of $\mathbf{R}^{p_u}$ will be empty, without any ratings.

When estimating the local model $\mathbf{S}^{p_u}$, only the corresponding $\mathbf{R}^{p_u}$ will be used. Following SLIM, the item–item coefficient matrices can be calculated per column, which

allows for the different columns (of both the global and the local coefficient matrices) to be estimated *in parallel*. In order to estimate the $i$th column of $\mathbf{S}$ ($\mathbf{s}_i$) and $\mathbf{S}^{p_u}$ ($\mathbf{s}_i^{p_u}$) where $p_u \in \{1, \ldots, k\}$, GLSLIM solves the following optimization problem:

$$\begin{aligned} \underset{\mathbf{s}_i, \{\mathbf{s}_i^1, \ldots, \mathbf{s}_i^k\}}{\text{minimize}} \quad & \tfrac{1}{2}||\mathbf{r}_i - \mathbf{g} \odot \mathbf{R}\mathbf{s}_i - \mathbf{g}' \odot \textstyle\sum_{p_u=1}^{k} \mathbf{R}^{p_u}\mathbf{s}_i^{p_u}||_2^2 + \\ & \tfrac{1}{2}\beta_g||\mathbf{s}_i||_2^2 + \lambda_g||\mathbf{s}_i||_1 + \\ & \textstyle\sum_{p_u=1}^{k} \tfrac{1}{2}\beta_l||\mathbf{s}_i^{p_u}||_2^2 + \lambda_l||\mathbf{s}_i^{p_u}||_1, \\ \text{subject to} \quad & \mathbf{s}_i \geq 0, \\ & \mathbf{s}_i^{p_u} \geq 0, \forall p_u \in \{1, \ldots, k\}, \\ & s_{ii} = 0, \\ & s_{ii}^{p_u} = 0, \forall p_u \in \{1, \ldots, k\}, \end{aligned} \tag{6.2}$$

where $\mathbf{r}_i$ is the $i$th column of $\mathbf{R}$. $\beta_g$ and $\beta_l$ are the $l_2$ regularization weights corresponding to $\mathbf{S}$ and $\mathbf{S}^{p_u}$ $\forall p_u \in \{1, \ldots, k\}$ respectively. Finally $\lambda_g$ and $\lambda_l$ are the $l_1$ regularization weights controlling the sparsity of $\mathbf{S}$ and $\mathbf{S}^{p_u}$ $\forall p_u \in \{1, \ldots, k\}$, respectively.

By having different regularization parameters for the global and the local sparse coefficient matrices, we allow flexibility in the model. In this way, we can control through regularization which of the two components will play a more major part in the recommendation.

The constraint $s_{ii} = 0$ makes sure that when computing $r_{ui}$, the element $r_{ui}$ is not used. If this constraint was not enforced, then an item would recommend itself. For the exact same reason, we enforce the constraint $s_{ii}^{p_u} = 0$, $\forall p_u \in \{1, \ldots, k\}$ for the local sparse coefficient matrices too.

The optimization problem of Equation (6.2) is an elastic net regularization problem and can be solved using coordinate descent and soft thresholding [40].

### 6.2.4 Finding the optimal assignment of users to subsets

After estimating the local models (and the global model), GLSLIM fixes them and proceeds with the second part of the optimization: updating the user subsets. While doing that, GLSLIM also determines the personalized weight $g_u$. We will use the term *refinement* to refer to finding the optimal user assignment to subsets.

Specifically, GLSLIM tries to assign each user $u$ to every possible cluster, while computing the weight $g_u$ that the user would have if assigned to that cluster. Then, for every cluster $p_u$ and user $u$, the training error is computed. The cluster for which this error is the smallest is the cluster to which the user is assigned. If there is no difference in the training error, or if there is no cluster for which the training error is smaller, the user $u$ remains at the initial cluster. The training error is computed for both the user's rated and unrated items.

In order to compute the personalized weight $g_u$, we minimize the squared error of Equation (6.1) for user $u$ who belongs to subset $p_u$, over all items $i$.

By setting the derivative of the squared error to 0, we get:

$$g_u = \frac{\sum_{i=1}^{m} (\sum_{l \in \mathcal{R}_u} s_{li} - \sum_{l \in \mathcal{R}_u} s_{li}^{p_u})(r_{ui} - \sum_{l \in \mathcal{R}_u} s_{li}^{p_u})}{\sum_{i=1}^{m} (\sum_{l \in \mathcal{R}_u} s_{li} - \sum_{l \in \mathcal{R}_u} s_{li}^{p_u})^2}. \tag{6.3}$$

Note that while updating the user subsets, every user is independent of the others, as the models are fixed, thus their new assignment an be computed *in parallel*. The overview of GLSLIM as well as the stopping criterion are shown in Algorithm 2.

## 6.3 Experimental results

In this section we present the results of our experiments. Details of the datasets we used can be found in Section 4.1.

An overview of the competing methods we compare GLSLIM against: PureSVD, BPRMF, and SLIM can be found in Chapter 3. Also, details on how we ran them (parameters tried and software used) can be found in Section 4.4.

As our method contains multiple elements, we want to investigate how each of them impacts the recommendation performance. Thus, beyond GLSLIM, we also investigate the following methods:

- LSLIMr0, which stands for Local SLIM without refinement. In LSLIMr0, a separate item-item model is estimated for each of the $k$ user subsets. No global model is estimated; so there is no personalized weight $g_u$ either. Specifically, the $i$th column of the $p_u$th local model $\mathbf{S}^{p_u}$ ($\mathbf{s}_i^{p_u}$) is estimated by solving the optimization

**Algorithm 3** LSLIM

Compute the initial clustering of users.
**while** number of users who switched clusters $> 1\%$ of the total number of users **do**
  Estimate $\mathbf{S}^{p_u}$, $\forall p_u \in \{1, \ldots, k\}$ with Equation (6.4).
  **for all** user $u$ **do**
    **for all** cluster $p_u$ **do**
      Compute the training error.
    **end for**
    Assign user $u$ to the cluster $p_u$ that has the smallest training error.
  **end for**
**end while**

**Algorithm 4** GLSLIMr0

Assign $g_u = 0.5$, to every user $u$.
Compute the initial clustering of users.
**while** $diff > 0.01\%$ **do**
  Estimate $\mathbf{S}$ and $\mathbf{S}^{p_u}$, $\forall p_u \in \{1, \ldots, k\}$ with Equation (6.2).
  $\forall$ user $u$ compute $g_u$ with Equation (6.3).
  Compute difference in the objective function ($diff$) between subsequent iterations.
**end while**

problem:

$$\begin{aligned} \underset{\{\mathbf{s}_i^1, \ldots, \mathbf{s}_i^k\}}{\text{minimize}} \quad & \tfrac{1}{2}||\mathbf{r}_i - \textstyle\sum_{p_u=1}^{k} \mathbf{R}^{p_u}\mathbf{s}_i^{p_u}||_2^2 + \\ & \textstyle\sum_{p_u=1}^{k} \tfrac{1}{2}\beta_l||\mathbf{s}_i^{p_u}||_2^2 + \lambda_l||\mathbf{s}_i^{p_u}||_1, \end{aligned} \tag{6.4}$$

subject to

$$\begin{aligned} & \mathbf{s}_i^{p_u} \geq 0, \forall p_u \in \{1, \ldots, k\}, \\ & s^{p_u}{}_{ii} = 0, \forall p_u \in \{1, \ldots, k\}, \end{aligned}$$

where the meanings of the different terms are identical to those used in Equation (6.2).

In LSLIMr0, the initial assignment of users to subsets is the one used and never

gets updated. The predicted rating for user $u$, who belongs to subset $p_u$, and item $i$ is estimated by:

$$\tilde{r}_{ui} = \sum_{l \in \mathcal{R}_u} s_{li}^{p_u}. \tag{6.5}$$

The overall recommendation quality is computed as the weighted average of the item-item model performance in every subset.

- LSLIM, which stands for Local SLIM with refinement. In LSLIM, the predicted rating for user $u$ and item $i$ is also estimated by Equation (6.5) and the local models are estimated in the same way as in LSLIMr0. However, the users switch subsets and the local models get updated accordingly, until convergence. The algorithm for LSLIM is shown in Algorithm 3.

- GLSLIMr0, which stands for Global and Local SLIM without refinement. In GLSLIMr0, both a global model and separate item-item local models are estimated along with the per user weight $g_u$. However the assignment of users to subsets remains fixed. The algorithm for this method is shown in Algorithm 4.

For our proposed approaches, we performed an extensive search over the parameter space, in order to find the set of parameters that gives us the best performance. We only report the performance corresponding to the parameters that lead to the best results. The number of clusters examined took on the values: {2, 3, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 60, 70, 80, 90, 100 and 150}. For fairness of comparison, the $l_1$ and $l_2$ regularization parameters were chosen from the same set of values as SLIM: $\{0.0001, 0.001, 0.01, 0.1, 1, 2, 3, 5, 7, 10\}$. The software for all our proposed approaches is available online[1] .

For the clustering of users, we used the CLUTO algorithm [83], as mentioned in Section 6.2.3. For running it, we used the CLUTO toolkit[2] and more specifically the *vcluster* clustering program, with the cosine similarity. All the other parameters were the default ones.

In the rest of this section, the following questions will be answered:

1. How do the proposed methods compare between them?

[1] `https://www-users.cs.umn.edu/~chri2951/code.html`
[2] `http://glaros.dtc.umn.edu/gkhome/cluto/cluto/overview`

2. How does our method compare against competing top-$N$ recommendation methods?

3. What is the time complexity of our method?

### 6.3.1 Performance of the proposed methods

The comparison of our proposed approaches in terms of HR (Equation (4.1)) and ARHR (Equation (4.2)) is shown in Tables 6.1, 6.2, 6.3, 6.4 and 6.5, respectively for each dataset. For each method, the columns correspond to the best HR and ARHR and the parameters for which they are achieved. The parameters are: the number of clusters, the global $l_2$ regularization parameter $\beta_g$, the local $l_2$ regularization parameter $\beta_l$, the global $l_1$ regularization parameter $\lambda_g$ and the local $l_1$ regularization parameter $\lambda_l$. The bold numbers show the best HR/ARHR achieved, per dataset.

Overall, we can see that the general pattern is that GLSLIM is the best-performing method, followed by GLSLIMr0 and LSLIM, while LSLIMr0 is the approach with the lowest performance.

By comparing these methods, we can see the relative benefits provided by the different components of GLSLIM. The comparisons of LSLIMr0 with GLSLIMr0 and also of LSLIM with GLSLIM show the benefit of adding a global model with a personalized weight $g_u$. The comparisons of LSLIMr0 with LSLIM, and also between GLSLIMr0 and GLSLIM demonstrate the benefit of allowing users to switch subsets.

We can see that both these components improve the performance. However, in all of the datasets but *ml10m*, the relative gain of considering a global model beyond the local item-item models and also computing a personalized weight $g_u$ is higher than the gain of allowing users to switch subsets. When all of the components are combined, as in the case of GLSLIM, we get the best performance, both in terms of HR and ARHR.

Table 6.1: Comparison between our proposed approaches for the *groceries* dataset.

| | *groceries* | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Method | Cls | $\beta_g$ | $\beta_l$ | $\lambda_g$ | $\lambda_l$ | HR | Cls | $\beta_g$ | $\beta_l$ | $\lambda_g$ | $\lambda_l$ | ARHR |
| LSLIMr0 | 15 | - | 5 | - | 0.1 | 0.263 | 3 | - | 3 | - | 0.1 | 0.133 |
| LSLIM | 15 | - | 5 | - | 1 | 0.268 | 15 | - | 3 | - | 3 | 0.135 |
| GLSLIMr0 | 3 | 5 | 5 | 1 | 1 | 0.280 | 3 | 5 | 5 | 1 | 1 | 0.144 |
| GLSLIM | 100 | 5 | 5 | 1 | 1 | **0.304** | 100 | 5 | 5 | 1 | 1 | **0.155** |

Table 6.2: Comparison between our proposed approaches for the *ml10m* dataset.

| | *ml10m* | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Method | Cls | $\beta_g$ | $\beta_l$ | $\lambda_g$ | $\lambda_l$ | HR | Cls | $\beta_g$ | $\beta_l$ | $\lambda_g$ | $\lambda_l$ | ARHR |
| LSLIMr0 | 20 | - | 5 | - | 1 | 0.329 | 25 | - | 7 | - | 2 | 0.163 |
| LSLIM | 15 | - | 5 | - | 3 | 0.339 | 15 | - | 7 | - | 3 | 0.167 |
| GLSLIMr0 | 15 | 7 | 3 | 1 | 5 | 0.335 | 15 | 7 | 7 | 1 | 3 | 0.166 |
| GLSLIM | 10 | 10 | 7 | 1 | 1 | **0.345** | 10 | 10 | 7 | 1 | 1 | **0.170** |

Table 6.3: Comparison between our proposed approaches for the *jester* dataset.

| | *jester* | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Method | Cls | $\beta_g$ | $\beta_l$ | $\lambda_g$ | $\lambda_l$ | HR | Cls | $\beta_g$ | $\beta_l$ | $\lambda_g$ | $\lambda_l$ | ARHR |
| LSLIMr0 | 5 | - | 5 | - | 0.1 | 0.898 | 10 | - | 0.1 | - | 0.1 | 0.775 |
| LSLIM | 10 | - | 0.1 | - | 0.1 | 0.916 | 10 | - | 10 | - | 5 | 0.804 |
| GLSLIMr0 | 20 | 7 | 1 | 10 | 1 | 0.929 | 150 | 1 | 1 | 1 | 1 | 0.820 |
| GLSLIM | 10 | 10 | 10 | 10 | 0.1 | **0.940** | 100 | 1 | 1 | 1 | 1 | **0.835** |

**Sensitivity on the number of clusters**

Figure 6.2 shows how the number of clusters affects the HR for GLSLIM and its variants in the *groceries* and *ml10m* datasets. The trends are the same for the rest of the datasets and for the metric ARHR. We can see that GLSLIM outperforms the rest of the methods for all clusters.

Also, we should note that for all datasets GLSLIM can achieve at least 95% of its best performance for only ten clusters, outperforming its closest competing method.

Table 6.4: Comparison between our proposed approaches for the *flixster* dataset.

| | *flixster* | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Method | Cls | $\beta_g$ | $\beta_l$ | $\lambda_g$ | $\lambda_l$ | HR | Cls | $\beta_g$ | $\beta_l$ | $\lambda_g$ | $\lambda_l$ | ARHR |
| LSLIMr0 | 3 | - | 1 | - | 2 | 0.248 | 3 | - | 0.1 | - | 2 | 0.121 |
| LSLIM | 3 | - | 0.1 | - | 3 | 0.250 | 3 | - | 1 | - | 3 | 0.122 |
| GLSLIMr0 | 3 | 1 | 1 | 5 | 5 | 0.254 | 3 | 5 | 5 | 1 | 3 | 0.125 |
| GLSLIM | 3 | 1 | 1 | 1 | 5 | **0.255** | 3 | 1 | 1 | 1 | 5 | **0.126** |

Table 6.5: Comparison between our proposed approaches for the *netflix* dataset.

| | *netflix* | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Method | Cls | $\beta_g$ | $\beta_l$ | $\lambda_g$ | $\lambda_l$ | HR | Cls | $\beta_g$ | $\beta_l$ | $\lambda_g$ | $\lambda_l$ | ARHR |
| LSLIMr0 | 10 | - | 1 | - | 5 | 0.238 | 20 | - | 0.1 | - | 5 | 0.113 |
| LSLIM | 10 | - | 1 | - | 5 | 0.241 | 10 | - | 3 | - | 10 | 0.114 |
| GLSLIMr0 | 20 | 1 | 1 | 5 | 5 | 0.243 | 20 | 1 | 1 | 5 | 5 | 0.115 |
| GLSLIM | 5 | 1 | 1 | 5 | 5 | **0.245** | 5 | 1 | 1 | 5 | 5 | **0.116** |

This is the case even for the datasets where the best performance occurred at a much bigger number of clusters.

### Initializing with random user subsets

The results presented up to this point have been obtained by initializing the user subsets with the user clustering algorithm CLUTO. In order to show that the good performance of GLSLIM is not dependent on the clustering algorithm used, we present the performance when the initialization of the user subsets is random.

In Figure 6.3, we can see for the *ml10m* and *flixster* datasets, the HR achieved across iterations with the two different ways of initialization, for the same regularization and for ten clusters. The same trends hold for ARHR and for different regularizations, clusters and the rest of the datasets.

We can see that the HR of the first iteration with random initialization is lower than the HR of the first iteration when initializing with CLUTO. This is expected, as in the first iteration, only the global and local models are estimated; no personalization nor

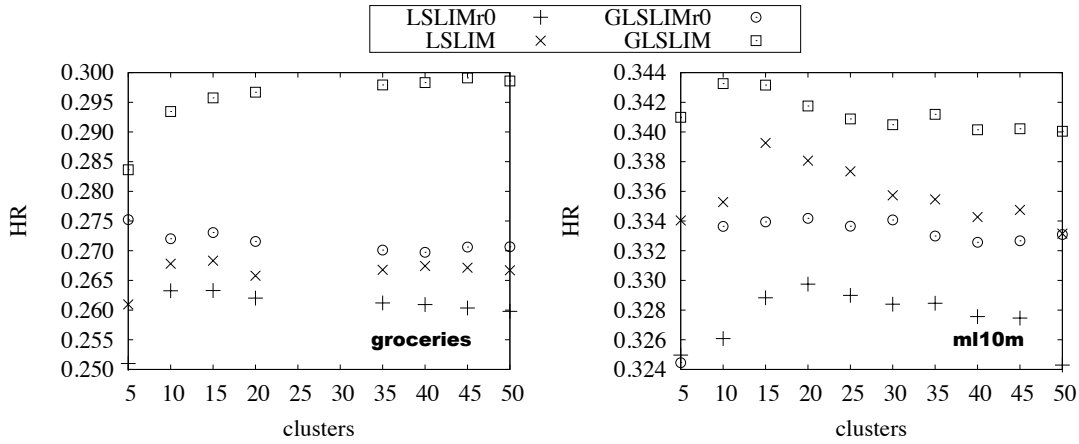



Figure 6.2: The effect of the number of clusters on the performance of GLSLIM.

cluster refinement has been done yet. Thus, the local models estimated from CLUTO are more meaningful than the local models on random user subsets.

As the iterations progress and cluster refinement is done, we see that the HR increases. In the converged state, the final HR achieved is very similar with both initializations. However, when starting from random user subsets, more iterations are needed until convergence. We can then conclude that our method is able to estimate the local models and reach convergence, even with random initialization.

### The interplay between the global and the local part of the model

In order to see how the local models affect the recommendation performance, we look at the $l_1$ norm of the global model $\mathbf{S}$ and the local models $\mathbf{S}^{p_u}$ in the beginning of the algorithm and when the algorithm has converged.

Figure 6.4 shows these $l_1$ norms for 5, 50 and 100 clusters, for the *groceries* and *ml10m* datasets. We can see that the $l_1$ norm of the global model $\mathbf{S}$ is small and it remains small for all possible clusters and throughout the iterations of the algorithm. For the local models $\mathbf{S}^{p_u}$, their $l_1$ norm is larger than the $l_1$ norm of the global model. As the number of clusters increases, the $l_1$ norm of the local models increases. In addition, the $l_1$ norm of the local models in the converged state is larger than the $l_1$ norm of the local models in the beginning. This shows that the effect of local information on the models is major and it becomes greater as the iterations progress and as the number of

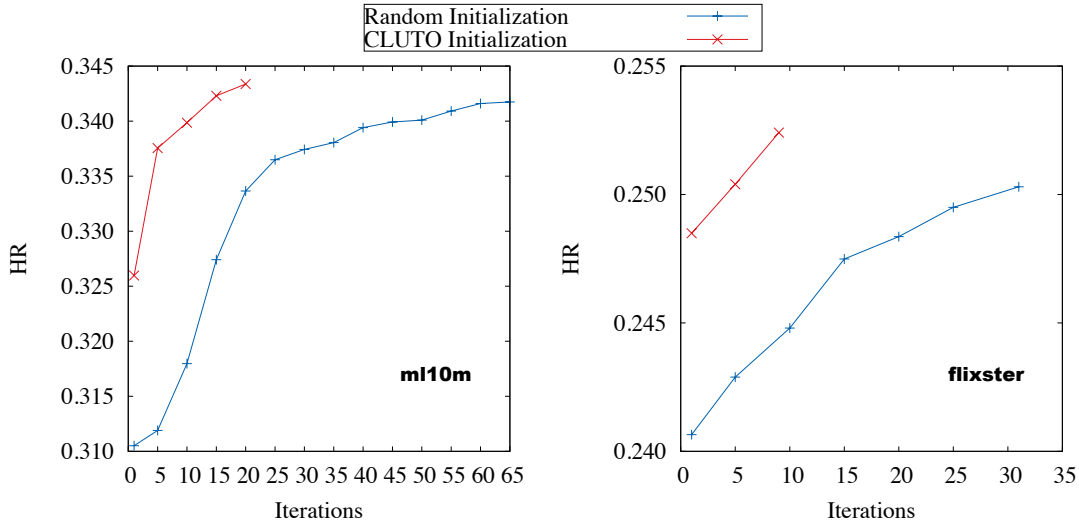



Figure 6.3: Comparing the performance of GLSLIM with CLUTO initialization versus with random initialization of user subsets.

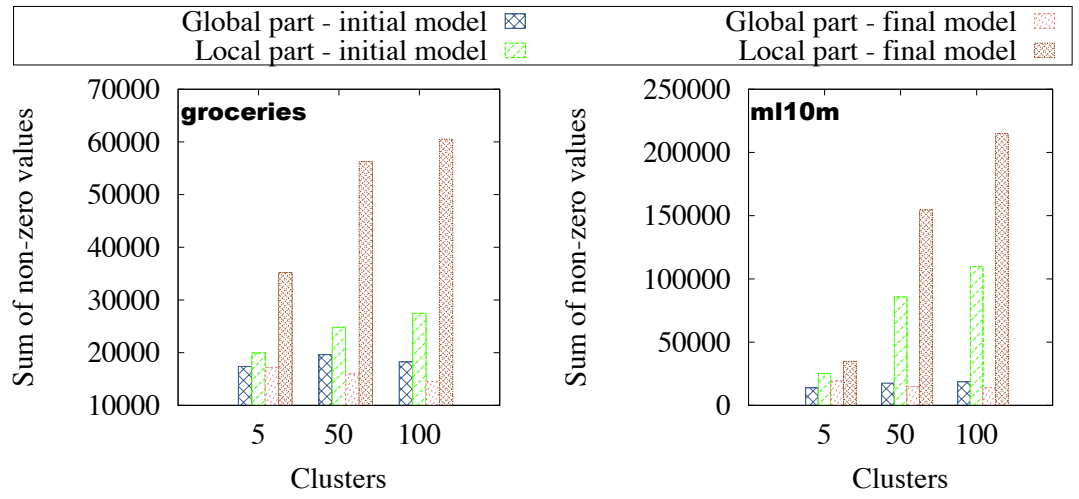



Figure 6.4: How the $l_1$ norm of the global model $\mathbf{S}$ and local models $\mathbf{S}^{p_u}$ changes from the beginning of GLSLIM until convergence.

clusters increases.

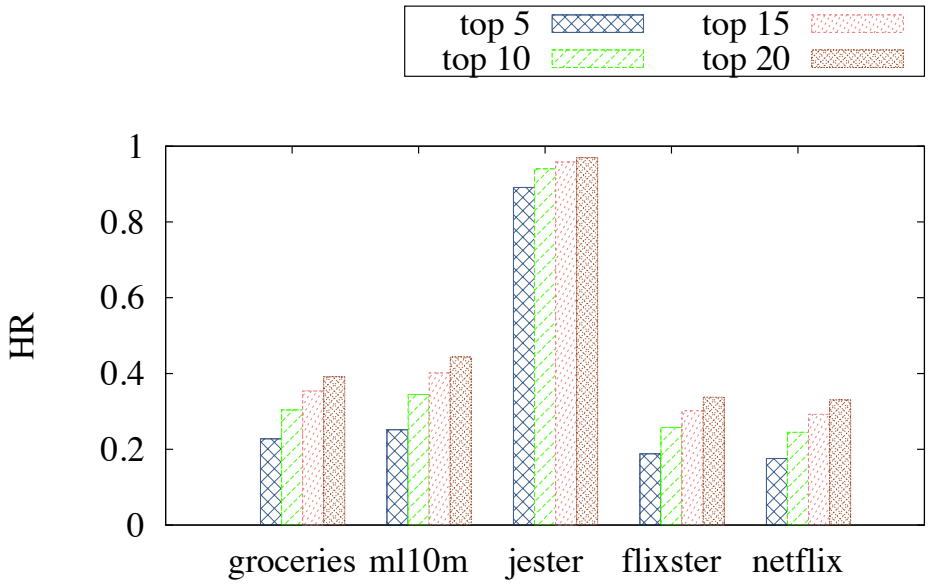



Figure 6.5: Varying the size of the top-$N$ list for GLSLIM.

**Sensitivity to the size of the top-$N$ list**

The results presented throughout the chapter show the performance of our algorithms for a list of size 10. The recommendation list can be of different sizes. In this section, we describe how the performance of our method is affected by using lists of sizes 5, 15 and 20 as well. We choose $N$ to be quite small because users do not look past the very top presented recommendations in a list, anyway.

In Figure 6.5, we can see the HR of GLSLIM, while using the parameters with the best results as presented in Tables 6.1, 6.2, 6.3, 6.4, 6.5, for the different sizes of top-$N$ list.

We can see that as $N$ increases, the performance of our method increases as well, which is expected, as there is higher probability that the hidden item of our test set will be in the top-$N$ list. The impact of the size of the recommendation list $N$ on ARHR is similar to the one shown in Figure 6.5.

Table 6.6: Comparison of GLSLIM with competing approaches in terms of HR.

| | PureSVD | | BPRMF | | | | SLIM | | | GLSLIM | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Dataset | $f$ | HR | factors | lrnrate | reg | HR | $\beta$ | $\lambda$ | HR | Cls | $\beta_g$ | $\beta_l$ | $\lambda_g$ | $\lambda_l$ | HR |
| groceries | 738 | 0.134 | 3000 | 0.01 | 0.001 | 0.214 | 5 | 0.1 | 0.259 | 100 | 5 | 5 | 1 | 1 | **0.304** |
| ml10m | 64 | 0.295 | 5000 | 0.01 | 0.01 | 0.240 | 7 | 5 | 0.312 | 10 | 10 | 7 | 1 | 1 | **0.345** |
| jester | 25 | 0.860 | 300 | 0.01 | 0.01 | 0.903 | 3 | 0.1 | 0.878 | 10 | 10 | 10 | 10 | 0.1 | **0.940** |
| flixster | 90 | 0.194 | 4000 | 0.01 | 0.001 | 0.200 | 0.1 | 2 | 0.242 | 3 | 1 | 1 | 1 | 5 | **0.255** |
| netflix | 50 | 0.204 | 5000 | 0.01 | 0.01 | 0.210 | 0.1 | 5 | 0.231 | 5 | 1 | 1 | 5 | 5 | **0.245** |

### 6.3.2 Performance against competing approaches

Tables 6.6 and 6.7 present the performance of the competing algorithms PureSVD, BPRMF and SLIM versus the performance of our best method, which is GLSLIM, in terms of HR and ARHR, respectively. The above-mentioned tables present the best performance achieved, along with the set of parameters for which they were achieved. For PureSVD the parameter is the number of singular values ($f$). For BPRMF, the parameters are: the number of factors, the learning rate and the regularization. For SLIM, the parameters are the $l2$ regularization parameter $\beta$ and the $l1$ regularization parameter $\lambda$. For GLSLIM, the parameters are the number of clusters, global $\beta$ ($\beta_g$), local $\beta$ ($\beta_l$), global $\lambda$ ($\lambda_g$) and local $\lambda$ ($\lambda_l$). Bold numbers indicate the best HR and ARHR across the different algorithms, for every dataset.

We can see that GLSLIM outperforms all competing approaches for all datasets. Moreover, we checked the statistical significance of this performance increase, following the methodology found in Section 4.4. The improvement of GLSLIM over the best competing baseline (which is SLIM in our case), was shown to be statistically significant in all of the datasets, both in terms of HR and ARHR.

By comparing Tables 6.6 and 6.7 with Tables 6.1, 6.2, 6.3, 6.4 and 6.5, we can also see that LSLIMr0, which is our simplest method, still outperforms the best competing approach, which shows that using multiple item-item models helps top-$N$ recommendation quality.

Table 6.7: Comparison of GLSLIM with competing approaches in terms of ARHR.

| | PureSVD | | BPRMF | | | | SLIM | | | GLSLIM | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Dataset | $f$ | ARHR | factors | lrnrate | reg | ARHR | $\beta$ | $\lambda$ | ARHR | Cls | $\beta_g$ | $\beta_l$ | $\lambda_g$ | $\lambda_l$ | ARHR |
| groceries | 700 | 0.059 | 3100 | 0.01 | 0.001 | 0.099 | 3 | 0.1 | 0.130 | 100 | 5 | 5 | 1 | 1 | **0.155** |
| ml10m | 56 | 0.139 | 7000 | 0.01 | 0.01 | 0.105 | 5 | 2 | 0.151 | 10 | 10 | 7 | 1 | 1 | **0.170** |
| jester | 15 | 0.740 | 100 | 0.01 | 0.01 | 0.766 | 7 | 0.1 | 0.755 | 100 | 1 | 1 | 1 | 1 | **0.835** |
| flixster | 80 | 0.086 | 4000 | 0.01 | 0.001 | 0.089 | 0.1 | 2 | 0.116 | 3 | 1 | 1 | 1 | 5 | **0.126** |
| netflix | 50 | 0.091 | 5000 | 0.01 | 0.01 | 0.100 | 5 | 5 | 0.107 | 5 | 1 | 1 | 5 | 5 | **0.116** |

### 6.3.3 Time complexity

#### Theoretical time complexity

We will use $O(SLIM_i(\mathbf{R}))$ to denote the computational cost of estimating the $i$th column of $\mathbf{S}$. Then, the complexity of estimating the $i$th column of $\mathbf{S}$ and $\mathbf{S}^1, \ldots, \mathbf{S}^k$, is $O(SLIM_i(\mathbf{R})) + O(SLIM_i(\mathbf{R}^1)) + \ldots + O(SLIM_i(\mathbf{R}^k))$, where $\mathbf{R}^1, \ldots, \mathbf{R}^k$ are non-overlapping submatrices of $\mathbf{R}$. Since in order to estimate the $i$th column of $\mathbf{S}$, we need to touch every non-zero in the input matrix $\mathbf{R}$, the complexity $O(SLIM_i(\mathbf{R}))$ is at least linear in the number of non-zeros (nnz). We can then say that the complexity of estimating the $i$th column for the submatrices $\mathbf{R}^1, \ldots, \mathbf{R}^k$ is less than or equal to the complexity of solving it on the matrix $\mathbf{R}$: $O(SLIM_i(\mathbf{R})) \geq O(SLIM_i(\mathbf{R}^1)) + \ldots + O(SLIM_i(\mathbf{R}^k))$. As a result, the complexity of Equation (6.2) is the dominant term $O(SLIM_i(\mathbf{R}))$. Since the regression problem of Equation (6.2) needs to be solved for all $m$ columns (items), the complexity of estimating the global and local models is $O(m \times SLIM_i(\mathbf{R}))$.

The complexity of updating the cluster assignment for each of the $n$ users, after trying to assign them to each of the $k$ clusters (lines $5-11$ of Algorithm 2), is $O(nmk)$, since both the computation of the training error and $g_u$ is $O(m)$. Thus, the per iteration cost of GLSLIM is $O(m(SLIM_i(\mathbf{R}) + nk))$. The number of iterations until GLSLIM converges is typically small, as can be seen in Section 6.3.1.

#### Experimental time complexity

Having seen the theoretical complexity, we will now proceed to investigate GLSLIM experimentally. For this purpose, we use our software, which is available online[3] . The

[3] `https://www-users.cs.umn.edu/~chri2951/code.html`

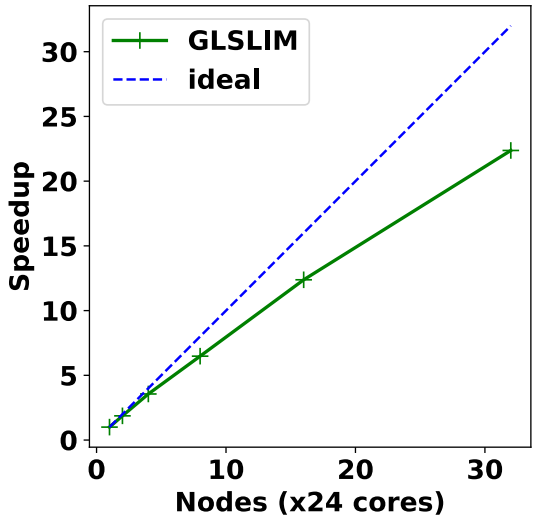



Figure 6.6: The speedup achieved by GLSLIM on the *ml10m* dataset, while increasing the number of nodes.

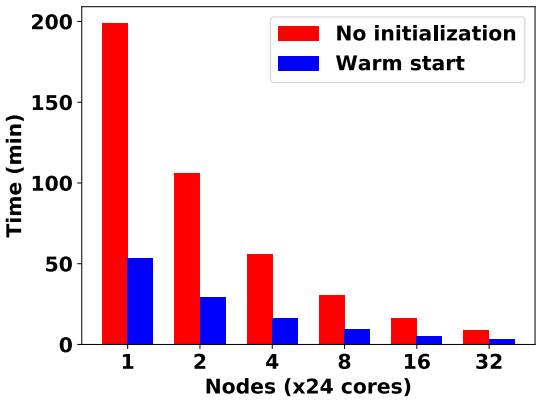



Figure 6.7: The total time in mins achieved by GLSLIM with and without warm start on the *ml10m* dataset, while increasing the number of nodes.

software is MPI-based, taking advantage of the inherent parallelism in terms of items in the model estimation, and in terms of users in the subset refinement. More details on how the parallelism is achieved can be found in Sections 6.2.3 and 6.2.4.

Figure 6.6 shows the speedup achieved by GLSLIM on different nodes, with respect to the time taken by GLSLIM on one node (which consists of 24 cores in our experiments)

for the *ml10m* dataset. The speedup is computed with respect to the time of running GLSLIM on one node. Similar trends hold for the rest of the datasets. The system we conducted the experiments on consists of identical nodes equipped with 62 GB RAM and two twelve-core 2.5 GHz Intel Xeon E5-2680v3 (Haswell) processors. We can see that distributing the computations across multiple nodes can greatly affect the performance of GLSLIM, making it more scalable.

Besides taking advantage of the parallelism, warm start is employed for further improving the efficiency of GLSLIM in the following two ways:

1. The model estimated in every iteration is initialized with the model estimated in the previous iteration (with the exception of the first iteration).

2. When estimating a model with a new choice of parameters, we use another model learned with a different choice of parameters as its initialization.

Figure 6.7 shows the time taken in minutes to run GLSLIM on the *ml10m* dataset, with and without warm start. Similar trends hold for the other datasets, as well. We can see that by using warm start, we can further decrease the required training time.

## 6.4 Conclusion

In this chapter, we proposed a method to improve upon top-$N$ recommendation item-based schemes, by capturing the differences in the preferences between different user subsets, which cannot be captured by a single model.

For this purpose, we estimate a separate local item-item model for every user subset, in addition to the global item-item model. The proposed method allows cluster refinement, in the context of users being able to switch the subset they belong to, which leads to updating the local model estimated for this subset, as well as the global model. The method is personalized, as we compute for all users their own personal weight, defining the degree to which their top-$N$ recommendation list will be affected from global or local information.

Our experimental evaluation shows that our method significantly outperforms competing top-$N$ recommender methods, indicating the value of multiple item-item models.

# Chapter 7

# Local Latent Space Models for Top-$N$ Recommendation

Continuing the same research direction as the previous chapter, this chapter investigates the benefits that multiple local models can bring to latent space methods. Users' behaviors are driven by their preferences across various aspects and latent space approaches model these aspects in the form of latent factors. Though such a user-model has been shown to lead to good results, the aspects that different users care about can vary. In many domains, there may be a set of aspects for which all users care about and a set of aspects that are specific to different subsets of users. To explicitly capture this, we consider models in which there are some latent factors that capture the shared aspects and some user subset specific latent factors that capture the set of aspects that the different subsets of users care about. In particular, we propose two latent space models: rGLSVD and sGLSVD, that combine such a global and user subset specific sets of latent factors. The rGLSVD model assigns the users into different subsets based on their rating patterns and then estimates a global and a set of user subset specific local models whose number of latent dimensions can vary. The sGLSVD model estimates both global and user subset specific local models by keeping the number of latent dimensions the same among these models but optimizes the grouping of the users in order to achieve the best approximation. Our experiments on various real-world datasets show that the proposed approaches significantly outperform state-of-the-art latent space

top-$N$ recommendation approaches.

## 7.1 Introduction

Latent space approaches do not suffer from inefficient personalization, as could be the case with item-item approaches. The reason is that the increase of the rank can easily lead to more latent features estimated for every user. However, they assume that users base their behavior on a set of aspects, shared by all, which they model by estimating a set of global latent factors. We believe that this user model is limiting; we instead propose that a user determines his/her preferences based on some global aspects, shared by all, and on some more specific aspects, that are shared by users that are similar to him/her. For example, a young girl can decide on a piece of clothing to purchase, based on some general aspects, such as whether it is in good condition, and also on some more specific aspects, such as whether this item of clothing is fashionable at the time for girls her age. Thus, we estimate for every user a set of factors capturing the aspects shared by all, and a set of factors capturing the aspects shared by the subset this user belongs to. Estimating such structure with a global latent model could be difficult, since the data at hand are often very sparse.

In this chapter, we propose explicitly encoding such structure, by estimating both a global low-rank model and multiple user subset specific low-rank models. We propose two approaches: rGLSVD (Global and Local Singular Value Decomposition with varying ranks) that considers fixed user subsets but allows for different local models to have varying ranks and sGLSVD (Global and Local Singular Value Decomposition with varying subsets) that allows users to switch subsets, while the local models have fixed ranks. The two approaches explore different ways to learn the local low-rank representations that will achieve the best top-$N$ recommendation quality for the users. The experimental evaluation shows that our approaches outperform competing top-$N$ latent space methods, on average by 13%.

## 7.2 Proposed approach

### 7.2.1 Motivation

Latent space approaches assume that every user's behavior can be described by a set of aspects, which are shared by all the users. However, consider the following scenario. When deciding on which restaurant to go to, people generally tend to agree on a set of aspects that are important: how clean the restaurant is, how delicious the food is. However, there could be other factors which are important to only a subset of users, such as if vegan options are available and if live music exists. Users of a different subset could care about other factors, such as what is the average waiting time, and how big the portions are. We hypothesize that a user model that assumes that users' preferences can be described by some aspects which are common to all but also some additional user subset specific aspects, can better capture user behavior such as the one described above.

As the available data is generally sparse, estimating the global and user subset specific factors from a global low-rank model could be difficult. Thus, we propose to impose such a structure explicitly, by estimating a global latent space model, and multiple user subset specific latent space models.

### 7.2.2 Overview

In this chapter, we present two approaches: Global and Local Singular Value Decomposition with varying ranks (rGLSVD) and Global and Local Singular Value Decomposition with varying subsets (sGLSVD), which estimate a personalized combination of the global and local low-rank models.

Both approaches utilize PureSVD (Section 3.2) as the underlying model, as it has been shown to have good top-$N$ recommendation performance, while being scalable [6, 14].

The rGLSVD approach assigns the users into different subsets based on their rating patterns, which remain fixed, and then estimates a global model and multiple user subset specific local models whose number of latent dimensions can vary.

The sGLSVD model estimates a global model and multiple user subset specific local models by keeping the number of latent dimensions the same among the different local

**Algorithm 5** rGLSVD

Assign $g_u = 0.5$ for every user $u$.
Compute the initial clustering of users.
**while** (users whose $g_u$ changed more than 0.01) $> 1\%$ of the total users **do**
  Construct $\mathbf{R}^g$ and $\mathbf{R}^c$, $\forall c \in \{1, \ldots, k\}$, as discussed in Section 7.2.3.
  Compute a truncated SVD of rank $f^g$ on $\mathbf{R}^g$.
  **for all** cluster $c$ **do**
    Compute a truncated SVD of rank $f^c$ on $\mathbf{R}^c$.
  **end for**
  **for all** user $u$ **do**
    Compute his personalized weight $g_u$ with Equation (7.3).
  **end for**
**end while**

models, but optimizes the grouping of the users in order to achieve the best approximation.

The reason why the two methods are not combined, in other words the reason why we do not allow users to switch subsets between local models with varying ranks, is because most of the users would always move to the subset with the highest corresponding number of local dimensions, causing a lot of them to overfit.

### 7.2.3 Estimation

We now proceed to describe both rGLSVD and sGLSVD, since they follow the same overall estimation methodology. Both approaches use alternating minimization. We will emphasize the points where the approaches differ.

The approaches first estimate the global and user subset specific latent factors. Then, rGLSVD proceeds to estimate the personalized weights, while sGLSVD proceeds to estimate the personalized weights and the user assignments. Then, the global and local latent space models are re-estimated and so on, until convergence.

We initially set the personalized weight $g_u$ controlling the interplay between the global and local low-rank model to be the same and equal to 0.5 for all users, so that the global and local component will have equal contribution. The personalized weight

**Algorithm 6** sGLSVD

```
Assign g_u = 0.5 for every user u.
Compute the initial clustering of users.
while number of users switching clusters > 1% of the total users do
    Construct R^g and R^c, ∀c ∈ {1, ..., k}, as discussed i n Section 7.2.3.
    Compute a truncated SVD of rank f^g on R^g.
    for all cluster c do
        Compute a truncated SVD of the same rank f^c on R^c.
    end for
    for all user u do
        for all cluster c do
            Project user u on cluster c with Equation 7.4.
            Compute his personalized g_u for cluster c with Equation 7.3
            Compute the training error.
        end for
        Assign u to the cluster c with the corresponding smallest training error and
        update his personalized weight g_u to the corresponding one for cluster c.
    end for
end while
```

can take values from 0 to 1, where 0 shows that only local models are utilized, and 1 that only a global model is used.

We construct the global $n \times m$ training matrix $\mathbf{R}^g$ by stacking the vectors $g_u \mathbf{r}_u^T$, for all users $u$. We then compute a truncated singular value decomposition on the global matrix $\mathbf{R}^g$ of rank $f^g$, which allows us to estimate the global user factors, in the following way:

$$\tilde{\mathbf{R}}^g = \mathbf{P} \boldsymbol{\Sigma}_{f^g} \mathbf{Q}^T, \tag{7.1}$$

where $\mathbf{P}$ is an $n \times f^g$ orthonormal matrix showing the global user factors, $\mathbf{Q}$ is an $m \times f^g$ orthonormal matrix showing the global item factors, and $\boldsymbol{\Sigma}_{f^g}$ is an $f^g \times f^g$ diagonal matrix containing the $f^g$ largest singular values.

Then, we separate the users into $k$ subsets with a clustering algorithm (we use CLUTO by Karypis [83]). Every user can belong to one subset. For every subset

$c \in \{1, \ldots, k\}$, we construct the corresponding local training matrix $\mathbf{R}^c$ by stacking the vectors $(1 - g_u)\mathbf{r}_u^T$, for all users $u$ belonging to subset $c$. So, every matrix $\mathbf{R}^c$ has $m$ columns and as many rows as the number of users belonging to subset $c$, which we note as $n^c$. For every subset $c$, we compute a truncated singular value decomposition on $\mathbf{R}^c$, of rank $f^c$:

$$\tilde{\mathbf{R}}^c = \mathbf{P}^c \boldsymbol{\Sigma}_{f^c} \mathbf{Q}^{cT}, \tag{7.2}$$

where $\mathbf{P}^c$ is a $n^c \times f^c$ matrix containing the local user factors which are specific to subset $c$, and $\mathbf{Q}^c$ is a $m \times f^c$ matrix containing the local item factors of subset $c$. Note that in rGLSVD, the ranks $f^c$ can be different for each local subset $c$. Instead, the ranks $f^c$ are the same across the local subsets $c$ in sGLSVD.

So, we estimate a global user latent factor matrix $\mathbf{P}$, a global item latent factor matrix $\mathbf{Q}$, $k$ user subset specific user latent factor matrices $\mathbf{P}^c$ and $k$ user subset specific item latent factor matrices $\mathbf{Q}^c$.

Then, we proceed to the step of updating the personalized weights for rGLSVD or to the step of updating the personalized weights with the user assignments for sGLSVD.

We compute the personalized weight $g_u$, $\forall u$ by minimizing the squared error for every user $u$ over all items (both rated and unrated ones). After setting the derivative of the squared error to 0, we get:

$$g_u = \frac{\sum_{i=1}^{m} (a - b)(r_{ui} - b)}{\sum_{i=1}^{m} (a - b)^2}, \tag{7.3}$$

where $a = \frac{1}{g_u} \mathbf{p}_u^T \boldsymbol{\Sigma}_{f^g} \mathbf{q}_i$ and $b = \frac{1}{1-g_u} \mathbf{p}_u^{cT} \boldsymbol{\Sigma}_{f^c} \mathbf{q}_i^c$.

The method sGLSVD updates the user subsets, in the following way: We try to assign each user $u$ to every possible cluster $c$, while computing the weight $g_u$ that the user would have if assigned to that cluster, with Equation (7.3). After every possible such assignment, we compute the training error for user $u$, and we assign him/her to the cluster that produced the smallest training error. In order to compute the training error for user $u$, who is trying to be assigned to a new subset $c$ he/she did not belong to before, we need to *project* him/her to the new subset $c$, by learning his/her projected user latent factor:

$$\mathbf{p}_u^{cT} = \mathbf{r}_u^T \mathbf{Q}^c {\boldsymbol{\Sigma}_{f^c}}^{-1}. \tag{7.4}$$

An overview of rGLSVD along with the stopping criterion is shown in Algorithm 5.

**Algorithm 7** rLSVD

1: Compute the initial clustering of users.
2: **for all** cluster $c$ **do**
3: Construct $\mathbf{R}^c$, as discussed in Section 7.3.
4: Compute a truncated SVD of rank $f^c$ on $\mathbf{R}^c$.
5: **end for**

An overview of sGLSVD along with the stopping criterion can be found in Algorithm 6.

When the user and item latent factors are fixed, we can estimate the personalized weights of the users for rGLSVD and the personalized weights and user assignments for sGLSVD *in parallel*.

### 7.2.4 Prediction and recommendation

The predicted rating of user $u$, who belongs to subset $c$, for item $i$ is a combination of the global model and the local model of subset $c$:

$$\tilde{r}_{ui} = \mathbf{p}_u^T \boldsymbol{\Sigma}_{f^g} \mathbf{q}_i + \mathbf{p}_u^{cT} \boldsymbol{\Sigma}_{f^c} \mathbf{q}_i^c, \tag{7.5}$$

where $\mathbf{p}_u^T$ is the $u$th row of $\mathbf{P}$ corresponding to user $u$, $\mathbf{q}_i$ is the $i$th column of $\mathbf{Q}^T$ corresponding to item $i$, $\mathbf{p}_u^{cT}$ is the $u$th row of $\mathbf{P}^c$ and $\mathbf{q}_i^c$ is the $i$th column of $\mathbf{Q}^{cT}$. Note that the personalized weights $g_u$ and $1 - g_u$ are enclosed inside the user latent factors $\mathbf{p}_u^T$ and $\mathbf{p}_u^{cT}$ correspondingly.

In order to compute the top-$N$ recommendation list for user $u$, we estimate the predicted rating $\tilde{r}_{ui}$ with Equation (7.5) for all his unrated items $i$, we sort their values in a descending order, and we recommend the $N$ items with the highest corresponding values.

## 7.3 Experimental results

In this section, we present the results of the experimental evaluation of rGLSVD and sGLSVD on a variety of real-world datasets. Details of the datasets we used can be found in Section 4.1. An overview of the competing methods: PureSVD, BPRMF and

**Algorithm 8** sLSVD

```
Compute the initial clustering of users.
while number of users switching clusters > 1% of the total users do
  for all cluster c do
    Construct R^c, as discussed in Section 7.3.
    Compute a truncated SVD of the same rank f^c on R^c.
  end for
  for all user u do
    for all cluster c do
      Project user u on cluster c with Equation 7.4.
      Compute the training error.
    end for
    Assign u to the cluster c with the corresponding smallest training error.
  end for
end while
```

LLORMA can be found in Chapter 3. Also, details on how we ran them (parameters tried and software used) can be found in Section 4.4.

As rGLSVD and sGLSVD estimate multiple components, we propose different variants, to investigate the effect of each component on the top-$N$ recommendation quality:

- **LSVD**, which stands for Local Singular Value Decomposition: We estimate multiple local latent space models of constant rank $f^c$. The user subsets remain fixed.
- **GLSVD**, which stands for Global and Local Singular Value Decomposition: We estimate a global latent space model along with multiple local latent space models of constant rank $f^c$. The user subsets are fixed.
- **rLSVD**, which stands for Local Singular Value Decomposition with varying ranks: We estimate multiple latent space models of varying ranks. There is no global model, and the users remain in their original predefined subsets. We compute the predicted rating of user $u$, who belongs to subset $c$, for item $i$ as:

$$\tilde{r}_{ui} = \mathbf{p}_u^{cT} \boldsymbol{\Sigma}_{f^c} \mathbf{q}_i^c. \tag{7.6}$$

After separating the users into $k$ subsets, we construct the corresponding local training matrices $\mathbf{R}^c$ $\forall c \in \{1, \ldots, k\}$ by stacking the vectors $\mathbf{r}_u^T$, for all users $u$ belonging to subset $c$. We then perform truncated singular value decompositions of varying ranks $f^c$ on each matrix $\mathbf{R}^c$. An overview of rLSVD can be found in Algorithm 7.

- **sLSVD**, which stands for Local Singular Value Decomposition with varying subsets: We estimate multiple latent space models of the same rank; however every user can switch to the subset $c$, which provides the low-rank representation of $u$ with the smallest training error. There is no global model. We also compute the predicted ratings with Equation (7.6). An overview of sLSVD can be found in Algorithm 8.

For our proposed approaches, we performed an extensive search over the parameter space, in order to find the set of parameters that gives us the best performance. We only report the performance corresponding to the parameters that lead to the best results. The number of clusters examined took on the values: $\{2, 3, 5, 10, 15, \ldots, 90, 95, 100\}$. The rank of the local models $f^c$ was varied among the values:
$\{1, 2, 3, 5, 10, 15, \ldots, 90, 95, 100\}$. We did not conduct parameter search on the rank of the global model $f^g$, instead we fixed it to the value $f$ shown to provide the best results in PureSVD.

In the rest of the section, the following questions will be answered:

1. How do the proposed approaches compare against each other?
2. How does our method compare against competing top-$N$ recommendation methods?

### 7.3.1 Performance of the proposed methods

Tables 7.1, 7.2, 7.3, 7.4 and 7.5 show the performance of our proposed approaches in terms of HR (Equation (4.1)) and ARHR (Equation (4.2)), respectively for every dataset, along with the set of parameters for which this performance was achieved. The parameters are: the number of user subsets/clusters (Cls), the rank of the global

model ($f^g$), and the ranks of the local models ($f^c$). The bold numbers show the best HR/ARHR achieved, per dataset.

We can see that the overall best performing methods are the proposed methods: rGLSVD and sGLSVD. We can also see that we can achieve the best low-rank representation in some datasets by varying the rank of local models (rGLSVD), and in others by allowing users to switch subsets, while having local models of fixed rank (sGLSVD). We can reach the same conclusion from the pairwise comparison of sLSVD and rLSVD. This shows the merit of both ways to reach the best local low-rank representation.

We can also observe that the global component improves the recommendation quality, by performing a pairwise comparison of LSVD with GLSVD, sLSVD with sGLSVD, and rLSVD with rGLSVD. After performing paired t-tests, the difference in their performance was shown to be statistically significant, with 95% confidence.

Finally, we can see that rLSVD and sLSVD outperform LSVD, both in terms of HR and ARHR, as LSVD is a simpler method than rLSVD and sLSVD: rLSVD with constant rank $f^c$ results in LSVD and sLSVD with fixed user subsets results in LSVD. Also, rGLSVD and sGLSVD outperform GLSVD, which is also expected as GLSVD results from sGLSVD with fixed user subsets, or rGLSVD with constant ranks $f^c$.

We do not show the rank of each local model $f^c$ that leads to the best performance of rLSVD and rGLSVD in Tables 7.1, 7.2, 7.3, 7.4 and 7.5 for space reasons, but we present it instead here. We will use the following notation scheme: $\{c_1 : f_1^c, c_2 : f_2^c, \ldots\}$, where $c_1$ shows how many clusters have local rank $f_1^c$, $c_2$ shows how many clusters have local rank $f_2^c$ etc. The sum of $c_1$, $c_2$, ... equals the total number of user subsets.

The ranks $f^c$ that correspond to the best rLSVD results in terms of HR are: $\{25 : 5, 42 : 10, 10 : 15, 6 : 20, 2 : 25, 2 : 30, 8 : 40, 2 : 50, 1 : 65, 1 : 85, 1 : 90\}$ for the *groceries* dataset, $\{1 : 2, 2 : 3, 4 : 5, 4 : 10, 1 : 15, 2 : 20, 1 : 25\}$ for the *ml10m* dataset, $\{1 : 1, 1 : 5, 3 : 10\}$ for the *jester* dataset, $\{3 : 5, 5 : 10, 2 : 20\}$ for the *flixster* dataset, and $\{45 : 5, 42 : 10, 3 : 15\}$ for the *netflix* dataset.

The ranks $f^c$ that correspond to the best rLSVD results in terms of ARHR are: $\{44 : 5, 35 : 10, 14 : 15, 7 : 20\}$ for the *groceries* dataset, $\{5 : 5, 7 : 10, 5 : 15, 1 : 20, 1 : 25, 1 : 30\}$ for the *ml10m* dataset, $\{2 : 1, 2 : 5, 1 : 10\}$ for the *jester* dataset, $\{5 : 5, 3 : 10, 2 : 20\}$ for the *flixster* dataset, and $\{51 : 5, 37 : 10, 2 : 15\}$ for the *netflix* dataset.

Table 7.1: Comparison between our proposed approaches for the *groceries* dataset.

| | *groceries* | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Method | Cls | $f^g$ | $f^c$ | HR | Cls | $f^g$ | $f^c$ | ARHR |
| LSVD | 100 | - | 20 | 0.192 | 100 | - | 15 | 0.091 |
| sLSVD | 100 | - | 25 | 0.271 | 100 | - | 15 | 0.130 |
| rLSVD | 100 | - | | 0.210 | 100 | - | | 0.105 |
| GLSVD | 100 | 25 | 25 | 0.204 | 100 | 20 | 20 | 0.100 |
| sGLSVD | 100 | 25 | 25 | **0.283** | 100 | 20 | 20 | **0.136** |
| rGLSVD | 90 | 30 | | 0.216 | 90 | 30 | | 0.105 |

The ranks $f^c$ that correspond to the best rGLSVD results in terms of HR are: $\{16:10, 13:15, 15:20, 7:25, 5:30, 7:35, 4:40, 4:45, 2:50, 4:55, 1:60, 2:65, 2:70, 3:80, 2:85, 2:95, 1:100\}$ for the *groceries* dataset, $\{6:10, 4:15, 1:20, 2:25, 2:30, 3:40, 1:45, 1:55, 1:60, 2:65, 1:80, 2:85, 1:90, 1:95, 2:100\}$ for the *ml10m* dataset, $\{2:2, 1:3, 1:5, 1:10, 2:15, 1:20, 2:35\}$ for the *jester* dataset, $\{1:25, 1:40, 1:50, 2:55\}$ for the *flixster* dataset, $\{2:10, 10:15, 11:20, 9:25, 9:30, 9:35, 8:40, 6:45, 3:55, 1:60, 1:75, 1:95\}$ for the *netflix* dataset.

The ranks $f^c$ that correspond to the best rGLSVD in terms of ARHR are: $\{28:10, 14:15, 13:20, 18:25, 1:30, 5:35, 3:40, 1:45, 1:50, 2:55, 1:60, 1:65, 1:85, 1:90\}$ for the *groceries* dataset, $\{9:10, 4:15, 2:20, 7:25, 1:30, 1:35, 2:45, 3:50, 1:55, 1:60, 1:65, 1:75, 1:80, 2:85, 2:90, 2:100\}$ for the *ml10m* dataset, $\{1:1, 1:10, 1:35\}$ for the *jester* dataset, $\{2:25, 2:35, 2:45, 1:55, 1:65, 1:80, 1:90\}$ for the *flixster* dataset, and $\{2:10, 11:15, 17:20, 21:30, 14:35, 7:40, 9:45, 1:50, 1:60, 2:100\}$ for the *netflix* dataset.

**Sensitivity on the number of user subsets**

We can see the above observations more clearly in Figure 7.1, which shows the performance of the proposed methods when varying the number of clusters, in terms of ARHR for the *ml10m* dataset. The trends are similar for HR and for the rest of the datasets.

We can see for a wide range of user subsets, and not for just a specific choice, that: (i) rGLSVD and sGLSVD outperform the rest of the approaches, (ii) estimating a global

Table 7.2: Comparison between our proposed approaches for the *ml10m* dataset.

| | *ml10m* | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Method | Cls | $f^g$ | $f^c$ | HR | Cls | $f^g$ | $f^c$ | ARHR |
| LSVD | 25 | - | 20 | 0.300 | 25 | - | 20 | 0.142 |
| sLSVD | 50 | - | 15 | 0.311 | 55 | - | 15 | 0.146 |
| rLSVD | 15 | - | | 0.317 | 20 | - | | 0.150 |
| GLSVD | 75 | 65 | 10 | 0.311 | 35 | 65 | 15 | 0.149 |
| sGLSVD | 85 | 55 | 10 | 0.320 | 45 | 55 | 15 | 0.152 |
| rGLSVD | 30 | 65 | | **0.321** | 40 | 65 | | **0.154** |

Table 7.3: Comparison between our proposed approaches for the *jester* dataset.

| | *jester* | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Method | Cls | $f^g$ | $f^c$ | HR | Cls | $f^g$ | $f^c$ | ARHR |
| LSVD | 5 | - | 3 | 0.816 | 5 | - | 1 | 0.693 |
| sLSVD | 2 | - | 3 | 0.816 | 3 | - | 2 | 0.697 |
| rLSVD | 5 | - | | 0.895 | 5 | - | | 0.772 |
| GLSVD | 2 | 25 | 2 | 0.863 | 3 | 15 | 1 | 0.746 |
| sGLSVD | 2 | 25 | 2 | 0.865 | 5 | 15 | 1 | 0.746 |
| rGLSVD | 10 | 15 | | **0.910** | 3 | 15 | | **0.783** |

Table 7.4: Comparison between our proposed approaches for the *flixster* dataset.

| | *flixster* | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Method | Cls | $f^g$ | $f^c$ | HR | Cls | $f^g$ | $f^c$ | ARHR |
| LSVD | 5 | - | 40 | 0.202 | 5 | - | 50 | 0.091 |
| sLSVD | 15 | - | 30 | 0.207 | 15 | - | 30 | 0.096 |
| rLSVD | 10 | - | | 0.207 | 10 | - | | 0.095 |
| GLSVD | 3 | 80 | 40 | 0.214 | 3 | 80 | 40 | 0.099 |
| sGLSVD | 25 | 80 | 30 | **0.218** | 25 | 80 | 35 | **0.102** |
| rGLSVD | 5 | 80 | | 0.217 | 10 | 80 | | 0.101 |

model beyond local models helps the performance, and (iii) allowing users to switch subsets or estimating multiple local models with varying ranks allows for estimation of

Table 7.5: Comparison between our proposed approaches for the *netflix* dataset.

| | *netflix* | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Method | Cls | $f^g$ | $f^c$ | HR | Cls | $f^g$ | $f^c$ | ARHR |
| LSVD | 90 | - | 20 | 0.211 | 90 | - | 20 | 0.097 |
| sLSVD | 65 | - | 20 | 0.215 | 95 | - | 20 | 0.100 |
| rLSVD | 90 | - | | 0.216 | 90 | - | | 0.099 |
| GLSVD | 65 | 50 | 25 | 0.219 | 65 | 50 | 20 | 0.101 |
| sGLSVD | 100 | 50 | 20 | **0.225** | 100 | 50 | 20 | **0.105** |
| rGLSVD | 70 | 50 | | 0.223 | 85 | 50 | | 0.104 |

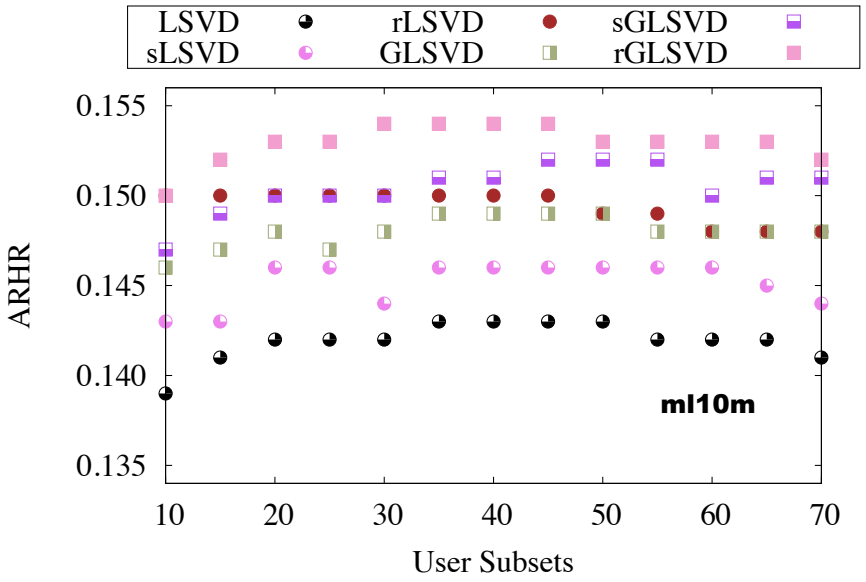



Figure 7.1: The performance of the proposed methods: LSVD, sLSVD, rLSVD, GLSVD, sGLSVD, and rGLSVD when varying the number of user subsets, in terms of ARHR for the *ml10m* dataset.

better low-rank representations than the ones estimated with constant local ranks and fixed user subsets.

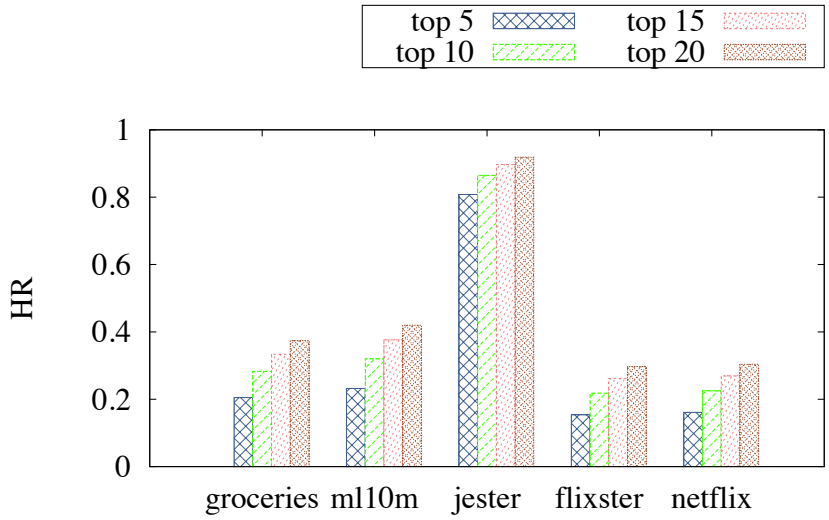



Figure 7.2: The performance of sGLSVD in terms of HR for different sizes $N$ of the recommendation list.

#### Varying the size $N$ of the recommendation list

We obtained the results shown until this point by using a recommendation list of size $N = 10$. Figure 7.2 shows the performance of sGLSVD for different sizes of recommendation list, namely $N = \{5, 10, 15, 20\}$, in terms of HR. The same trends hold for ARHR, and for the rest of the proposed approaches. We consider small sizes of $N$, because users are interested only in the first set of recommendations returned, as not many people would look at the hundredth item recommended.

We can see that as the size of the recommendation list increases, the performance of sGLSVD is improved, as a bigger list means that it is more probable for the test item to appear in the recommended list.

### 7.3.2 Performance against competing approaches

#### Comparison against competing latent space approaches

Tables 7.6 and 7.7 compare the performance of our proposed methods rGLSVD and sGLSVD against competitive modern latent space top-$N$ recommendation approaches:

Table 7.6: Comparison with competing latent space approaches in terms of HR.

| | LLORMA | | | PureSVD | | BPRMF | | | | sGLSVD | rGLSVD |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Dataset | $\lambda$ | rank | HR | $f$ | HR | factors | lrnrate | reg | HR | HR | HR |
| groceries | 0.01 | 20 | 0.096 | 738 | 0.134 | 3000 | 0.01 | 0.001 | 0.214 | **0.283** | 0.216 |
| ml10m | 0.01 | 35 | 0.194 | 64 | 0.295 | 5000 | 0.01 | 0.01 | 0.240 | 0.320 | **0.321** |
| jester | 0.01 | 30 | 0.812 | 25 | 0.860 | 300 | 0.01 | 0.01 | 0.903 | 0.865 | **0.910** |
| flixster | 0.001 | 35 | 0.148 | 90 | 0.194 | 4000 | 0.01 | 0.001 | 0.200 | **0.218** | 0.217 |
| netflix | 0.01 | 7 | 0.108 | 50 | 0.204 | 5000 | 0.01 | 0.01 | 0.210 | **0.225** | 0.223 |

LLORMA, PureSVD and BPRMF for all datasets, in terms of HR and ARHR, respectively. The columns indicate the best HR/ARHR achieved along with the set of parameters for which the best results were achieved. The parameters for which the best performance of rGLSVD and sGLSVD is achieved are the same as the ones reported in Tables 7.1, 7.2, 7.3, 7.4 and 7.5.

We can see that our proposed methods outperform the competing latent space top-$N$ approaches, both in terms of HR and in terms of ARHR. We performed paired t-tests of rGLSVD/sGLSVD against the best competing latent space baseline, which was either BPRMF or PureSVD, and the performance difference was shown to be statistically significant. The results of LLORMA being lower in top-$N$ quality than PureSVD surprised us; we believe that the reason is that the original use of LLORMA was for datasets with explicit feedback and for the rating prediction task, thus not necessarily resulting in as good of recommendation quality for performing top-$N$ recommendation on implicit feedback.

From comparing Tables 7.6 and 7.7 with Tables 7.1, 7.2, 7.3, 7.4 and 7.5 we can also see that rLSVD and sLSVD tend to outperform the best competing baseline as well.

**Comparing global & local approaches against standard global approaches**

Tables 7.8 and 7.9 compare the use of global and local approaches, against standard global models both in terms of item-based models (SLIM vs GLSLIM) and latent space models (PureSVD vs proposed method sGLSVD). The comparison is shown both in terms of HR and ARHR.

We can see from the pairwise comparison of SLIM with GLSLIM and of PureSVD

Table 7.7: Comparison with competing latent space approaches in terms of ARHR.

| | LLORMA | | | PureSVD | | BPRMF | | | | sGLSVD | rGLSVD |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Dataset | $\lambda$ | rank | ARHR | $f$ | ARHR | factors | lrnrate | reg | ARHR | ARHR | ARHR |
| groceries | 0.01 | 20 | 0.046 | 700 | 0.059 | 3100 | 0.01 | 0.001 | 0.099 | **0.136** | 0.105 |
| ml10m | 0.01 | 25 | 0.080 | 56 | 0.139 | 7000 | 0.01 | 0.01 | 0.105 | 0.152 | **0.154** |
| jester | 0.01 | 7 | 0.673 | 15 | 0.740 | 100 | 0.01 | 0.01 | 0.766 | 0.746 | **0.783** |
| flixster | 0.001 | 35 | 0.058 | 80 | 0.086 | 4000 | 0.01 | 0.001 | 0.089 | **0.102** | 0.101 |
| netflix | 0.01 | 7 | 0.043 | 50 | 0.091 | 5000 | 0.01 | 0.01 | 0.100 | **0.105** | 0.104 |

Table 7.8: Comparison of global approaches with global & local approaches in terms of HR.

| | SLIM models | | | PureSVD models | | |
|---|---|---|---|---|---|---|
| Dataset | SLIM | GLSLIM | Improved | PureSVD | sGLSVD | Improved |
| groceries | 0.259 | 0.304 | 17.37% | 0.134 | 0.283 | 111.19% |
| ml10m | 0.312 | 0.345 | 10.58% | 0.295 | 0.320 | 8.47% |
| jester | 0.878 | 0.940 | 7.06% | 0.860 | 0.865 | 0.58% |
| flixster | 0.242 | 0.255 | 5.37% | 0.194 | 0.218 | 12.37% |
| netflix | 0.231 | 0.245 | 6.06% | 0.204 | 0.225 | 10.29% |

Table 7.9: Comparison of global approaches with global & local approaches in terms of ARHR.

| | SLIM models | | | PureSVD models | | |
|---|---|---|---|---|---|---|
| Dataset | SLIM | GLSLIM | Improved | PureSVD | sGLSVD | Improved |
| groceries | 0.130 | 0.155 | 19.23% | 0.059 | 0.136 | 130.51% |
| ml10m | 0.151 | 0.170 | 12.58% | 0.139 | 0.152 | 9.35% |
| jester | 0.755 | 0.835 | 10.60% | 0.740 | 0.746 | 0.81% |
| flixster | 0.116 | 0.126 | 8.62% | 0.086 | 0.102 | 18.60% |
| netflix | 0.107 | 0.116 | 8.41% | 0.091 | 0.105 | 15.38% |

with sGLSVD that the global and local approaches always outperform the standard global models. The paired t-tests we ran showed that the performance difference is statistically significant. This showcases their value.

We can also see that GLSLIM performs better than the rest of the approaches. We

Table 7.10: The training time for *ml10m* dataset with 5 clusters.

| Method | mins |
|---|---|
| sGLSVD | 9.3 |
| GLSLIM | 199.2 |
| GLSLIM-warm | 53.7 |

believe that the reason GLSLIM outperforms sGLSVD is that its underlying model, which is SLIM outperforms PureSVD. Also, even though rGLSVD/sGLSVD does not outperform GLSLIM, we can see that in different cases, its percentage of improvement beyond the underlying global model PureSVD can be higher than the corresponding percentage of improvement of GLSLIM beyond SLIM.

Finally, Table 7.10 shows the training time needed for GLSLIM versus sGLSVD, for the *ml10m* dataset with 5 clusters. GLSLIM-warm corresponds to an optimized runtime for GLSLIM, where we initialize the estimated model with a previous model learned, instead of starting from scratch. More details on GLSLIM-warm and on its experimental timing results can be found in Section 6.3.3. For SLIM and GLSLIM, the times shown correspond to $\beta_g = \beta_l = 10$ and $\lambda_g = \lambda_l = 1$. For sGLSVD, the times correspond to $f^g = 55$ and $f^c = 10$. Similar timewise comparisons hold for other parameter choices and for the rest of the datasets. The times shown correspond to one node of the supercomputer Mesabi[1] , which is equipped with 62 GB RAM and 24 cores. We can see that the time needed to train sGLSVD is only a fraction of the time needed to train GLSLIM, which can be of use in cases when faster training is needed.

## 7.4 Conclusion

In this chapter, we proposed the following user model: the behavior of a user can be described by a combination of a set of aspects shared by all users, and of a set of aspects which are specific to the subset the user belongs to. This user model is an extension of the model usually employed by the latent space approaches, which assumes that the behavior of a user can be described by a set of aspects shared by all.

Learning the user model we proposed with a global latent space approach can be

[1] `https://www.msi.umn.edu/content/mesabi`

difficult, because we often have sparse data. Thus, we propose two methods: rGLSVD and sGLSVD, which explicitly encode this structure, by estimating both a set of global factors and sets of user subset specific latent factors. The rGLSVD method assigns the users into different subsets based on their rating patterns and estimates a global model and a set of user subset specific local models whose number of latent dimensions can vary. The sGLSVD method estimates both global and user subset specific local models by keeping the number of latent dimensions the same among the local models but optimizes the grouping of the users.

The experimental evaluation shows that the proposed approaches estimate better latent representations for the users, outperforming competing latent space top-$N$ recommendation approaches significantly, thus showing the merits of the proposed user model. The performance improvement is on average 13% and up to 37%.

# Chapter 8

# Investigating & Using the Error in Top-$N$ Recommendation

Different popular top-$N$ recommender methods, such as SLIM (presented in Section 3.1.1) and PureSVD (presented in Section 3.2.1) recommend items that users have not yet consumed, and as such correspond to missing entries in the user-item matrix. These methods estimate their respective parameters by treating the missing entries as zeros. Consequently, when recommending the missing entries with the highest predicted values, they essentially recommend the missing entries with the highest error. A natural question that arises is what are the properties of the error, how they correlate with the top-$N$ recommendation quality, and how the performance of these algorithms can be improved by *shaping* their errors.

In this chapter, we consider the SLIM and PureSVD methods and that users and items with similar ratings also have similar errors in their missing entries, and vice versa. In particular, for each of these two methods, we show that for the same training set, among the different models that are estimated by changing their respective hyper-parameters, the ones that achieve the best recommendation performance are those that display the closest rating-based and error-based similarities. Utilizing this insight, we develop a method, called ESLIM, which extends SLIM, by enforcing users with similar rating behaviors to also have similar error in their missing entries and likewise for the items. The method is shown to outperform SLIM, especially for predicting items that

have been rated by few users (tail items).

## 8.1 Introduction

Many popular top-$N$ recommender methods, such as PureSVD [6] and SLIM [5], have loss functions which minimize the error on both the observed and the missing entries. They treat the missing entries as zeros, under the assumption that unconsumed items by a user are disliked items, as well. The predictions correspond to the missing entries that have the highest value. Since during model estimation, the missing entries were set to zero, what those methods do is recommend the missing entries that contribute the most to the loss function; i.e., the missing entries with high error.

Consequently, the question that arises is: which are the properties of the error associated with the missing entries and how do they relate to the recommendation performance of top-$N$ recommender methods that estimate their models by treating the missing entries as zero?

In this chapter, we study for the PureSVD and SLIM methods, how the top-$N$ recommendation performance and the error varies for different models, which are estimated with the same training set, by varying the corresponding hyperparameters. Our results show that users and items with similar rating patterns also have similar patterns of error on their missing entries and the best-performing models are the ones that maximize this property. Utilizing these insights, we develop a method called Error-Constrained Sparse LInear Method for top-$N$ recommendation (ESLIM), which enforces the constraint of users and items with similar rating patterns to also have similar error at their missing entries. This is done by incorporating in the SLIM loss function the constraints that the error-based and rating-based representations of users and items need to be close, as additional regularization factors. ESLIM is shown to outperform SLIM, especially for the items that have not been rated by a large number of users (tail items).

Table 8.1: Overview of the notations used in this chapter.

| Symbol | Meaning |
|---|---|
| $\dot{\mathbf{E}}$ | Error on the missing entries matrix of size $n \times m$ |
| $\mathbf{A}$ | User rating-based similarities matrix of size $n \times n$ |
| $\mathbf{B}$ | User error-based similarities matrix of size $n \times n$ |
| $\mathbf{C}$ | Item rating-based similarities matrix of size $m \times m$ |
| $\mathbf{D}$ | Item error-based similarities matrix of size $m \times m$ |

## 8.2 Notation and definitions

### 8.2.1 Error on the missing entries

We use the notation $\dot{\mathbf{E}}$ to represent the $n \times m$ matrix of the error on the missing entries. For SLIM, every entry $\dot{e}_{ui}$ corresponding to user $u$ and item $i$ of matrix $\dot{\mathbf{E}}$ is:

$$\dot{e}_{ui} = \begin{cases} \mathbf{r}_u^T \mathbf{s}_i, & \text{if } r_{ui} = 0 \\ 0, & \text{if } r_{ui} \neq 0, \end{cases} \tag{8.1}$$

whereas for PureSVD is:

$$\dot{e}_{ui} = \begin{cases} \mathbf{p}_u^T \mathbf{\Sigma}_f \mathbf{q}_i, & \text{if } r_{ui} = 0 \\ 0, & \text{if } r_{ui} \neq 0. \end{cases} \tag{8.2}$$

### 8.2.2 Similarity matrices

We represent a user $u$ as a vector of size $n$, which shows the similarities of user $u$ to other users. We utilize the cosine similarity measure for the similarity computations. We use two representations for every user: a *rating-based* representation, that shows how similar he/she is to other users in terms of their ratings, and an *error-based* representation, which shows how similar he/she is to other users in terms of their error at the missing entries, as shown in Equations 8.1 and 8.2. Thus, we have two $n \times n$ matrices containing the user similarities to the other users: the matrix $\mathbf{A}$ that contains the rating-based user similarities, and the matrix $\mathbf{C}$ that contains the error-based user similarities.

Correspondingly, we use two $m \times m$ matrices containing the cosine similarities of items to other items: the matrix $\mathbf{B}$ that contains the item similarities based on the

ratings, and the matrix $\mathbf{D}$ that contains the item similarities based on the error at the missing entries. All the matrices representing the user and item similarities are dense, non-negative and symmetric. An overview of the notations we use throughout the chapter can be found in Table 8.1.

## 8.3 Analysis of the properties of the error for SLIM and PureSVD

### 8.3.1 Theoretical analysis

We hypothesize that in good-performing models users with similar rating behaviors have similar error in their missing entries. Likewise for items, we hypothesize that similarly rated items have similar error in their missing entries. Also, the better the performance of a model the closer their rating-based and error-based representations are.

The reasoning behind our hypothesis is the following: If users $u$ and $v$ are very similar based on their ratings, their rating-based similarity $a_{uv}$ will have a large value. We expect their error-based similarity $c_{uv}$ to also have a large value, as a good-performing model should have similar predicting performance, thus similar error on users with similar ratings. Similarly, if users $u$ and $v$ are extremely dissimilar, their rating similarity $a_{uv}$ will be small. Then, we would also expect their error-based similarity $c_{uv}$ to be small, as a good-performing model should have different performance on users with very different rating behaviors, thus different error on their missing entries. A similar argument can be made for the items.

The above hypothesis can be shown mathematically in the following way: If we denote with $\mathcal{N}_u$ the set of items that have not been rated by user $u$, and with $\mathcal{N}_v$ the set of items that have not been rated by user $v$, the error-based similarity for users $u$

and $v$, for SLIM models can be expressed as:

$$
\begin{aligned}
c_{uv} &= \frac{\dot{\mathbf{e}}_u^T \dot{\mathbf{e}}_v}{||\dot{\mathbf{e}}_u||_2 ||\dot{\mathbf{e}}_v||_2} = \frac{\sum\limits_{i \in \mathcal{N}_u \cap \mathcal{N}_v} (\dot{e}_{ui})(\dot{e}_{vi})}{\sqrt{\sum\limits_{i \in \mathcal{N}_u} (\dot{e}_{ui})^2} \sqrt{\sum\limits_{i \in \mathcal{N}_v} (\dot{e}_{vi})^2}} = \frac{\sum\limits_{i \in \mathcal{N}_u \cap \mathcal{N}_v} (\mathbf{r}_u^T \mathbf{s}_i)(\mathbf{r}_v^T \mathbf{s}_i)}{\sqrt{\sum\limits_{i \in \mathcal{N}_u} (\mathbf{r}_u^T \mathbf{s}_i)^2} \sqrt{\sum\limits_{i \in \mathcal{N}_v} (\mathbf{r}_v^T \mathbf{s}_i)^2}} \\
&= \frac{\sum\limits_{i \in \mathcal{N}_u \cap \mathcal{N}_v} (\mathbf{r}_u^T \mathbf{s}_i)(\mathbf{s}_i^T \mathbf{r}_v)}{\sqrt{\sum\limits_{i \in \mathcal{N}_u} (\mathbf{r}_u^T \mathbf{s}_i)(\mathbf{s}_i^T \mathbf{r}_u)} \sqrt{\sum\limits_{i \in \mathcal{N}_v} (\mathbf{r}_v^T \mathbf{s}_i)(\mathbf{s}_i^T \mathbf{r}_v)}} = \frac{\sum\limits_{i \in \mathcal{N}_u \cap \mathcal{N}_v} (\mathbf{r}_u^T ||\mathbf{s}_i||_2^2 \mathbf{r}_v)}{\sqrt{\sum\limits_{i \in \mathcal{N}_u} \mathbf{r}_u^T ||\mathbf{s}_i||_2^2 \mathbf{r}_u)} \sqrt{\sum\limits_{i \in \mathcal{N}_v} \mathbf{r}_v^T ||\mathbf{s}_i||_2^2 \mathbf{r}_v)}} \\
&= \frac{\mathbf{r}_u^T \mathbf{r}_v}{||\mathbf{r}_u||_2 ||\mathbf{r}_v||_2} \frac{\sum\limits_{i \in \mathcal{N}_u \cap \mathcal{N}_v} ||\mathbf{s}_i||_2^2}{\sqrt{\sum\limits_{i \in \mathcal{N}_u} ||\mathbf{s}_i||_2^2} \sqrt{\sum\limits_{i \in \mathcal{N}_v} ||\mathbf{s}_i||_2^2}} = a_{uv} \frac{\sum\limits_{i \in \mathcal{N}_u \cap \mathcal{N}_v} ||\mathbf{s}_i||_2^2}{\sqrt{\sum\limits_{i \in \mathcal{N}_u} ||\mathbf{s}_i||_2^2} \sqrt{\sum\limits_{i \in \mathcal{N}_v} ||\mathbf{s}_i||_2^2}}.
\end{aligned}
\tag{8.3}
$$

A similar mathematical relation holds for PureSVD models: The rating-based similarity $a_{uv}$ between pairs of users $u$ and $v$ can be expressed as:

$$
\begin{aligned}
a_{uv} &= \frac{\mathbf{r}_u^T \mathbf{r}_v}{||\mathbf{r}_u||_2 ||\mathbf{r}_v||_2} = \frac{\sum\limits_{i=1}^{m} (r_{ui})(r_{vi})}{\sqrt{\sum\limits_{i=1}^{m} (r_{ui})^2} \sqrt{\sum\limits_{i=1}^{m} (r_{vi})^2}} = \frac{\sum\limits_{i=1}^{m} (\mathbf{p}_u^T \boldsymbol{\Sigma}_f \mathbf{q}_i)(\mathbf{p}_v^T \boldsymbol{\Sigma}_f \mathbf{q}_i)}{\sqrt{\sum\limits_{i=1}^{m} (\mathbf{p}_u^T \boldsymbol{\Sigma}_f \mathbf{q}_i)^2} \sqrt{\sum\limits_{i=1}^{m} (\mathbf{p}_v^T \boldsymbol{\Sigma}_f \mathbf{q}_i)^2}} \\
&= \frac{\sum\limits_{i=1}^{m} \mathbf{p}_u^T \boldsymbol{\Sigma}_f ||\mathbf{q}_i||_2^2 \boldsymbol{\Sigma}_f \mathbf{p}_v}{\sqrt{\sum\limits_{i=1}^{m} \mathbf{p}_u^T \boldsymbol{\Sigma}_f ||\mathbf{q}_i||_2^2 \boldsymbol{\Sigma}_f \mathbf{p}_u} \sqrt{\sum\limits_{i=1}^{m} \mathbf{p}_v^T \boldsymbol{\Sigma}_f ||\mathbf{q}_i||_2^2 \boldsymbol{\Sigma}_f \mathbf{p}_v}} \\
&= \frac{\mathbf{p}_u^T \boldsymbol{\Sigma}_f^2 \mathbf{p}_v \sum\limits_{i=1}^{m} ||\mathbf{q}_i||_2^2}{\sqrt{\mathbf{p}_u^T \boldsymbol{\Sigma}_f^2 \mathbf{p}_u \sum\limits_{i=1}^{m} ||\mathbf{q}_i||_2^2} \sqrt{\mathbf{p}_v^T \boldsymbol{\Sigma}_f^2 \mathbf{p}_v \sum\limits_{i=1}^{m} ||\mathbf{q}_i||_2^2}} = \frac{\mathbf{p}_u^T \boldsymbol{\Sigma}_f^2 \mathbf{p}_v}{\sqrt{\mathbf{p}_u^T \boldsymbol{\Sigma}_f^2 \mathbf{p}_u} \sqrt{\mathbf{p}_v^T \boldsymbol{\Sigma}_f^2 \mathbf{p}_v}}.
\end{aligned}
\tag{8.4}
$$

Thus, by taking into account Equation (8.4), the error-based similarity between

users $u$ and $v$ is:

$$c_{uv} = \frac{\dot{\mathbf{e}}_u^T \dot{\mathbf{e}}_v}{||\dot{\mathbf{e}}_u||_2 ||\dot{\mathbf{e}}_v||_2} = \frac{\sum\limits_{i \in \mathcal{N}_u \cap \mathcal{N}_v} (\dot{e}_{ui})(\dot{e}_{vi})}{\sqrt{\sum\limits_{i \in \mathcal{N}_u} (\dot{e}_{ui})^2} \sqrt{\sum\limits_{i \in \mathcal{N}_v} (\dot{e}_{vi})^2}} = \frac{\sum\limits_{i \in \mathcal{N}_u \cap \mathcal{N}_v} (\mathbf{p}_u^T \boldsymbol{\Sigma}_f \mathbf{q}_i)(\mathbf{p}_v^T \boldsymbol{\Sigma}_f \mathbf{q}_i)}{\sqrt{\sum\limits_{i \in \mathcal{N}_u} (\mathbf{p}_u^T \boldsymbol{\Sigma}_f \mathbf{q}_i)^2} \sqrt{\sum\limits_{i \in \mathcal{N}_v} (\mathbf{p}_v^T \boldsymbol{\Sigma}_f \mathbf{q}_i)^2}}$$

$$= \frac{\sum\limits_{i \in \mathcal{N}_u \cap \mathcal{N}_v} \mathbf{p}_u^T \boldsymbol{\Sigma}_f ||\mathbf{q}_i||_2^2 \boldsymbol{\Sigma}_f \mathbf{p}_v}{\sqrt{\sum\limits_{i \in \mathcal{N}_u} \mathbf{p}_u^T \boldsymbol{\Sigma}_f ||\mathbf{q}_i||_2^2 \boldsymbol{\Sigma}_f \mathbf{p}_u} \sqrt{\sum\limits_{i \in \mathcal{N}_v} \mathbf{p}_v^T \boldsymbol{\Sigma}_f ||\mathbf{q}_i||_2^2 \boldsymbol{\Sigma}_f \mathbf{p}_v}}$$

$$= \frac{\mathbf{p}_u^T \boldsymbol{\Sigma}_f^2 \mathbf{p}_v \sum\limits_{i \in \mathcal{N}_u \cap \mathcal{N}_v} ||\mathbf{q}_i||_2^2}{\sqrt{\mathbf{p}_u^T \boldsymbol{\Sigma}_f^2 \mathbf{p}_u \sum\limits_{i \in \mathcal{N}_u} ||\mathbf{q}_i||_2^2} \sqrt{\mathbf{p}_v^T \boldsymbol{\Sigma}_f^2 \mathbf{p}_v \sum\limits_{i \in \mathcal{N}_v} ||\mathbf{q}_i||_2^2}} = a_{uv} \frac{\sum\limits_{i \in \mathcal{N}_u \cap \mathcal{N}_v} ||\mathbf{q}_i||_2^2}{\sqrt{\sum\limits_{i \in \mathcal{N}_u} ||\mathbf{q}_i||_2^2} \sqrt{\sum\limits_{i \in \mathcal{N}_v} ||\mathbf{q}_i||_2^2}}. \tag{8.5}$$

This shows that the error-based similarity $c_{uv}$ between users $u$ and $v$ is their rating-based similarity $a_{uv}$ multiplied by a term, which is $\frac{\sum\limits_{i \in \mathcal{N}_u \cap \mathcal{N}_v} ||\mathbf{s}_i||_2^2}{\sqrt{\sum\limits_{i \in \mathcal{N}_u} ||\mathbf{s}_i||_2^2} \sqrt{\sum\limits_{i \in \mathcal{N}_v} ||\mathbf{s}_i||_2^2}}$ for SLIM models and $\frac{\sum\limits_{i \in \mathcal{N}_u \cap \mathcal{N}_v} ||\mathbf{q}_i||_2^2}{\sqrt{\sum\limits_{i \in \mathcal{N}_u} ||\mathbf{q}_i||_2^2} \sqrt{\sum\limits_{i \in \mathcal{N}_v} ||\mathbf{q}_i||_2^2}}$ for PureSVD models, from which we can conclude that users with similar error should have similar ratings and vice versa. Similar conclusions can be reached for the items.

### 8.3.2 Experimental analysis

We estimate multiple PureSVD and SLIM models for the same train and test data, by varying the corresponding parameters: the rank $f$ for PureSVD and the $l_2$ regularization parameter $\beta$ for SLIM. We keep the $l_1$ regularization parameter $\lambda$ fixed for SLIM, in order to only have one parameter affecting the performance. We thus decided to run SLIM with only $l_2$ regularization. For every model estimated, we compare the error-based and the rating-based representations of users and items and see how they correlate with the performance of the model.

Figures 8.1 and 8.2 show for every pair of users $(u, v)$ their rating-based similarity $a_{uv}$ and their error-based similarity $c_{uv}$, for SLIM and PureSVD models, correspondingly, for the *ml100k* dataset. The line shown corresponds to the line that best fits the data shown, minimizing the least square error. Similar trends can be seen for other datasets,

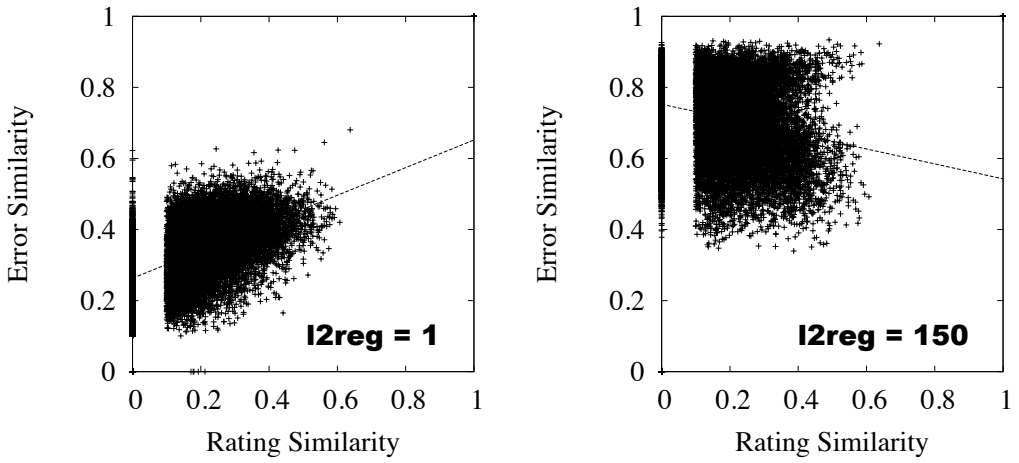



Figure 8.1: Scatterplot of rating and error similarities $a_{uv}$ and $c_{uv}$ for all pairs of users $u$ and $v$, for a good-performing SLIM model (estimated with $\beta = 1$ and resulting in HR = 0.33) and a worse-performing one (estimated with $\beta = 150$ and resulting in HR = 0.24) for the *ml100k* dataset.

and for item-based similarities. Note that the rating-based similarities remain constant across the different models, while the error-based similarities change.

Figure 8.1 shows the user similarities for a good-performing SLIM model (estimated with $\beta = 1$ and resulting in HR = 0.33) and for a bad-performing SLIM model (estimated with $\beta = 150$ with HR = 0.24). We can see that for the majority of user pairs, their error-based similarities remain in the same range of values as their rating-based similarities $[0.2, 0.6]$, for the good-performing SLIM model, generally indicating a linear-type relationship between $a_{uv}$ and $c_{uv}$. On the other hand, we can see for the SLIM model with the worse performance, that the error-based similarities tend to be in a different range $[0.4, 0.9]$ than the corresponding rating-based similarities. As the regularization is very high, the model estimated is very sparse, thus most of the users are very similar in terms of their error. We also computed the Pearson correlation coefficient among all the pairs of similarities $a_{uv}$ and $c_{uv}$, and it is 0.787 for the good-performing SLIM model with $\beta = 1$ and 0.580 for the worse-performing SLIM model with $\beta = 150$.

Similarly, Figure 8.2 shows the user similarities for a good-performing PureSVD

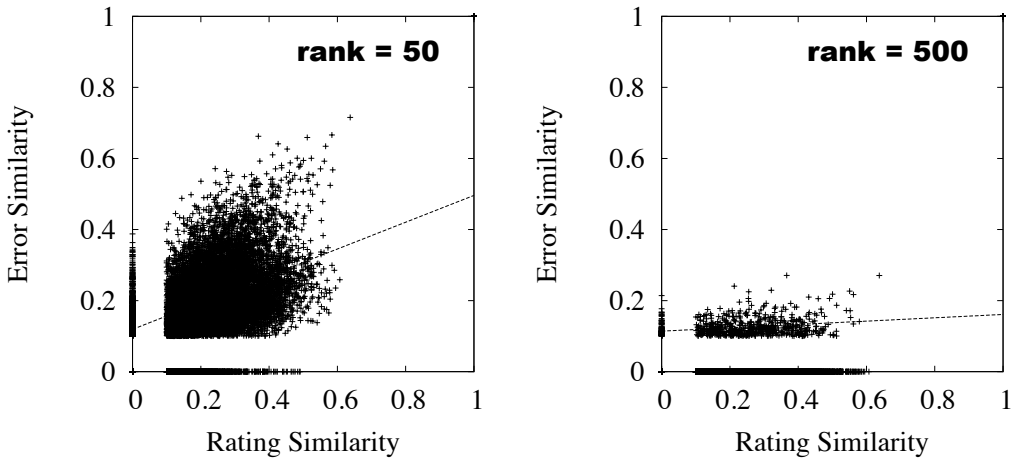



Figure 8.2: Scatterplot of rating and error similarities $a_{uv}$ and $c_{uv}$ for all pairs of users $u$ and $v$, for a good-performing PureSVD model (estimated with $f = 50$ and resulting in HR = 0.296) and for a bad-performing PureSVD model (estimated with $f = 500$ and resulting in HR = 0.056), for the *ml100k* dataset.

model (estimated with $f = 50$ with HR = 0.296) and for a bad-performing PureSVD model (estimated with $f = 500$ with HR = 0.056). We can see that with the good-performing PureSVD model, the majority of users have error similarity within the values of 0.2 and 0.6, which is where the majority of rating-based similarities lie. The Pearson correlation coefficient was found to be 0.817. On the other hand, the bad-performing PureSVD model leads to the majority of the users having a zero error similarity, as the estimated model overfits the users. The Pearson correlation coefficient was found to be 0.288.

We can see that the good-performing models (both SLIM and PureSVD models) tend to show for the majority of pairs of users error-based similarities very close to their rating-based similarities, as indicated from the similar range of values, the shape of the data, and the high Pearson correlation coefficient. On the other hand, the models with worse performance exhibit error-based similarities, which are not close to the corresponding rating-based similarities.

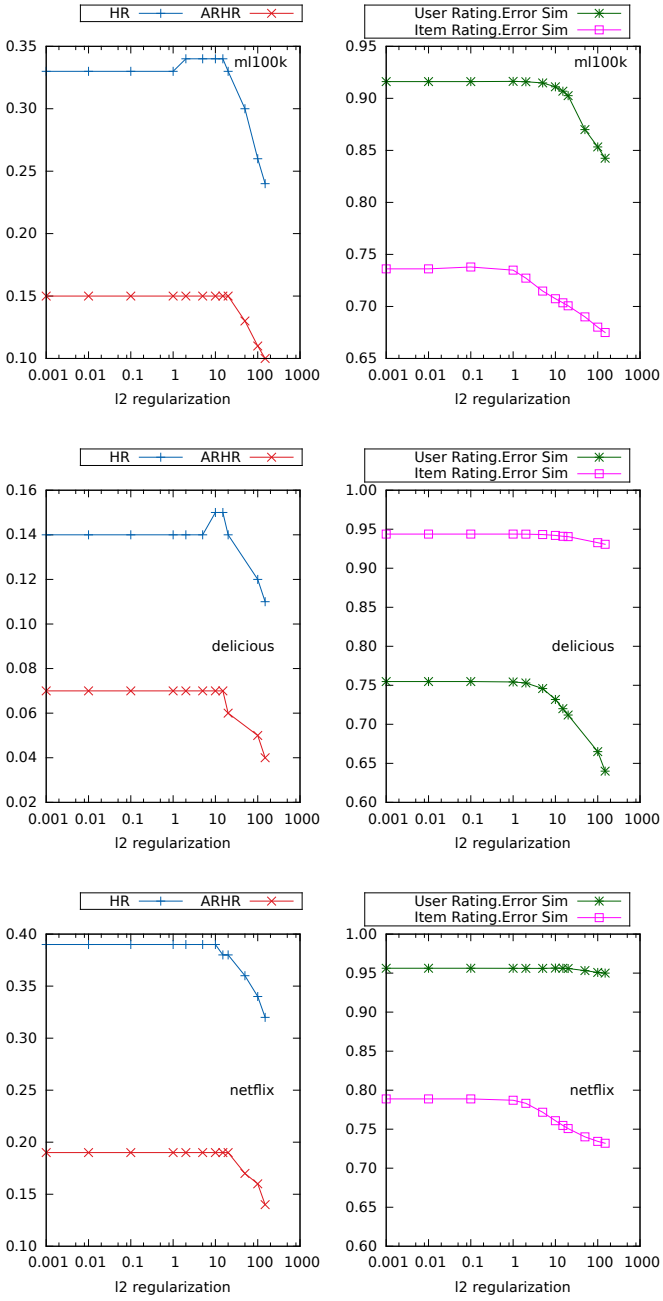



Figure 8.3: The effect of the $l_2$ regularization $\beta$ on the performance of SLIM and on the corresponding 'User Rating.Error Similarity' and 'Item Rating.Error Similarity'. The maximum HR and ARHR are achieved for the values of $\beta$ for which the 'User Rating.Error Similarity' and 'Item Rating.Error Similarity' also obtain their local maxima.

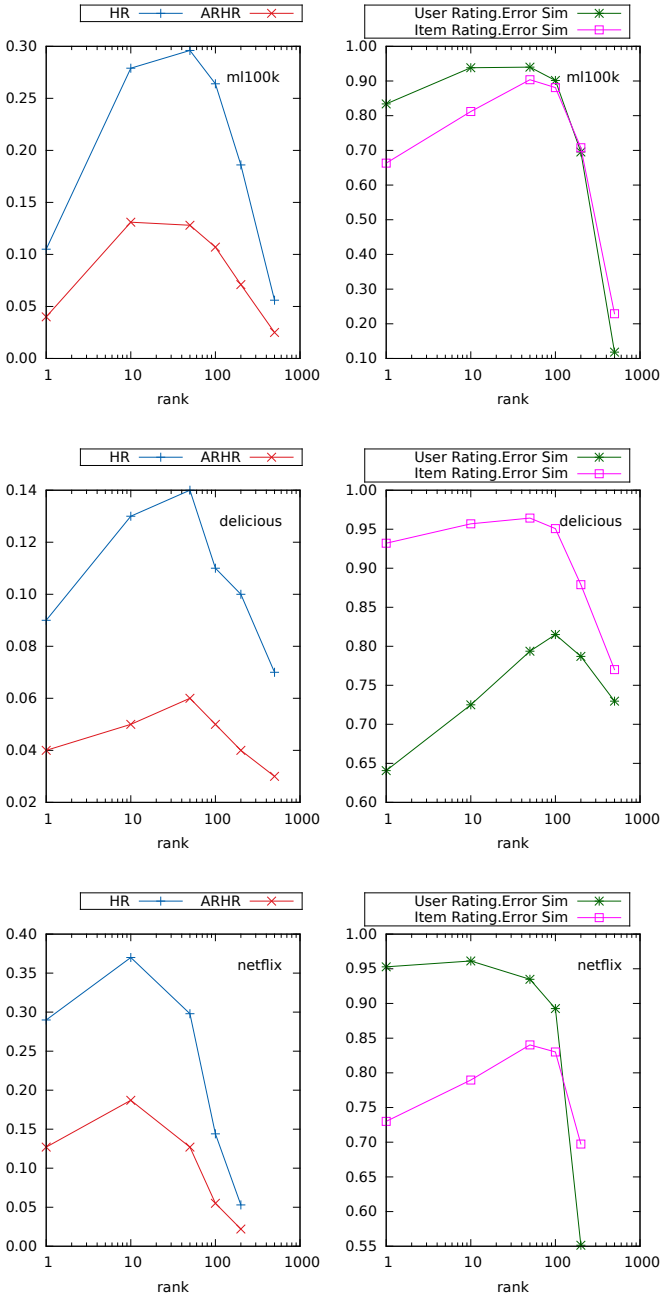



Figure 8.4: The effect of the rank $f$ on the performance of PureSVD and on the corresponding 'User Rating.Error Similarity' and 'Item Rating.Error Similarity'. The maximum HR and ARHR are achieved for the values of the rank $f$ for which the 'User Rating.Error Similarity' and 'Item Rating.Error Similarity' also obtain their local maxima.

In order to better examine the performance of the model in relation to how similar the error-based and the rating-based representations of users are, we compute the measure:

$$\text{User Rating.Error Similarity} = \frac{\sum_{u=1}^{n} \cos(\mathbf{a}_u, \mathbf{c}_u)}{n}, \tag{8.6}$$

which computes for every user $u$ the cosine similarity between his/her rating-based vector of similarities $\mathbf{a}_u$ and his/her error-based vector of similarities $\mathbf{c}_u$, thus finding how similar his/her two representations are and then takes the average for all of the users.

Similarly, we compute for the items the measure:

$$\text{Item Rating.Error Similarity} = \frac{\sum_{i=1}^{m} \cos(\mathbf{b}_i, \mathbf{d}_i)}{m}, \tag{8.7}$$

which computes for every item $i$ how close its rating-based representation $\mathbf{b}_i$ and its error-based representation $\mathbf{d}_i$ are, using the cosine similarity measure and then finds the average for all items.

Figures 8.3 and 8.4 show how the performance of the models (SLIM and PureSVD correspondingly), the 'User Rating.Error Similarity' (Equation (8.6)) and the 'Item Rating.Error Similarity' (Equation (8.7)) vary while varying the regularization parameters, for the *ml100k*, *delicious* and *netflix* datasets. The regularization parameters are $\beta$ for SLIM models and the rank $f$ for the PureSVD models.

We can see that for both PureSVD and SLIM, the performance in terms of HR and ARHR follows the same trend as the 'User Rating.Error Similarity' and 'Item Rating.Error Similarity' measures, showing that the performance of the models achieves its peak for the values of the parameters for which the 'User Rating.Error Similarity' and 'Item Rating.Error Similarity' measures are maximum.

We can also see that the best performing model is the one producing very close error-based and rating-based representations. In other words, the performance on the test set is the highest in terms of HR and ARHR, when the 'User Rating.Error Similarity' and 'Item Rating.Error Similarity' obtain their highest values.

Although Figures 8.3 and 8.4 compute how close the rating-based and error-based representations are using the cosine similarity measure, we can reach the same conclusion by using a different measure. Figure 8.5 shows for the *delicious* dataset the average cosine similarity between the user rating-based and error-based representations ('User

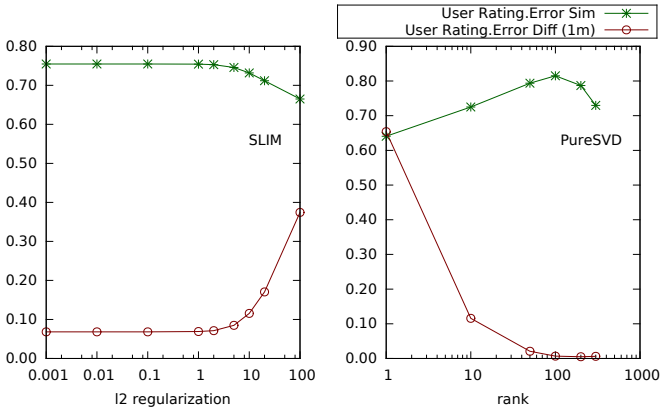



Figure 8.5: Examining how close the user rating-based and error-based representations are, in terms of their average cosine similarity and the frobenius norm of their difference for the *delicious* dataset, for SLIM and PureSVD models, while varying the respective parameters. The cosine similarity takes its highest values for the parameter values for which the frobenius norm of their difference takes its lowest values.

Rating.Error Similarity') and the frobenius norm of their difference ($||\mathbf{C} - \mathbf{A}||_F$) for SLIM and PureSVD models, while varying their respective regularization parameters. Similar conclusions can be drawn for the items as well, and for the rest of the datasets. We can see that the average cosine similarity between the two representations becomes lower for the values of the regularization parameters for which the frobenius norm of the difference between the two representations (shown in millions) becomes higher, and vice versa.

Thus, from Figures 8.3, 8.4 and 8.5, we can see that the performance on the test set is the highest in terms of HR and ARHR, when the rating-based and error-based representations are the closest for users and items, which can be expressed in terms of their cosine similarity being the highest, or in terms of the frobenius norm of their difference being the lowest.

## 8.4 Proposed approach

### 8.4.1 Overview

Utilizing the above insights, we develop a method called Error-Constrained Sparse LInear Method for top-$N$ recommendation (ESLIM), which modifies the loss function of SLIM (presented in Section 3.1.1) to introduce a regularization term that shapes the error.

The overall optimization problem that ESLIM solves $\forall i \in \{1, \ldots, m\}$ is:

$$\begin{aligned} \underset{\mathbf{s}_i}{\text{minimize}} \quad & \tfrac{1}{2}||\mathbf{r}_i - \mathbf{R}\mathbf{s}_i||_2^2 + \tfrac{\beta}{2}||\mathbf{s}_i||_2^2 + \tfrac{l_u}{2}||\mathbf{C} - \mathbf{A}||_F^2 + \tfrac{l_i}{2}||\mathbf{D} - \mathbf{B}||_F^2, \\ \text{subject to} \quad & \mathbf{s}_i \geq 0, \text{and } s_{ii} = 0. \end{aligned} \tag{8.8}$$

The optimization problem has four components: (i) the main SLIM component of fitting the ratings $||\mathbf{r}_i - \mathbf{R}\mathbf{s}_i||_2^2$, (ii) the $l_2$ regularization of $\mathbf{s}_i$ controlled by the parameter $\beta$ (iii) the term that the user rating similarity matrix $\mathbf{A}$ and the user error similarity matrix $\mathbf{C}$ should be similar which is controlled by the parameter $l_u$ and (iv) the term that the item rating similarity matrix $\mathbf{B}$ and the item error similarity $\mathbf{D}$ should be similar which is controlled by the regularization parameter $l_i$.

Higher values of $l_u$ and $l_i$ lead to more severe regularization. The constraints $\mathbf{s}_i \geq 0$ and $s_{ii} = 0$ enforce that the sparse aggregation vector $\mathbf{s}_i$ will have non-negative coefficients and when computing the weights of an item $i$, the item itself will not be used; as this would lead to trivial solutions.

By stacking together every column $\mathbf{s}_i$ $\forall i$, we get the sparse aggregation coefficient matrix $\mathbf{S}$. Every column $\mathbf{s}_i$ can be estimated *in parallel*.

We use the RMSprop method [84] to solve the optimization problem of Equation (8.8), which eliminates the need to manually tune the learning rate.

The top-$N$ recommendation in ESLIM is performed in the following way: For every user $u$, we compute the estimated ratings $\tilde{r}_{ui}$ for all the unrated items $i$:

$$\tilde{r}_{ui} = \mathbf{r}_u^T \mathbf{s}_i, \tag{8.9}$$

we sort these values and we recommend the top-$N$ with the highest ratings to the target user $u$.

## 8.5 Experimental results

Here, we present the performance of ESLIM, and compare it to SLIM to see how enforcing the constraint of similar structure between the rating similarity and the error similarity matrices affects the quality of top-$N$ recommendation.

Details of the datasets we used can be found in Section 4.1. We compared the performance of ESLIM against SLIM [5], which we implemented for fairness of comparison, by solving the optimization problem of Equation (8.8), by setting $l_u = l_i = 0$. We performed an extensive search over the parameter space, to find the set of parameters that gives us the best performance. The $\beta$ regularization parameter was chosen from the set of values: $\{0.1, 1, 10, 100, 1000\}$. The $l_u$ and $l_i$ regularization parameters were chosen from the set of values: $\{0, 0.001, 0.01, 0.1, 1\}$.

As ESLIM enforces the constraint of having close rating-based and error-based representations for both the users and the items, we wanted to investigate how each of these constraints affects the recommendation performance. Thus, we experimentally tested two variants of ESLIM:

- ESLIM-u, which stands for ESLIM for users. In ESLIM-u, the constraint shaping the error for users is enforced: the users with similar ratings are enforced to have a similar error on their missing entries. The optimization problem of ESLIM-u is the following:

$$\begin{aligned} \underset{\mathbf{s}_i}{\text{minimize}} \quad & \tfrac{1}{2}||\mathbf{r}_i - \mathbf{R}\mathbf{s}_i||_2^2 + \tfrac{\beta}{2}||\mathbf{s}_i||_2^2 + l_u||\mathbf{C} - \mathbf{A}||_F^2, \\ \text{subject to} \quad & \mathbf{s}_i \geq 0, \text{and } s_{ii} = 0. \end{aligned} \tag{8.10}$$

- ESLIM-i, which stands for ESLIM for items. In ESLIM-i, the constraint shaping the error for items is enforced: the items that are rated similarly are enforced to have similar error on their missing entries. The optimization problem of ESLIM-i is the following:

$$\begin{aligned} \underset{\mathbf{s}_i}{\text{minimize}} \quad & \tfrac{1}{2}||\mathbf{r}_i - \mathbf{R}\mathbf{s}_i||_2^2 + \tfrac{\beta}{2}||\mathbf{s}_i||_2^2 + l_i||\mathbf{D} - \mathbf{B}||_F^2, \\ \text{subject to} \quad & \mathbf{s}_i \geq 0, \text{and } s_{ii} = 0. \end{aligned} \tag{8.11}$$

Table 8.3 compares the performance of ESLIM-u, ESLIM-i and SLIM, in terms of HR and ARHR, for the *ml100k* dataset, the *delicious* dataset and a subset of the *netflix*

Table 8.2: Comparison between SLIM, ESLIM-u and ESLIM-i in terms of HR.

| | SLIM | | ESLIM-u | | | ESLIM-i | | |
|---|---|---|---|---|---|---|---|---|
| Dataset | $\beta$ | HR | $l_u$ | $\beta$ | HR | $l_i$ | $\beta$ | HR |
| ml100k | 1 | 0.333 | 0.001 | 100 | **0.342** | 0.01 | 10 | **0.342** |
| delicious | 100 | **0.150** | 0.01 | 100 | 0.142 | 0.01 | 100 | 0.146 |
| netflix-s | 10 | 0.394 | 0.01 | 10 | 0.395 | 0.01 | 1 | **0.396** |

Table 8.3: Comparison between SLIM, ESLIM-u and ESLIM-i in terms of ARHR.

| | SLIM | | ESLIM-u | | | ESLIM-i | | |
|---|---|---|---|---|---|---|---|---|
| Dataset | $\beta$ | ARHR | $l_u$ | $\beta$ | ARHR | $l_i$ | $\beta$ | ARHR |
| ml100k | 10 | 0.153 | 0.1 | 100 | **0.155** | 0.01 | 10 | 0.154 |
| delicious | 100 | 0.069 | 0.001 | 1 | 0.066 | 0.01 | 100 | **0.070** |
| netflix-s | 100 | 0.187 | 0.01 | 100 | **0.189** | 0.01 | 100 | 0.188 |

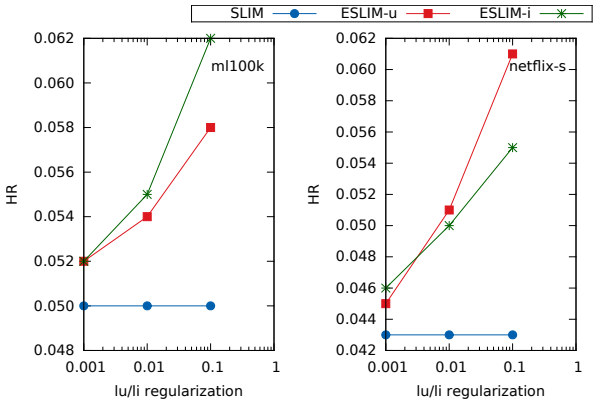



Figure 8.6: The performance of ESLIM-u and ESLIM-i for the tail items (50% least frequent items), while varying the $l_u/l_i$ regularization parameters. The performance of SLIM on the tail items is also shown for comparison purposes.

dataset, which we call *netflix-s*. The *netflix-s* dataset was created by choosing random $2,000$ out of the top 25% of the densest users and from this subset choosing random $1,000$ items out of the top 50% of the densest items. For each method, the columns correspond to the best HR and ARHR and the parameters for which they are achieved.

The parameters are: $\beta$ for SLIM, $\beta$ and $l_u$ for ESLIM-u and $\beta$ and $l_i$ for ESLIM-i. The best performance is shown for each dataset in bold, along with the parameters for which it was achieved.

We can see that ESLIM-u and ESLIM-i tend to outperform SLIM for the majority of the cases, but the gains are not shown to be significant. This can be accounted to the fact that the best SLIM model is found through model selection; in other words the results shown correspond to the model that already exhibits the property that similar users and items have similar error. Thus, by explicitly enforcing this property, we do not have significant benefits.

We can better understand how adding the constraints of having close rating-based and error-based representations for users and items in the loss function impacts the top-$N$ recommendation performance, in the following way: We split the items in two groups: the 50% most frequent items in the train set which comprise the head items and the 50% least frequent items which comprise the tail items and examine the performance of SLIM, ESLIM-u and ESLIM-i on each group separately. Figure 8.6 shows the performance in terms of HR on the tail items for the *ml100k* and the *netflix-s* datasets. The performance of SLIM on the tail items is shown as a constant line across the different $l_u$, $l_i$ regularization values for comparison purposes, (although it was achieved for the value of $l_u = l_i = 0$). Similar trends hold for the ARHR.

We can see that ESLIM-u and ESLIM-i outperform SLIM for the tail items. Also, higher values of the regularization parameters $l_u$ and $l_i$, which means more enforced constraints of having close rating-based and error-based similarity matrices, lead to even better recommendation performance. On the other hand, the performance of ESLIM-u and ESLIM-i is similar to or worse than SLIM on the head items, with the effect increasing while the value of the parameters $l_u$ and $l_i$ increases.

Thus, we can conclude that the gains of ESLIM-u and ESLIM-i beyond SLIM are achieved for datasets which have a lot of tail items. The frequencies of the 50% least frequent items for the *ml100k* dataset lie in the interval $[1, 27]$, which means that they have been rated from 1 up to at most 27 times. The frequencies of the 50% least frequent items for the *netflix-s* dataset lie in the interval $[8, 40]$. So, both of these datasets have a lot of infrequent items. On the other hand, for the *delicious* dataset, the frequencies of the 50% least frequent items lie in the interval $[63, 88]$ showing that there are not

infrequent items. We believe that the reason why the gains for the *delicious* dataset are not as clear can be explained by the absence of tail items.

We can thus see that although ESLIM-u and ESLIM-i might not lead to significant overall gains over SLIM, they achieve better performance over SLIM on the tail items. The gains are more significant, when the tail is more prevalent. We think that the reason is that while SLIM estimates models that tend to exhibit the property that similar users/items should have similar error for the head items; the property is not satisfied as clearly for the tail items, thus enforcing it explicitly leads to better performance for them.

## 8.6 Conclusion

In this chapter, we studied how the properties of the error change, while the performance of the models changes, for popular top-$N$ recommendation methods SLIM and PureSVD, which treat missing entries as zeros. We showed that users/items with similar rating patterns, also have similar error on their missing entries. Moreover, the best-performing model is the one that maximizes this property.

We used this finding to develop an approach ESLIM, which modifies the loss function of SLIM, by adding constraints that enforce the rating similarity matrix to be close to the error similarity matrix. The experimental evaluation of our method showed that ESLIM, while achieving performance gains, does not outperform significantly SLIM, since the best-performing SLIM model is chosen by model selection and already exhibits the property of the rating similarity matrix and the error similarity matrix to be close. However, ESLIM was shown to outperform SLIM, for the tail items.

# Chapter 9

# Conclusion

## 9.1 Thesis summary

Recommender systems are present on the everyday lives of millions of people. They help them navigate through a plethora of choices and information and make an educated and informed choice. Among them, top-$N$ recommender systems that provide users with a ranked list of $N$ items are very popular as they present the users with a list of few $N$ items they would likely be interested in, and thus the user can make decisions fast, without having to browse through a huge list. The quality of the recommendations is crucial; a top-$N$ recommendation system that provides bad recommendations will leave the user unsatisfied and he/she will stop using it.

This thesis focused on the development of novel methods to improve the quality of top-$N$ recommendations in a scalable manner. The methods we proposed can be applied on user-item implicit feedback data, which are prevalent. Our methods have been applied on multiple real-world datasets and show significant improvement above competing state-of-the-art baselines. Moreover, the thesis provided insight into the top-$N$ recommendation task, drawing novel conclusions.

The main areas that our thesis explored are:

- **Identifying and exploiting higher-order sets of items, beyond pairs to perform top-$N$ recommendation.** Although item-item approaches that utilize pairs of items have been shown to perform well for the top-$N$ recommendation task, in many cases users consume items in sets. We showed that there

are a lot of real-world datasets with prevalent higher-order information. In order to take advantage of this higher-order information, we contributed an approach (HOSLIM) based on structural equation modeling to generalize the item-item approaches to also incorporate itemset-item information. The experimental evaluation of this approach, performed on a variety of real-world datasets, shows that HOSLIM achieves considerable improvements of 7.86% on average over competing item-item approaches. Also, for domains that exhibit such set-based consumption characteristics, the gains can reach up to 32% over competing baselines.

- **Estimating multiple user-subset-specific item-item models for top-$N$ recommendation.** The item-item approaches also suffer from the fact that they only estimate a global model, thus not being very personalized. If two items are considered very similar for a user subset, but dissimilar for another, their similarity computed from a global model will tend towards some average value; thus losing the important information that they are considered very similar for the users of the first subset. We contributed an approach (GLSLIM) that combines the global model along with local item-item models estimated for different subsets of users. The assignment of the users to the subsets is also refined. The models, their personalized combination and the assignment of the users to the subsets are estimated through solving an optimization problem. Our experimental evaluation on different real-world datasets shows that GLSLIM outperforms the standard global approach and also both latent space and item-item state-of-the-art approaches, on average by 9.29% and up to 17.37%.

- **Estimating multiple latent space models for top-$N$ recommendation.** Seeing the benefits of the multiple local item-item models, we extended this line of research to latent space top-$N$ recommendation approaches. Latent space approaches model the aspects which contribute to users' preferences in the form of latent factors. Though such a user-model has been shown to lead to good results, the aspects that different users care about can vary. In many domains, there may be a set of aspects for which all users care about and a set of aspects that are specific to different subsets of users. In order to capture this user model explicitly, we proposed two latent space models: rGLSVD and sGLSVD, that combine a

global and multiple user subset specific sets of latent factors. The rGLSVD model assigns the users into different subsets based on their rating patterns and then estimates a global and a set of user subset specific local models whose number of latent dimensions can vary. The sGLSVD model estimates both global and user subset specific local models by keeping the number of latent dimensions the same among these models but optimizes the grouping of the users in order to achieve the best approximation. Our experimental evaluation on different real-world datasets shows that the proposed approaches outperform significantly the global low-rank model as well as other competing latent space approaches for top-$N$ recommendation, on average by 13% and up to 37%.

- **Investigating and using the error in top-$N$ recommendation.** Different popular top-$N$ recommender methods, such as SLIM and PureSVD treat the missing entries as zeros. Thus, when recommending items that users have not yet consumed, they recommend items that are assumed to be 'disliked' by the user. Consequently, when recommending the missing entries with the highest predicted values, they essentially recommend the missing entries with the highest error. We believe that since the error drives the top-$N$ recommendation in these methods, it is important to look into what are the properties of the error, how they correlate with the top-$N$ recommendation quality, and how the performance of these algorithms can be improved by *shaping* their errors. We showed that users and items with similar ratings also have similar errors in their missing entries, and vice versa for SLIM and PureSVD. Also, among the different models that are estimated by changing their respective hyperparameters, the ones that achieve the best recommendation performance are those that display the closest rating-based and error-based similarities. Utilizing this insight, we developed a method, called ESLIM, which extends SLIM, by enforcing users with similar rating behaviors to also have similar error in their missing entries and likewise for the items. The method is shown to outperform SLIM, especially for predicting items that have been rated by few users (tail items).

## 9.2 Future research directions

In this work, we have taken different steps towards developing algorithms to improve the quality of top-$N$ recommendation. Here we outline some future research directions that stem from our work.

- Our work on higher-order sets was shown to be effective for the top-$N$ recommendation task. A possible next step is utilizing the proposed method for set recommendation, such as travel package recommendation, course catalog recommendation e.t.c.

- We showed that estimating multiple user-subset-specific latent space models allows us to learn better low-rank representations for users, which led to improvement of the top-$N$ recommendation quality. What would happen if the subsets were based on items, instead of users? Would the proposed method also improve the low-rank representations for items?

- After seeing that both the update of user subsets, and the different ranks among local models are great ways to learn better low-rank representations, an exciting future direction would be to combine both in the context of a *regularized* latent space model, such as regularized SVD. In this way, users would not all switch to the subset of higher rank, as this would be penalized.

- We can extend to multiple levels of local latent space models, instead of one as shown in this thesis, thus resulting in a hierarchical model.

- In Chapter 8, we saw that ESLIM improves the quality of top-$N$ recommendation for tail items (items that have not been rated by many users). In the future, we believe it would be useful to develop a method which will combine a SLIM model for the head items and ESLIM model for the tail items, in order to achieve bigger performance gains.

- A possible future direction would be to add the constraints that enforce user similarity matrices and error similarity matrices to be close in the loss function of other approaches, beyond SLIM. Such an example is PureSVD or another latent

space approach that treats the missing entries as zeros, in order to investigate their effect on the top-$N$ recommendation quality.

- The methods we developed have been applied in the recommendation domain. However, the main ideas we are contributing could have an impact on other domains as well. One such domain would be personalized medicine, where the users could be mapped to patients, and the items to drugs. Another domain is course recommendation, where the users would correspond to students, and the items to courses. It would be beneficial to research such applications.

- The methods developed were evaluated in terms of their accuracy as measured by the hit rate and the average reciprocal hit rank. Examining their performance with respect to a different measure, such as novelty or diversity is another interesting research direction.

- Our work was done in the context of utilizing implicit feedback data, since they are prevalent. It would be interesting to modify the proposed methods, to also handle additional data whenever they are available (such as social network information, or contextual side information) and examine how they would affect the top-$N$ recommendation performance.

In conclusion, the development of novel scalable methods which improve the top-$N$ recommendation quality and the insights that the analysis of the top-$N$ recommendation task provides have high impact on millions of people, and bring on exciting new directions.

# References


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

[2] Gediminas Adomavicius and Alexander 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 engineering*, 17(6):734–749, 2005.

[3] Dietmar Jannach, Paul Resnick, Alexander Tuzhilin, and Markus Zanker. Recommender systems—: beyond matrix completion. *Communications of the ACM*, 59(11):94–102, 2016.

[4] Charu C Aggarwal. *Recommender systems*. Springer, 2016.

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

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

[7] Evangelia Christakopoulou and George Karypis. Hoslim: higher-order sparse linear method for top-n recommender systems. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 38–49. Springer, 2014.

[8] Evangelia Christakopoulou. Moving beyond linearity and independence in top-n recommender systems. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 409–412. ACM, 2014.

[9] Evangelia Christakopoulou and George Karypis. Local item-item models for top-n recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 67–74. ACM, 2016.

[10] David C. Anastasiu, Evangelia Christakopoulou, Shaden Smith, Mohit Sharma, and George Karypis. Big data and recommender systems. *Novtica: Journal of the Spanish Computer Scientist Association*, (240), October 2016.

[11] Evangelia Christakopoulou, Shaden Smith, Mohit Sharma, Alex Richards, David Anastasiu, and George Karypis. Scalability and distribution of collaborative recommenders. In *Collaborative Recommendations: Algorithms, Practical Challenges and Applications*. World Scientific Publishing, 2018.

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

[13] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B Kantor. *Recommender systems handbook*. Springer, 2015.

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

[15] Vito Claudio Ostuni, Tommaso Di Noia, Eugenio Di Sciascio, and Roberto Mirizzi. Top-n recommendations from implicit feedback leveraging linked open data. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 85–92. ACM, 2013.

[16] Cataldo Musto, Pierpaolo Basile, Pasquale Lops, Marco De Gemmis, and Giovanni Semeraro. Linked open data-enabled strategies for top-n recommendations. In *CBRecSys@ RecSys*, pages 49–56, 2014.

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

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

[19] Xin Liu and Karl Aberer. Soco: a social network aided context-aware recommender system. In *Proceedings of the 22nd international conference on World Wide Web*, pages 781–802. International World Wide Web Conferences Steering Committee, 2013.

[20] Jiliang Tang, Xia Hu, Huiji Gao, and Huan Liu. Exploiting local and global social context for recommendation. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 2712–2718. AAAI Press, 2013.

[21] James Bennett, Stan Lanning, et al. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. New York, NY, USA, 2007.

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

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

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

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

[26] Hanhuai Shan and Arindam Banerjee. Generalized probabilistic matrix factorizations for collaborative filtering. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 1025–1030. IEEE, 2010.

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

[28] Fabio Aiolli. A preliminary study on a recommender system for the million songs dataset challenge. *Preference Learning: Problems and Applications in AI*, 1, 2012.

[29] Fabio Aiolli. Efficient top-n recommendation for very large scale binary rated datasets. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 273–280. ACM, 2013.

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

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

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

[33] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001.

[34] George Karypis. Evaluation of item-based top-n recommendation algorithms. In *Proceedings of the tenth international conference on Information and knowledge management*, pages 247–254. ACM, 2001.

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

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

[37] Yong Zheng, Bamshad Mobasher, and Robin Burke. Cslim: Contextual slim recommendation algorithms. In *Proceedings of the 8th ACM Conference on Recommender Systems*, pages 301–304. ACM, 2014.

[38] Xiaodong Feng, Ankit Sharma, Jaideep Srivastava, Sen Wu, and Zhiwei Tang. Social network regularized sparse linear model for top-n recommendation. *Engineering Applications of Artificial Intelligence*, 2016.

[39] Mark Levy and Kris Jack. Efficient top-n recommendation by linear regression. In *RecSys Large Scale Recommender Systems Workshop*, 2013.

[40] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1, 2010.

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

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

[43] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Application of dimensionality reduction in recommender system-a case study. Technical report, Minnesota Univ Minneapolis Dept of Computer Science, 2000.

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

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

[46] Rong Pan, Yunhong Zhou, Bin Cao, Nathan N Liu, Rajan Lukose, Martin Scholz, and Qiang Yang. One-class collaborative filtering. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 502–511. IEEE, 2008.

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

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

[49] Yue Shi, Alexandros Karatzoglou, Linas Baltrunas, Martha Larson, and Alan Hanjalic. Gapfm: Optimal top-n recommendations for graded relevance domains. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 2261–2266. ACM, 2013.

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

[51] Konstantina Christakopoulou and Arindam Banerjee. Collaborative ranking with a push at the top. In *Proceedings of the 24th International Conference on World Wide Web*, pages 205–215. International World Wide Web Conferences Steering Committee, 2015.

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

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

[54] Markus Weimer, Alexandros Karatzoglou, Quoc Viet Le, and Alex Smola. Maximum margin matrix factorization for collaborative ranking. *Advances in neural information processing systems*, pages 1–8, 2007.

[55] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *ACM SIGMOD Record*, volume 22, pages 207–216. ACM, 1993.

[56] Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994.

[57] Bamshad Mobasher, Honghua Dai, Tao Luo, and Miki Nakagawa. Effective personalization based on association rule discovery from web usage data. In *Proceedings of the 3rd international workshop on Web information and data management*, pages 9–15. ACM, 2001.

[58] Weiyang Lin, Sergio A Alvarez, and Carolina Ruiz. Efficient adaptive-support association rule mining for recommender systems. *Data Mining and Knowledge Discovery*, 6(1):83–105, 2002.

[59] Ayhan Demiriz. Enhancing product recommender systems on sparse binary data. *Data Mining and Knowledge Discovery*, 9(2):147–170, 2004.

[60] Weike Pan and Li Chen. Cofiset: Collaborative filtering via learning pairwise preferences over item-sets. In *Proceedings of the 2013 SIAM International Conference on Data Mining*, pages 180–188. SIAM, 2013.

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

[62] Mathieu Blondel, Akinori Fujino, Naonori Ueda, and Masakazu Ishihata. Higher-order factorization machines. In *Advances in Neural Information Processing Systems*, pages 3351–3359, 2016.

[63] Alex Beutel, Ed H Chi, Zhiyuan Cheng, Hubert Pham, and John Anderson. Beyond globally optimal: Focused learning for improved recommendations. In *Proceedings of the 26th International Conference on World Wide Web*, pages 203–212. International World Wide Web Conferences Steering Committee, 2017.

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

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

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

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

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

[69] Bin Xu, Jiajun Bu, Chun Chen, and Deng Cai. An exploration of improving collaborative recommender systems via user-item subgroups. In *Proceedings of the 21st international conference on World Wide Web*, pages 21–30. ACM, 2012.

[70] Lyle H Ungar and Dean P Foster. Clustering methods for collaborative filtering. In *AAAI workshop on recommendation systems*, volume 1, pages 114–129, 1998.

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

[72] Zijian Zheng, Ron Kohavi, and Llew Mason. Real world performance of association rule algorithms. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 401–406. ACM, 2001.

[73] Tom Brijs, Gilbert Swinnen, Koen Vanhoof, and Geert Wets. Using association rules for product assortment decisions: A case study. In *Proceedings of the fifth*

*ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 254–260. ACM, 1999.

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

[75] Flixster dataset. `http://http://www.cs.sfu.ca/~sja25/personal/datasets/`.

[76] Pang-Ning Tan et al. *Introduction to data mining*. Pearson Education India, 2007.

[77] Neil Hurley and Mi Zhang. Novelty and diversity in top-n recommendation–analysis and evaluation. *ACM Transactions on Internet Technology (TOIT)*, 10(4):14, 2011.

[78] Michael W Berry. Large-scale sparse singular value computations. *The International Journal of Supercomputing Applications*, 6(1):13–49, 1992.

[79] Guibing Guo, Jie Zhang, Zhu Sun, and Neil Yorke-Smith. Librec: A java library for recommender systems. In *UMAP Workshops*, 2015.

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

[81] Douglas C Montgomery and George C Runger. *Applied statistics and probability for engineers*. John Wiley & Sons, 2010.

[82] Masakazu Seno and George Karypis. Lpminer: An algorithm for finding frequent itemsets using length-decreasing support constraint. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 505–512. IEEE, 2001.

[83] George Karypis. Cluto-a clustering toolkit. Technical report, MINNESOTA UNIV MINNEAPOLIS DEPT OF COMPUTER SCIENCE, 2002.

[84] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.