Towards Better User Preference Learning for Recommender Systems

by

Yao Wu

M.Sc., Chinese Academy of Sciences, 2012
B.Sc., University of Science and Technology of China, 2009

Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

in the
School of Computing Science
Faculty of Applied Science

SIMON FRASER UNIVERSITY
Summer 2016

Approval

Name: **Yao Wu**

Degree: **Doctor of Philosophy (Computing Science)**

Title: ***Towards Better User Preference Learning for Recommender Systems***

Examining Committee:

Chair: Ze-Nian Li
Professor
School of Computing Science

Martin Ester
Senior Supervisor
Professor
School of Computing Science _____

Jian Pei
Supervisor
Professor
School of Computing Science _____

Ke Wang
Internal Examiner
Professor
School of Computing Science _____

George Karypis
External Examiner
Professor
Department of Computer Science & Engineering
University of Minnesota, Twin Cities _____

Date Defended: August 11, 2016

Abstract

In recent years, recommender systems have become widely utilized by businesses across industries. Given a set of users, items, and observed user-item interactions, these systems learn user preferences by collective intelligence, and deliver proper items under various contexts to improve user engagements and merchant profits. Collaborative Filtering is the most popular method for recommender systems. The principal idea of Collaborative Filtering is that users might be interested in the items that are preferred by users with similar preferences. Therefore, learning user preferences is the core technique of Collaborative Filtering.

In this thesis, we study new methods to help us better understand user preferences from three perspectives. We first dive into each rating that users give on the items, and study the reasons behind the ratings by analyzing user reviews. We propose a unified model that combines the advantages of aspect-based opinion mining and collaborative filtering, which could extract latent aspects and sentiments from reviews and learn users' preferences of different aspects of items collectively. In our next work, we study the problem from each user's perspective, and propose a general and flexible model that embraces several popular models as special cases. The new model achieves better top-N recommendation results on several popular data sets. Finally, we study how to utilize the general structure of the user-item matrix to better apply collaborative filtering methods. We propose a co-clustering based method that first partitions the users and items into several overlapping and focused subgroups, and then applies collaborative filtering methods within each subgroup. The final recommendations for users are aggregated from the results from the subgroups they are involved in. Experimental results show that this method could produce better recommendations than other co-clustering methods and methods that directly apply collaborative filtering on the original user-item matrix.

Keywords: Recommender Systems; Personalization

Dedicated to my parents and Gengchun.

Acknowledgements

First and foremost, I want to thank my advisor Dr. Martin Ester, for his supervision throughout my study at SFU. Working with Martin has always been enjoyable – I had full freedom of choosing the research topics that I am really passionate about, and I could always get insightful feedback from him. Besides his guidance on my research, he also encouraged me to do internships during summer time, and helped me a lot on building my career. Additionally, I would like to thank the rest of the examining committee: Dr. Jian Pei, Dr. Ke Wang, Dr. George Karypis and Dr. Ze-Nian Li. Thank you all for your feedback and guidance that significantly improved this thesis.

I am sincerely grateful to all my mentors, collaborators and friends who have helped me throughout my study and career so far, and/or have contributed to the content of this thesis.

Special thanks to my parents and Gengchun for their support all the way.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In recent years, recommender systems have become widely utilized by businesses across industries, and have changed the way how users discover new items. For example, we have noted how Amazon or similar on-line vendors strive to present each returning users with some suggestions of products that they might like to buy. These suggestions are not randomly chosen, but are based on the purchasing decisions by similar customers or some other techniques we will discuss in this thesis. Recommender systems have contributed a large proportion of the traffic and revenue to a large number of on-line services. To give an example, LinkedIn's recommendation systems power more than 50% of the social connections and job applications that are created by their users, and more than 80% of a user's LinkedIn homepage feed is generated by recommendation [42].

Recommender systems use a number of different methodologies. We can generally classify these methods into the following two categories [1] [27] :

- **Content-base Filtering** methods make recommendations by examining the contents/properties of items and user profiles. For instance, if a YouTube user has watched many basketball videos before, recommending him some more videos with the same genre is a straightforward solution.

- **Collaborative Filtering** methods are based on analyzing a large amount of user behaviors and predict user preferences like by collective intelligence. The principal idea of Collaborative Filtering is that users might be interested in items that are favorited by users sharing similar tastes in the past.

Usually, Content-based approaches rely on high quality contents that could accurately describe users and items in order to make good recommendations. Collaborative Filtering approaches do not require these contents and therefore it is capable of recommending complex items like movies and musics without requiring an understanding of the items themselves. In this thesis, we focus our discussion on the Collaborative Filtering approaches as they have been shown to be able to produce

[1] Real recommender systems might also use some hybrid methods mixing both of them in production.

Table 1.1: A sample utility matrix, representing users' ratings of movies.

	Avatar	**The Godfather**	**The Shawshank Redemption**	**Transformers**	**Star Wars**
Alice		5	4		
Bob				4	5
Chris		5	4		
David	4		1	5	5

more accurate recommendations and have been widely adopted in most successful recommender systems like Amazon, Netflix, YouTube and LinkedIn.

In a recommender system application, there are two main classes of entities – *users* and *items*. The input data is usually represented as a utility matrix. For each user-item pair, the corresponding element in the matrix represents the degree of preference of that user for that item. Each user only knows or has consumed a small subset of all the available items. Therefore, the utility matrix is usually sparse, meaning that most entries are *missing* or *unknown*. An unknown rating implies that we have no information about the user's preference for the item. Table 1.1 shows a sample utility matrix for a movie recommender system. We call the available ratings in the matrix as *observed* values. The blanks represent the situation where the user has not rated the movie, and therefore we do not whether they like the item or not. We call them as *missing* values. The goal of recommender systems is to predict the missing values and pick those with highest predicted values to recommend to users.

Collaborative Filtering methods can be further classified to memory-based and model-base approaches. Memory-base approaches typically use instant-based methods, such as variants of k-Nearest-Neighbors on users and items. Although simple, memory-based approaches are still widely used in a lot of real world systems, since they are easy to implement, scale and interpret. Model-based approaches apply machine learning models to learn the relationship of the inputs (e.g., users, items, contexts and features) and desired outputs (e.g., ratings, likes, purchases).

1.1 Research Questions and Contributions

In this thesis, we mainly study the following three research questions raised around the utility matrix in Table 1.1. Our goal is to better understand user preferences from different perspectives.

1.1.1 Research question from element-wise perspective

If we zoom into each element of the matrix, is a numeric rating good enough to describe user's preference?

We highlight this problem in Table 1.2. Both Alice and Chris have rated the movie *The Shawshank Redemption* with 4 stars, but they might have different reasons – Alice might like its story, while Chris just likes the movie star *Morgan Freeman*. If we only use the ratings as input, rec-

Table 1.2: Element-wise perspective.

	Avatar	The Godfather	The Shawshank Redemption	Transformers	Star Wars
Alice		5	4		
Bob				4	5
Chris		5	4		
David	4		1	5	5

ommender systems would treat the two ratings equally. However, Alice is expecting some new movies with great stories, and Chris is hoping to watch more movies by *Morgan Freeman*. How can we distinguish the differences between them? This question drives us to explore the possibility of whether we could understand the ratings in a finer level. Fortunately, we have abundant user reviews available. For example, users like to provide reviews on items they gave bought or consumed before on Yelp, Amazon, eBay, etc. In these reviews, they prefer to write their feedback on different aspects of the items. Therefore, we can learn these latent opinions from the reviews and predict users' feedback on aspects of some new items.

While previous works on Collaborative Filtering mainly focus on the ratings themselves, few papers have studied how to utilize the reviews to extract fine-grained user feedback from the reviews. The main challenge is that users' opinions are implicitly expressed in the texts, but not explicitly provided as numeric ratings. In order to achieve this goal, we need to first mine the latent aspects and opinions from the texts. On the other hand, review understanding has been well studied by Aspect based Opinion mining, of which the main goal is to extract the latent aspects and latent sentiments from the reviews. But existing works on Aspect-based Opinion Mining only work on review-level or product-level. The main limitation is that before they could infer a user's opinions on different aspects of an item, they need the review as input. Here comes one problem: how can we predict a user's preference on a new item which he has not rated/reviewed yet? Interestingly, predicting user preferences on future items is just what Collaborative Filtering does.

Given above motivation, in Chapter 3, we propose a unified probabilistic model called *Factorized Latent Aspect ModEl* (FLAME), which combines the advantages of both collaborative filtering and aspect-based opinion mining so that the two methods can mutually enhance each other. In the new framework, we use opinion mining techniques to help us perform collaborative filtering on a finer level, and meanwhile, we use collaborative filtering to help better extract the latent aspects and latent opinions, and empower the opinion mining techniques to be able to predict future preferences.

We empirically evaluated the proposed FLAME on a hotel review data set from TripAdvisor[2] and a restaurant review data set from Yelp[3]. Experimental results show that FLAME can effectively extract meaningful aspects and predict aspect ratings of a user on new items.

[2] `http://www.tripadviosr.com`
[3] `http://www.yelp.com`

1.1.2 Research question from row-wise perspective

*If we look through a user's view (*i.e.*, a row in the matrix), what is personalized recommendation in principle?*

Table 1.3: Row-wise perspective.

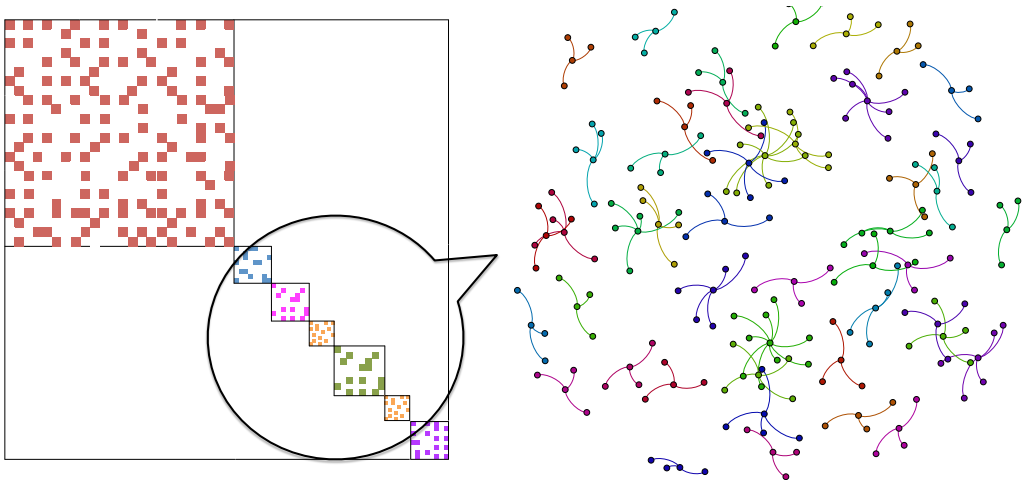
	Avatar	The Godfather	The Shawshank Redemption	Transformers	Star Wars
Alice		5	4		
Bob				4	5
Chris		5	4		
David	4		1	5	5

We highlight the problem in Table 1.3. For a user in the recommender systems, we have his historical behaviors, such as ratings and clicks, and we want to predict his future behaviors on the items that he has not seen or taken actions. These items which could maximize user engagement or some other business goals are selected to recommended to users. This problem is known as Top-N Recommendation. For the user Bob, we know that he likes Sci-Fi movies *Transformers* and *Star Wars*. The question is what other movies he would also like.

This is the most common task in recommender systems, and a lot of work has been done to solve it. In this thesis, we study this problem from a new perspective and propose a generalized solution that embraces some popular methods as special cases.

To make the problem easier to understand, we focus our discussion on the implicit feedback data, but it is worth noting that the techniques that we will discuss and propose also work for explicit feedback data with slight modifications. In an implicit feedback data set, we are given a subset of items that a user liked before. The task of recommender systems is to find other items he might also like. We connect this problem with the Denoising Auto-Encoder, which uses intentional corruptions of the input during training time. We propose a model called Collaborative Denoising Auto-Encoder (CDAE). During the training time, we intentionally dropout a random subset of a user's favorite item set, and trains the neural network to be able to reconstruct the whole set. When making recommendation, we use the user's current favorite set as input, in order to find some other potentially interesting items for him.

The proposed CDAE model is very flexible that it embraces several popular models such as Latent Factor Models [23] and SVDPP [22] special cases. The flexibility and the denoising trick enable CDAE to achieve better performance on top-N recommendation. We carry out comprehensive experiments on the selection of the components and compare CDAE with some state-of-the-art methods. The details will be shown in Chapter 4.



(a) Yelp dataset Matrix (b) Yelp dataset Graph

Figure 1.1: Structures of Yelp dataset [79]. In the left is the exampled structure of the rating matrix, and in the right is the real structure of the scattered blocks.

1.1.3 Research question from matrix-wise perspective

If we step back and take a look at the whole matrix, can we do better job on recommendation by leveraging the general structure of the matrix?

Table 1.4: Matrix-wise perspective.

	Avatar	**The Godfather**	**The Shawshank Redemption**	**Transformers**	**Star Wars**
Alice		5	4		
Bob				4	5
Chris		5	4		
David	4		1	5	5

We highlight this problem in Table 1.4. From the example, we can see that there are two main groups of users with different tastes, i.e., Bob and David like *Sci-Fi* movies while Alice and Chris prefer to watch *classic* movies.

Typical Collaborative Filtering methods measure users' preferences by their historical behaviors over the entire item space. For example, the user-based nearest neighbors method measures the similarity between pairs of users by their preferences on all the items [46]; Matrix Factorization decomposes the whole user-item matrix into two low-rank sub-matrices where the collective intelligences are represented as latent factors in the model [23]. However, this assumption does not always hold, especially when the underlying system includes a large set of items of different types.

An illustrating example on the Yelp rating dataset [4] can be found in [79]. By permuting the rows and columns of the rating matrix, it can be rearranged into a BDF structure with 53 diagonal blocks,

[4] `https://www.yelp.com/dataset_challenge`

with a dominating block and 52 scattered blocks, which is shown in Figure 1.1. This is because of the fact that users and items in the data set are located at different places, and people living in Phoenix are more likely to visit local restaurants rather than a restaurant in New York. Thus users and items are organized as subgroups, where each subgroup includes a subset of users and items which are likely to be linked together.

Generally, these subgroups are not disjointed, but heavily overlapping with each other. The main reason is that users typically have multiple interests and share the same tastes on different things with different group of people. For instance, one might share the same taste on movies with his family, while liking the similar musics with his friends. People with similar tastes tend to like the the same subset of items. Therefore, there exists several overlapping co-clusters in the user-item matrix, where users and items within the same co-cluster are more connected than with other users/item outside the co-clusters.

In Chapter 5, we proposed a new co-clustering methods CCCF for collaborative filtering. Comparing to previous work, CCCF has several advantages such as scalability, flexibility, interpretability and extensibility. We experimentally compare CCCF with other co-cluster methods and the methods that directly apply collaborative filtering methods to the original user-item matrix, and the results demonstrate the superiority of CCCF.

1.2 Thesis Organization

The rest of this thesis is organized as follows. In Chapter 2, we present some background knowledge on recommender systems, including the problem definition, evaluation measures and main techniques. We also discuss some related work that tries to improve recommender systems from the three perspectives we mentioned above. In Chapter 3, we describe the FLAME model in details – including the motivation, proposed methodology and inference algorithms, as well as some experimental results on two real world data sets. We further point out some potential applications by better understanding user preferences. In Chapter 4, we study the generic problem of top-N recommendation. With the motivation that we have mentioned above, we propose a generalized framework CDAE based on neural networks. The effectiveness of CDAE is verified on several popular recommendation benchmarks. In Chapter 5, we propose a new co-clustering method CCCF for better performing Collaborative Filtering methods. We conclude this thesis in Chapter 6 with a summary of our contributions and some future directions that are worth exploring.

Chapter 2

Background

In this chapter, we will first present some preliminary knowledge on the research of recommender systems, and then discuss some related work that tries to solve similar problems with ours.

2.1 Preliminaries

What is a Recommender System? – A Recommender selects a product that if acquired by the "buyer" maximizes value of both "buyer" and "seller" at a given point in time [42].

2.1.1 Notations

In the following, we use $\mathscr{U} = \{u|u = 1,...,U\}$ to denote the set of users, and $\mathscr{I} = \{i|i = 1,...,I\}$ to denote the items. The elements of the utility matrix are represented as $\mathscr{M} = \{(u,i,y_{ui})|(u,i) \in \mathscr{O}\}$, where $\mathscr{O}$ is the set of observed elements, and y_{ui} is user u's feedback on item i. The remaining elements in the matrix $\bar{\mathscr{O}} = \{\mathscr{U} \times \mathscr{I}\} \setminus \mathscr{O}$ are the missing values that we need to predict. Let $\mathscr{O}_u$ denote the set of item preferences in the training set for a particular user u, and $\bar{\mathscr{O}}_u$ is the unobserved preferences of user u. Items in $\bar{\mathscr{O}}_u$ are the candidates to be considered to recommended to user u.

In the rest of thesis, we use u to index a user, and i and j to index items. Vectors and matrices are denoted by bold symbols, where symbols in lower case (*e.g.*, $\boldsymbol{x}$) represent vectors and symbols in upper case (e.g., $\boldsymbol{X}$) represent matrices. Unless stated differently, $\boldsymbol{x}_i$ also represents a vector where i is used to distinguish different vectors. We denote the i-th row of matrix $\boldsymbol{X}$ by $\boldsymbol{X}_i$ and its (i,j)-th element by $\boldsymbol{X}_{ij}$.

2.1.2 Tasks and Evaluation Measures

Before we start discussing the methodologies, it is essential to make our target clear, i.e., what goals the recommender systems should achieve. Ideally, the best way to evaluate the recommender systems is to carry out online A/B testing experiments on real systems with a large amount of user, measuring the business metrics after deploying different methods. However, online A/B testing is

usually very expensive and might hurt user experience if the tested models are not optimal. Thus, offline evaluation is a necessary step to compare candidate methods before doing large scale online tests. In this subsection, we will review three popular offline evaluation protocols for recommender systems.

Rating Prediction

The goal of a recommender system is to predict the blanks in the utility matrix. A straightforward evaluation method is to examine how the system performs on predicting the ratings in the utility matrix. *Rating prediction* is widely used in most of the early works of recommender systems and some competitions such as Netflix Prize and KDD CUP contests. It works as follows. We randomly hold out a small set of ground-truth ratings in the observed set, train the models on the remaining subset, and then evaluate the models on the held-out set in terms of the prediction errors of the ratings.

Two commonly used evaluation measures are the Root of Mean Square Error (RMSE) and Mean Absolute Error (MAE):

$$\text{RMSE} = \sqrt{\frac{\sum_{(u,i)\in\mathscr{T}} \left(y_{ui} - \hat{y}_{ui}\right)^2}{|\mathscr{T}|}}, \tag{2.1}$$

$$\text{MAE} = \frac{\sum_{(u,i)\in\mathscr{T}} |y_{ui} - \hat{y}_{ui}|}{|\mathscr{T}|}, \tag{2.2}$$

where $\mathscr{T}$ is the set of held-out ratings.

Item Ranking

Rating prediction considers the square loss or variants as a measure of prediction accuracy. In reality, for a 5-point rating scale, predicting a true 5 as a 3 is often more costly than predicting a 3 as a 1, although their square loss is the same. More generally, what matters in a recommendation system is the ranking of items, especially the ranking performance at the top of the list, i.e., the items the user will actually see.

Several Collaborative Ranking models (e.g., [65, 25]) have been proposed to model the relative orders of items. Similar to rating prediction, they also hold out a subset of items from the observed set as test data. Instead of training a point-wise regression/classification model, they apply Learning to Rank techniques to model the relative orders of items for each user. At last, the models are evaluated on the test data by some popular measures for Information Retrieval, such as AUC [1], NDCG [2], MAP [3], etc.

[1] `https://www.kaggle.com/wiki/AreaUnderCurve`
[2] `https://www.kaggle.com/wiki/NormalizedDiscountedCumulativeGain`
[3] `https://www.kaggle.com/wiki/MeanAveragePrecision`

Top-N Recommendation

In recent years, top-N recommendation has become a more standard approach to evaluate the performance of recommendation methods. For top-N recommendation, we still hold out a subset of items in the observed set as test set. But differently, we evaluate the models by ranking the items in both of the test set and the missing set, and examining the Precision and Recall or other metrics like MAP, NDCG at the top positions of the list. The main idea is that a good recommender model should rank the items in the test set higher than the items in the missing set.

For each user, given a top-N recommendation list $C_{N,\text{rec}}$, precision and recall are defined as

$$\begin{aligned} \text{Precision@}N &= \frac{|C_{N,\text{rec}} \bigcap C_{\text{adopted}}|}{N} \\ \text{Recall@}N &= \frac{|C_{N,\text{rec}} \bigcap C_{\text{adopted}}|}{|C_{\text{adopted}}|}, \end{aligned} \tag{2.3}$$

where C_{adopted} are the items that a user has adopted in the test data. The precision and recall for the entire recommender system are computed by averaging the precision and recall over all the users, respectively.

Average precision (AP) is a ranked precision metric that gives larger credit to correctly recommended items in top ranks. AP@N is defined as the average of precisions computed at all positions with an adopted item, namely,

$$\text{AP@}N = \frac{\sum_{k=1}^{N} \text{Precision@}k \times \text{rel}(k)}{\min\{N, |C_{\text{adopted}}|\}}, \tag{2.4}$$

where Precision(k) is the precision at cut-off k in the top-N list $C_{N,\text{rec}}$, and rel(k) is an indicator function equaling 1 if the item at rank k is adopted, otherwise zero. Finally, Mean Average Precision (MAP@N) is defined as the mean of the AP scores for all users.

2.2 Related Work

2.2.1 Collaborative Filtering

Most machine learning models can be specified through two components: **model definition** and **objective function** during training. The model definition formulates the relationship between the inputs (*e.g.*, user ids, item ids, interactions, other features, etc.) and outputs (ratings or implicit feedback of items). The objective function is what the training process optimizes to find the best model parameters.

Model-based Collaborative Filtering methods can also be seen in this way. We will review several commonly used models and objective functions for modern recommender systems.

Recommender Models

In general, recommender models are defined as

$$\hat{y}_{ui} = \mathscr{F}_{\boldsymbol{\theta}}(u, i), \tag{2.5}$$

where $\hat{y}_{ui}$ is the predicted preference of user u on item i, and $\boldsymbol{\theta}$ denotes the model parameters we need to learn from training data. Different choices of the function $\mathscr{F}_{\boldsymbol{\theta}}$ correspond to different assumptions about how the output depends on the input.

Latent Factor Models (LFMs) is a family of popular models used for recommender systems. They represent the preference $\hat{y}_{ui}$ as the dot product of latent factor vectors $\boldsymbol{v}_u$ and $\boldsymbol{v}_i$, representing the user and the item, respectively [23, 48]. [4]

$$\hat{y}_{ui} = \boldsymbol{v}_u^\top \boldsymbol{v}_i \tag{2.6}$$

LFMs are originated from Matrix Factorization (or Matrix Completion), which usually assume the user-item utility matrix is generated from a low-rank underlying score matrix. With a small portion of the elements that we have observed, we could recover the score matrix under some conditions [8]. With the great success of the Matrix Factorization during the Netflix Prize, it has been extended to a lot of variants in order to consider other information. Some of these variants could not be explained by the low-rank theory any more, but they all share the same idea – using use latent factors/vectors to represent users and items.

Feature-aware Models

Typical LFMs use only the utility matrix as input, without knowing what the items/users themselves are. In some application scenarios, there are informative user/item features that can help us improve the recommendation. For example, in the application of *article* recommendation, the content of the articles is importance information because users usually like articles with specific topics; in the application of *friend* recommendation on social networks, the demographic information (e.g., *location*, *education*, *affiliation*, etc.) is important to infer the relationship between users. Furthermore, features are helpful to solve the cold-start problem [5] by incorporating prior knowledges into the model, e.g., two users from the same company are more likely to know each other than a randomly chosen pair of users.

There are several ways to extending the above models to consider features. Here we review a generalized model built on the Latent Factor Model. Other models can also be extended in a similar manner.

[4] To simplify notation, we assume that the latent factors are padded with a constant to model the bias.

[5] The cold-start problem in recommender systems is that for new users/items we do not have enough historical data to apply Collaborative Filtering methods to make accurate predictions for them.

We use $\boldsymbol{x}_u$ ($\boldsymbol{y}_i$) denote the feature vector of user u (item i). The feature vector can be a sparse binary vector of categorical features (e.g., location, education), a TF-IDF representation of the words in a document, a low-dimensional representation of an image from a Deep Learning framework, or a combination of them.

Bilinear Model. The preference $\hat{y}_{ui}$ is a result of a bilinear model:

$$\hat{y}_{ui} = \boldsymbol{x}_u^\top \boldsymbol{W} \boldsymbol{y}_i. \tag{2.7}$$

Here $\boldsymbol{W}$ is a coefficient matrix modeling the interactions between the features from both users and items. In most cases, the feature vector is high dimensional that the coefficient matrix could be too large to fit in memory or has too many parameters that might be easily over-fitted to the training data. To solve this problem, one can use two low-dimensional sub-matrices to approximate the coefficient matrix with the similar idea of Latent Factor Model.

$$\hat{y}_{ui} = \boldsymbol{x}_u^\top \boldsymbol{U}^\top \boldsymbol{V} \boldsymbol{y}_i. \tag{2.8}$$

Some advanced feature transformations [10] and non-linear kernels [67] can be further applied to enhance the power of the model. We can also use one-hot encoding [6] representation of user/items ids as input feature, and then the Latent Factor Models can be thought as a special case. Factorization Machine [43] can also be formulated as a special case of Bilinear Model.

Context-aware Models

In many applications, one might benefit from incorporating contextual information (such as time, location, browser session, etc.) into the recommendation process in order to recommend right items to users in certain circumstances. One approach is to consider the interaction between the user, the item and the context using a tensor factorization model:

$$\hat{y}_{uic} = \boldsymbol{v}_u \bullet \boldsymbol{v}_i \bullet \boldsymbol{v}_c \tag{2.9}$$

where c is the current context such as a time index, and $\boldsymbol{v}_c$ is its latent factor.

Objective Functions for Recommenders

Objective functions for training recommender models can be roughly grouped into three groups: point-wise, pair-wise and list-wise. Pair-wise objectives approximates ranking loss by considering the relative order of the predictions for pairs of items. Point-wise objectives, on the other hand, only depend on the accuracy of the prediction of individual preferences. List-wise objectives consider the list of items as a whole and optimize some metrics that focus on the overall quality of list.

[6] https://en.wikipedia.org/wiki/One-hot

We note that regardless of the choice of a pair-wise, point-wise or a list-wise objective function, if one aims at making satisfactory top-N recommendations, it is critical to properly take unobserved feedback into consideration within the model. Models that only consider the observed feedback fail to account for the fact that ratings are *not* missing at random. These models are not suitable for top-N recommendation [40, 55].

2.2.2 Topic Modeling and Opinion Mining

Topic models are introduced by [6] for learning the hidden dimensions of text. The basic assumption of topic models is that documents are represented by mixtures of some latent topics where topics are associated with a multinomial distribution over words of a vocabulary.

The earliest work integrating collaborative filtering with topic model is CTM [59], which is proposed for article recommendation. CTM simultaneously trains a topic model on the collection of articles and a latent rating factor model on the ratings of users on articles, while assuming that the latent factors of items depend on the latent topic distributions of their text.

Much work has been proposed to help users digest and exploit large number of reviews by aspect-based opinion mining techniques, including information extraction from reviews [30], uncovering the latent aspects of the review sentences [33], inferring the latent aspect ratings and aspect weights of each review document [61, 62], aspect-based review summarization for products [30, 28], etc. These methods either focus on *review-level* analysis (extracting useful information within each review) to help users easily find what they need from a piece of review, or make *product-level* summarization (aggregating the opinions of all the users) to provide an overview of users' feedback on a product. Among above work, the Latent Aspect Rating Analysis Model (LARAM) proposed in [61, 62] is most closest one for our purpose. LARAM assumes the overall rating of a review is generated by a weighted sum of the latent aspect ratings, which are generated from the words and the latent topic allocations of the words by a linear regression function. LARAM learns the latent aspect ratings for each review and aspect weights for each reviewer. The weights represent the importance of the aspects for a reviewer, and can support a wide range of application tasks, such as aspect-based opinion summarization, personalized entity ranking and recommendation, and review behavior analysis.

HTF [32] is the first work that tries to understand user ratings in recommender systems by utilizing review texts. HFT considers latent rating factors of an item as the properties that the item possesses, and assumes that if a product exhibits a certain property (higher latent rating factor value), this will correspond to a particular topic being discussed frequently (higher probability in topic distribution) [32]. HTF first aggregates all the reviews of an item into a single document, and uses a similar method as Collaborative Topic Regression model [59] to train a topic model and a latent factor model together. Experiments show HFT can not only predict product ratings more accurately, but can be also used to facilitate tasks such as genre discovery and to suggest informative reviews.

Explicit Factor Model (EFM) [76] uses aspect-based opinion mining techniques explain the ratings and recommendations explicitly using the aspect features. It first extracts explicit product features (i.e. aspects) and user opinions by phrase-level sentiment analysis on user reviews, and then models a user's rating as a summation of two parts – the utility score on these aspects, and the utility on the items as the Latent Factor Models do. An important advantage of utilizing explicit features for recommendation is its ability to provide intuitional and reasonable explanations for both recommended and dis-recommended items.

2.2.3 Co-Clustering

Partitioning users and items into subgroups for CF has been studied by several previous works, where user clustering [70], item clustering [39] and user-item co-clustering [14] methods have been proposed to boost the performance of collaborative filtering. However, in these methods each user/item is only allowed to be assigned to a single subgroup, so that they are not able to model the case where users have multiple interests. To address this problem, several other papers [52, 63, 64] extend the Mixed Membership Stochastic Blockmodels (MMSB) [2] to allow mixed memberships.

In [69], the authors propose a unified framework for improving collaborative filtering via overlapping co-clustering, which can be employed for top-N recommendation. It works as follows: 1) Users and items are first grouped into multiple overlapping subgroups with different weights. 2) After obtaining the subgroups, any traditional CF method can be applied to each subgroup separately to make the predictions for the users on their unrated items in the same subgroup. 3) The final recommendation list for a user is aggregated from the results in the subgroups that she/he is involved in. A co-clustering method based one Spectral Clustering and fuzzy K-Means is proposed to learn these subgroups.

The authors of [77] adopt a similar framework, and propose to partition the user-item matrix by permuting the rows and columns of the user-item matrix to form Approximate Bordered Block Diagonal Form (ABBDF) matrices. Two density-based algorithms are designed to transform sparse user-item utility matrix into ABBDF structures. They propose a general Collaborative Filtering framework based on these structures to make rating predictions. They also proposed a unified model LFM [78] that learns ABBDF and Latent Factor Model simultaneously.

The framework used in [69, 77] has several benefits. 1) It partitions the matrix into several overlapping sub-matrices, which are denser than the original matrix, thus making the CF methods more feasible. 2) Any CF method can be used in each subgroup. Since the performance of different methods varies under different scenarios, this allows users to select their favorite CF base methods for the subgroups. 3) The training of the CF methods in subgroups can be trivially parallelized.

Chapter 3

FLAME: On Fine-grained Rating Understanding

Typical collaborative filtering methods take the numeric overall ratings as inputs [23, 48], assuming that users with the same ratings share the same tastes. However, two users who have assigned the same 4 stars to a restaurant might have significantly different reasoning; one might like its *food* while the other likes its *service*. Here comes a research problem: *can we understand the reasons behind a rating*? In this chapter, we propose a model called FLAME that tries to solve this problem by utilizing users reviews to help us better understand user preferences on items.

3.1 Background and Overview

Nowadays, products and services offered on most online E-commerce websites are accompanied by abundant user-generated reviews, which can help users make better decision. For instance, if a user wants to know more about the battery life of a laptop, comments on the battery life of this specific laptop by other users are more reliable than those given in the official description of the product. However, the volume of reviews grows so rapidly that it gets extremely difficult for users to find useful information in short time. Thus mining useful information out of these huge amount of reviews has become an important way to improve user satisfaction of these online E-commerce websites.

Aspect-based opinion mining [29] has attracted a lot of attention recently. Given a collection of reviews on a set of items, aspect-based opinion mining methods extract major aspects out of every item based on how often they have been commented by users, and learn users' sentiment polarities toward each aspect based on the opinionated words they used in the reviews. Figure 3.1 shows a sample review from Amazon[1]. The user assigns a 5-star overall rating, and expresses his opinions on several aspects of the product. From a set of reviews like this, aspect-based opinion mining

[1] `http://www.amazon.com`

Figure 3.1: A Sample Review On Amazon

methods can automatically extract the aspects of the product, such as *performance*, *display*, *value* and *size*, as well as infer latent sentiment scores for each aspect, e.g., 5 stars on its *display*.

Much work has been proposed to help users digest and exploit large number of reviews by aspect-based opinion mining techniques, including information extraction from reviews [30], uncovering the latent aspects of the review sentences [33], inferring the latent aspect ratings and aspect weights of each review document [61, 62], aspect-based review summarization for products [30, 28], etc. These methods either focus on *review-level* analysis (extracting useful information within each review) to help users easily find what they need from a piece of review, or make *product-level* summarization (aggregating the opinions of all the users) to provide an overview of users' feedback on a product. However, an important factor is typically ignored – *preference diversity*, i.e., users have different preferences that their opinions on the same item may differ from each other. For example, the *food* of the same restaurant might be delicious for some users but terrible for others. When choosing restaurants, a user might want to know whether a restaurant meets his *own* expectation on the aspect of *food*. But when facing a large number of reviews expressing various opinions, it becomes extremely difficult for the user to make the decision: 1) it's impossible for the user to read all the reviews even if the fine-grained *review-level* analysis is provided. 2) the user has no idea of which reviews are more reliable or which reviewers share similar tastes with him. 3) *product-level* summarization is also unreliable since it is generated from reviews by users with different tastes. To help users better utilize the existing reviews, we argue that a new method is required, which can learn a user's personalized preferences on different aspects from his past reviews of other items, and predict his preferences on the aspects of a given item by mining the opinions by other users with similar preferences.

In this chapter, we introduce the problem of *Personalized Latent Aspect Rating Analysis*. Given a collection of reviews of a set of items by a set of users, the goal is to solve the following two tasks: *a*) learn the latent aspects and their word distribution over a pre-defined vocabulary, and the latent aspect ratings for each review; *b*) for any user u in the data set, predict the latent aspect ratings on the items that he has not yet reviewed. Existing aspect-based opinion mining methods such as [61, 62, 34] are able to solve task *a*, but are unsuitable for solving task *b* since they require the text

of user u's review for item i as input. Task b is also different from the well-studied rating prediction problem in recommender systems, the goal of which is to predict the overall rating while we want to predict the aspect ratings.

To address the problem of Personalized Latent Aspect Rating Analysis, we propose a unified probabilistic model called *Factorized Latent Aspect ModEl* (FLAME), which combines the advantages of both collaborative filtering and aspect-based opinion mining so that the two methods can mutually enhance each other. The general idea of FLAME is that we can learn users' preferences based on their past reviews, so that we can collaboratively predict a user's preference of an aspect of an item from the opinions of other users with similar tastes. FLAME improves existing aspect-based opinion mining methods by being able to infer aspect ratings of users on new items[2], and enhances collaborative filtering methods by leveraging reviews to analyze users' preferences on different aspects of items.

We empirically evaluate the proposed FLAME on a hotel review data set from TripAdvisor[3] and a restaurant review data set from Yelp[4]. Experimental results show that FLAME can effectively extract meaningful aspects and predict aspect ratings of a user on new items to him.

3.2 Related Work

In this section, we discuss the connections and differences between related work and FLAME. Our work is related to two research topics: Collaborative Filtering and Aspect-based Opinion Mining.

3.2.1 Collaborative Filtering

Collaborative filtering (CF) is a popular method widely used in recommender systems. The assumption behind collaborative filtering is that a given user is more likely to like items that are liked by other users with similar tastes. Various state-of-the-art CF methods are based on latent factor models [23]. Latent factor models assume that a user's rating on a particular item depends on the inner dot product of the latent user factors and the latent item factors.

Some work combining collaborative filtering with *Topic Models* has been proposed to leverage text information in recommender systems. Topic models are introduced by [6] for learning the hidden dimensions of text. The basic assumption of topic models is that documents are represented by mixtures of some latent topics where topics are associated with a multinomial distribution over words of a vocabulary. The earliest work integrating collaborative filtering with topic model is CTM [59], which is proposed for article recommendation. CTM simultaneously trains a topic model on the collection of articles and a latent rating factor model on the ratings of users on articles, while assuming that the latent factors of items depend on the latent topic distributions of their text. A recent work [32] proposes a model called HFT, which aims at improving collaborative filtering using

[2]Note that *new items* for a user are the items that he has not rated yet.

[3]`http://www.tripadviosr.com`

[4]`http://www.yelp.com`

reviews. HFT considers latent rating factors of an item as the properties that the item possesses, and assumes that if a product exhibits a certain property (higher latent rating factor value), this will correspond to a particular topic being discussed frequently (higher probability in topic distribution) [32]. HTF first aggregates all the reviews of an item into a single document, and uses a similar method as CTM to train a topic model and a latent factor model together.

Different from CTM and HTF which learn topic distributions for each item, our approach learns for each review its aspect distribution *as well as* its rating distribution on each aspect.

3.2.2 Aspect-based Opinion Mining

The main task of aspect-based opinion mining is extracting the aspects and learning the aspect ratings from a collection of reviews of a given item. Most of the early works of opinion mining are frequency-based approaches [18, 30]. These approaches usually mine the aspects and sentiments by counting the frequencies of words and their co-occurrences with some pre-defined seed words. Recently, several methods based on the variants of topic models [6] have been proposed [56, 80, 20, 34, 35] to learn the aspects and sentiments automatically from the data. These work extends topic models by adding another kind of latent variables to model the latent sentiments of words, i.e., words in reviews are not only dependent on the topics they belong to, but are also related to the sentiments of the reviewers. The most related work is the Latent Aspect Rating Analysis Model (LARAM) [61, 62], which aims at inferring the latent aspect ratings of given reviews. LARAM assumes the overall rating of a review is generated by a weighted sum of the latent aspect ratings, and are generated from the words and the latent topic allocations of the words by a linear regression function. LARAM learns the latent aspect ratings for each review and aspect weights for each reviewer. It should be noted that the aspect weights in LARAM are different from the personalized aspect ratings in our problem. The weights in LARAM represent the importance of the aspects for a reviewer, but personalized tastes represent the ratings/sentiments of users on different aspects. Two reviewers may share similar aspect weights but have totally different ratings on a given aspect.

The main limitation of above aspect-based opinion mining methods is that they do not consider user preferences (across multiple reviews and items) in the learning procedures so that they are unable to predict users' opinions on other items which they have not written reviews on.

ETF [76] also considers aspect based opinion mining and collaborative filtering simultaneously. However, ETF employs the aspect-based opinion mining as a preprocessing step, while ours is a unified model with opinion mining as a part of the model. This enables our approach to be used to analyze the aspect distributions of the reviews and latent aspect ratings expressed in the reviews as in [61, 62]. Besides, ETF can not predict users' preferences on the aspects of items.

3.3 Problem Definition

We assume as input a collection of reviews of some products from a specific category (e.g. restaurant) by a group of reviewers, and each review comes with an overall rating (e.g. 1-5 stars) to express the overall satisfaction of the reviewer.

Review: A review is a piece of text describing opinions of a reviewer towards a specific item. Formally, we use $\mathscr{D} = \{d_1, d_2, ..., d_D\}$ to denote a set of review text documents. For each $d \in \mathscr{D}$, $u_d \in \mathscr{U}$ denotes the user who writes review d and $i_d \in \mathscr{I}$ denotes the reviewed item. We use $\mathscr{D}_u$ to denote the set of reviews that user u writes and use $\mathscr{D}_i$ to denote the set of reviews for item i.

Overall Rating: The overall rating r_d of a review document d is a numerical rating indicating the overall opinion of d, i.e., $r_d \in \mathscr{R}$, where $\mathscr{R} = \{1, 2, ..., R\}$.

Aspect: An aspect is an attribute of the item that has been commented on in a review, e.g., "food", "location" and "service" for a restaurant. In this chapter, we only consider the case that all the items are from a same category, i.e., they share the same set of aspects. We use a to denote an aspect, where $a \in \mathscr{A}$ and $\mathscr{A} = \{1, 2, ..., A\}$.

Aspect Rating: The aspect rating $r_{d,a}$ of a review document d is the reviewer u_d's rating towards to the aspect a of the item i_d.i It indicates the opinion of the reviewer regarding to the properties of the corresponding aspect of the item. Note that our method does not need aspect ratings as input, but instead it infers them from the data.

Personalized Latent Aspect Rating Analysis: Given a collection of reviews of a set of items by a set of users, the goal is to solve two tasks: *a*) learn the latent aspects, which represents each aspect as a distribution on a pre-defined vocabulary, and the latent aspect ratings for each review, which indicate the opinions of the reviewer towards the aspects of the item; *b*) predict the latent aspect ratings for user u on new item i that he has not reviewed.

Some important notations used in this chapter are listed in Table 5.1. We use bold math symbols $\boldsymbol{x}_i$ to denote vectors, where the subscript i is used for indexing different vectors. The j-th element of the vector $\boldsymbol{x}_i$ is denoted by $\boldsymbol{x}_i[j]$.

3.4 Proposed Methodology

To address the problem of Personalized Latent Aspect Rating Analysis, we propose a unified probabilistic model called Factorized Latent Aspect ModEl (FLAME), which combines the advantages of both collaborative filtering and aspect-based opinion mining so that the two methods can mutually enhance each other. The general idea of FLAME is that we can learn users' preferences based on their past reviews, so that we can collaboratively predict a user's preference of an aspect of an item from the opinions of other users with similar tastes. FLAME improves existing aspect-based opinion mining methods by being able to infer aspect ratings of users on new items[5], and enhances collaborative filtering methods by leveraging reviews to analyze users' preferences on different as-

[5]Note that *new items* for a user are the items that he has not rated yet.

Table 3.1: Mathematical Notations

Symbol	Size	Description
$\mathscr{U}$	U	Users $\mathscr{U} = \{u \mid u = 1, ..., U\}$
$\mathscr{I}$	I	Items $\mathscr{I} = \{i \mid i = 1, ..., I\}$
$\mathscr{D}$	D	Documents $\mathscr{D} = \{d \mid d = 1, ..., D\}$
$\mathscr{A}$	A	Aspects $\mathscr{A} = \{a \mid a = 1, ..., A\}$
$\mathscr{R}$	R	Numerical ratings $\mathscr{R} = \{r \mid r = 1, ..., R\}$
$\boldsymbol{\phi}_u$	$\mathbb{R}^K$	latent vector of user u
$\boldsymbol{\phi}_i$	$\mathbb{R}^K$	latent vector of item i
$\boldsymbol{\phi}_{i,a}$	$\mathbb{R}^K$	latent vector of aspect a of i
$\boldsymbol{\eta}$	$\mathbb{R}^A$	background aspect distribution
$\boldsymbol{\eta}_u$	$\mathbb{R}^A$	aspect distribution of user u
$\boldsymbol{\eta}_i$	$\mathbb{R}^A$	aspect distribution of item i
$\boldsymbol{\beta}_a$	$\mathbb{R}^V$	word distribution of aspect a
$\boldsymbol{\gamma}_{a,r}$	$\mathbb{R}^V$	word distribution of aspect a and rating r
$\boldsymbol{\theta}_d$	$\mathbb{R}^A$	aspect distribution of document d
$\boldsymbol{\varphi}_{d,a}$	$\mathbb{R}^R$	rating distribution of aspect a of document d
a_t	$\mathbb{R}^1$	aspect of sentence t
s_t	$\mathbb{R}^1$	aspect rating of sentence t

pects of items. In the following, we describe the details of FLAME in the view of the generative process when users writing reviews.

When writing the review, the reviewer first selects a subset of aspects he wants to comment on. We assume that each review document d is associated with an aspect distribution $\boldsymbol{\theta}_d \in \mathbb{R}^A$, which represents the importances of the aspects in the review. The aspect distribution $\boldsymbol{\theta}_d$ depends on three factors: the global aspect distribution $\boldsymbol{\eta}_0$, the aspect distribution of the reviewer $\boldsymbol{\eta}_u$ and the aspect distribution of the item $\boldsymbol{\eta}_i$. $\boldsymbol{\eta}_0$ represents how much each aspect is likely to be mentioned among all the reviews. $\boldsymbol{\eta}_u$ represents reviewer u's preferences on the aspects to comment, e.g., if a user cares more on the *value* of a hotel, he prefers to mention this aspect in his reviews. $\boldsymbol{\eta}_i$ indicates which aspects of item i are more likely to be mentioned. Some aspects are more likely to be mentioned in the reviews of an item. For example, if the food of a restaurant is great, it will receive a lot of praises of the *food* in the reviews. On the other hand, if some aspects of an item are terrible, reviewers would like to criticize on these aspects in their reviews.

Based on this assumption, we define $\boldsymbol{\theta}_d$ using a additive generative methods as follows:

$$\boldsymbol{\theta}_d[a] = \frac{\exp\left(\boldsymbol{\eta}_0[a] + \boldsymbol{\eta}_u[a] + \boldsymbol{\eta}_i[a]\right)}{\sum_{a'=1}^{A} \exp\left(\boldsymbol{\eta}_0[a'] + \boldsymbol{\eta}_u[a'] + \boldsymbol{\eta}_i[a']\right)} \tag{3.1}$$

where $\{\boldsymbol{\eta}\} = \{\boldsymbol{\eta}_0, \boldsymbol{\eta}_u, \boldsymbol{\eta}_i | u \in \mathscr{U}, i \in \mathscr{I}\}$ are A-dimensional vectors generated from zero-mean Gaussian distributions.

$$\begin{aligned} \boldsymbol{\eta}_0 &\sim \mathscr{N}(\mathbf{0}, \sigma_\eta \boldsymbol{I}) \\ \boldsymbol{\eta}_u &\sim \mathscr{N}(\mathbf{0}, \sigma_\eta \boldsymbol{I}) \\ \boldsymbol{\eta}_i &\sim \mathscr{N}(\mathbf{0}, \sigma_\eta \boldsymbol{I}) \end{aligned} \tag{3.2}$$

For each aspect, the reviewer has a latent sentiment polarity expressing his opinion on that aspect of the item. We extend Probabilistic Matrix Factorization (PMF) [48] to model the user-specific aspect ratings. PMF assumes that the user u has a vector of latent factors $\boldsymbol{\phi}_u \in \mathbb{R}^K$, which represents his personalized preferences that influence his opinions. Analogously, each item has a latent vector $\boldsymbol{\phi}_i \in \mathbb{R}^K$. The overall rating of u for i is generated by the dot product of the user latent factor and the item latent factor. In our model, to predict user u's opinion on a specific aspect a of item i, we assume there is a latent factor $\boldsymbol{\phi}_{i,a} \in \mathbb{R}^K$ for each aspect a of an item i, and the aspect rating $r_{d,a}$ of review document d is generated from the dot product of the user latent vector $\boldsymbol{\phi}_u$ and the item aspect latent vector $\boldsymbol{\phi}_{i,a}$[6]. The item aspect latent vector $\boldsymbol{\phi}_{i,a}$ describes the latent properties of the corresponding aspect of the item.

$$r_{d,a} \sim \mathscr{N}(\boldsymbol{\phi}_u^\top \boldsymbol{\phi}_{i,a}, \sigma_a^2) \tag{3.3}$$

To control the model complexity, zero-mean Gaussian priors are placed on the latent factors:

$$\begin{aligned} \boldsymbol{\phi}_u &\sim \mathscr{N}(\mathbf{0}, \sigma_u^2 \boldsymbol{I}) \\ \boldsymbol{\phi}_{i,a} &\sim \mathscr{N}(\mathbf{0}, \sigma_{i,a}^2 \boldsymbol{I}) \end{aligned} \tag{3.4}$$

We can not directly use the continuous value $r_{d,a}$ to model the word generative process since we need discrete ratings to define the aspect-sentiment vocabulary (see Equation 3.11). We introduce another latent variable $\boldsymbol{\varphi}_{d,a} \in \mathbb{R}^R$ to represent document d's rating distribution on aspect a, where $\boldsymbol{\varphi}_{d,a}[r]$ is the probability of $p(r_{d,a} = r)$ and $\sum_{r=1}^R \boldsymbol{\varphi}_{d,a}[r] = 1$. We define $\boldsymbol{\varphi}_{d,a}$ as follows:

$$\begin{aligned} \boldsymbol{\varphi}_{d,a}[r] &= \frac{\mathscr{N}(r | \boldsymbol{\phi}_u^\top \boldsymbol{\phi}_{i,a}, \sigma_{r,a}^2)}{\sum_{r'=1}^R \mathscr{N}(r' | \boldsymbol{\phi}_u^\top \boldsymbol{\phi}_{i,a}, \sigma_{r,a}^2)} \\ &= \frac{\exp\left(-\frac{\left(r - \boldsymbol{\phi}_u^\top \boldsymbol{\phi}_{i,a}\right)^2}{2\sigma_{r,a}^2}\right)}{\sum_{r'=1}^R \exp\left(-\frac{\left(r' - \boldsymbol{\phi}_u^\top \boldsymbol{\phi}_{i,a}\right)^2}{2\sigma_{r,a}^2}\right)} \end{aligned} \tag{3.5}$$

[6] Note that for simplicity we always assume that there is an extra constant column in user/item latent factors to model the bias effect.

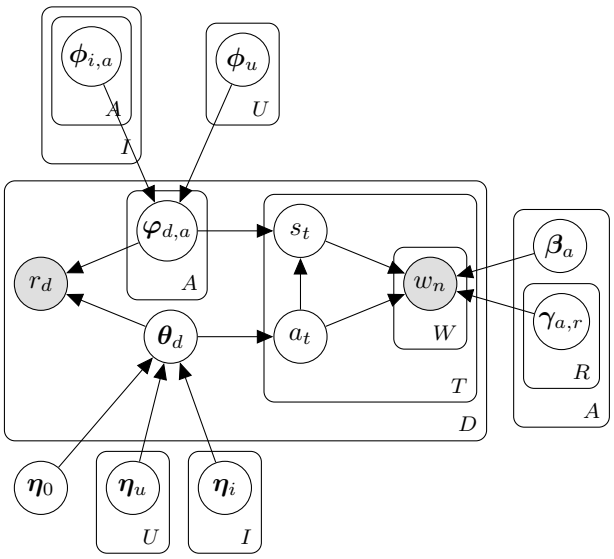


Figure 3.2: FLAME in graphical model notation.

We assume the overall rating of document d is generated from the weighted sum of the aspect ratings, where the aspect weights consist to the aspect distribution of the document.

$$r_d \sim \mathcal{N}(\sum_a \boldsymbol{\theta}_d[a]\mathbb{E}[r_{d,a}], \sigma_r^2) \tag{3.6}$$

where $\mathbb{E}[r_{d,a}] = \boldsymbol{\phi}_u^\top \boldsymbol{\phi}_{i,a}$.

For the process of generating words, we follow the assumption in [56, 33, 80] that the words in one sentence of a review refer to the same aspect. Topics learned under this assumption are *local* topics that preserve sentence-level word concurrences [56], while models like LDA [6] produce *global* topics that preserve document-level word concurrences. *Global* topic models are not suitable for aspect identification. For example, because the words *room* and *location* appear together in most reviews, *global* topic models are mostly likely to cluster them in the same topic, but *local* topic models assume they refer to different topics.

For each sentence t in the review d, we draw an aspect a_t from the aspect distribution of the review:

$$a_t \sim Multi(\boldsymbol{\theta}_d) \tag{3.7}$$

and a sentiment rating $s_t \in \{1,2,...,R\}$ on the aspect a_t from the aspect rating distribution:

$$s_t \sim Multi(\boldsymbol{\varphi}_{d,a_t}) \tag{3.8}$$

Then, in each sentence t, the reviewer selects a set of words $w_n \in t$ to express his opinions on the aspect a_t. We define an aspect-sentiment multinomial word distribution $\boldsymbol{\alpha}_{a,s}$ on the vocabulary, where $\boldsymbol{\alpha}_{a,s}[j]$ represents the probability of generating the j-th word from the vocabulary for aspect

Algorithm 1 Generative process of FLAME.

print *Draw* $\boldsymbol{\eta} \sim \mathcal{N}(\mathbf{0}, \sigma_\eta^2 \boldsymbol{I})$
for all $u \in \mathscr{U}$ **do**
  print *Draw* $\boldsymbol{\phi}_u \sim \mathcal{N}(\mathbf{0}, \sigma_u^2 \boldsymbol{I})$
  print *Draw* $\boldsymbol{\eta}_u \sim \mathcal{N}(\mathbf{0}, \sigma_\eta^2 \boldsymbol{I})$
end for
for all $i \in \mathscr{I}$ **do**
  print *Draw* $\boldsymbol{\eta}_i \sim \mathcal{N}(\mathbf{0}, \sigma_\eta^2 \boldsymbol{I})$
  for all $a \in \mathscr{A}$ **do**
    print *Draw* $\boldsymbol{\phi}_{i,a} \sim \mathcal{N}(\mathbf{0}, \sigma_{i,a}^2 \boldsymbol{I})$
  end for
end for
for all $a \in \mathscr{A}$ **do**
  print *Draw* $\boldsymbol{\beta}_a \sim \mathcal{N}(\mathbf{0}, \sigma_\beta^2 \boldsymbol{I})$
  for all $r \in \mathscr{R}$ **do**
    print *Draw* $\boldsymbol{\gamma}_{a,r} \sim \mathcal{N}(\mathbf{0}, \sigma_\gamma^2 \boldsymbol{I})$
    print *Set* $\boldsymbol{\alpha}_{a,r}$ using Equation (3.9)
  end for
end for
for all $d \in \mathscr{D}$ **do**
  print *Set* $\boldsymbol{\theta}_d$ using Equation (3.1)
  print *Set* $\boldsymbol{\varphi}_d$ using Equation (3.5)
  print *Draw* r_d using Equation (3.6)
  // Generate each sentence t in document d
  for all $t \in d$ **do**
    print *Draw* $a_t \sim \text{Multi}(\boldsymbol{\theta}_d)$
    print *Draw* $s_t \sim \text{Multi}(\boldsymbol{\varphi}_d, \boldsymbol{a}_t)$
    // Generate each word n in sentence t
    for all $n \in t$ **do**
      print *Draw* $w_n \sim \text{Multi}(\boldsymbol{\alpha}_{a_t, s_t})$
    end for
  end for
end for

a and aspect rating s. w_n can be an aspect word, e.g., *battery*, or a sentiment word, e.g., *good*. So we assume that $\boldsymbol{\alpha}_{a,s}$ depends on two factors: $\boldsymbol{\beta}_a$ and $\boldsymbol{\gamma}_{a,s}$, where $\boldsymbol{\beta}_a$ represents the correlation between the words and the aspect a, and $\boldsymbol{\gamma}_{a,s}$ represents the correlation between the words and the pair of aspect a and aspect rating s.

$$\boldsymbol{\alpha}_{a,s}[j] = \frac{\exp(\boldsymbol{\beta}_a[j] + \boldsymbol{\gamma}_{a,s}[j])}{\sum_{l=1}^{V} \exp(\boldsymbol{\beta}_a[l] + \boldsymbol{\gamma}_{a,s}[l])} \tag{3.9}$$

where $\boldsymbol{\beta}_a$ and $\boldsymbol{\gamma}_{a,r}$ are V-dimensional vectors generated from zero-mean Gaussian distributions.

$$\begin{aligned}\boldsymbol{\beta}_a &\sim \mathcal{N}(\mathbf{0}, \sigma_\beta \boldsymbol{I}) \\ \boldsymbol{\gamma}_{a,r} &\sim \mathcal{N}(\mathbf{0}, \sigma_\gamma \boldsymbol{I})\end{aligned} \tag{3.10}$$

The word w_n is generated as follows:

$$p(w_n|a_t, s_t, \boldsymbol{\alpha}) \sim Multi(\boldsymbol{\alpha}_{a_t,s_t}) \tag{3.11}$$

Figure 3.2 shows the graphical representation of FLAME, omitting the priors $\{\sigma\}$. We summarize the generative process in Algorithm 1.

3.4.1 Parameter Estimation

In general, we use an EM-style method to learn the parameters in our model. We adopt a mixture of maximum a posteriori (MAP) point estimates and Bayesian inference as in [12]. To be specific, we use a combination of MAP estimation over $\boldsymbol{\Xi} = \{\{\boldsymbol{\eta}\}, \{\boldsymbol{\phi}\}, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$ and Bayesian variational inference over the other latent variables $\boldsymbol{\Delta} = \{\boldsymbol{a}, \boldsymbol{s}\}$ to derive a lower bound of the log likelihood of the data, and maximize the bound with respect to $\boldsymbol{\Xi}$ and variational parameters. It should be noted that $\boldsymbol{\theta}$ and $\boldsymbol{\varphi}$ are not latent variables in our model. They are fixed given $\boldsymbol{\eta}$ and $\boldsymbol{\phi}$, as shown in Equation (3.1) and (3.5).

The variational distributions of the latent variables in $\boldsymbol{\Delta}$ are defined as follows:

$$q(\boldsymbol{a}, \boldsymbol{s}|\boldsymbol{\pi}, \boldsymbol{\lambda}) = \prod_d \prod_{t \in d} q(a_t|\boldsymbol{\pi}_t) q(s_t|\boldsymbol{\lambda}_t) \tag{3.12}$$

where $\boldsymbol{\pi}_t \in \mathbb{R}^A$ and $\boldsymbol{\lambda}_t \in \mathbb{R}^R$ are free multinomial parameters.

We get the lower bound of the log likelihood of the data as follows:

$$\begin{aligned}\mathscr{L} = &\sum_d \Bigg(\langle \log p(r_d|\boldsymbol{\phi}_u, \boldsymbol{\phi}_{i,a}, \boldsymbol{\theta}_d) \rangle + \sum_{t \in d} \Big(\langle \log p(a_t|\boldsymbol{\theta}_d) \rangle \\ &+ \langle \log p(s_t|\boldsymbol{\varphi}_d, a_t) \rangle + \sum_{n \in t} \langle \log p(w_n|a_t, s_t, \boldsymbol{\beta}, \boldsymbol{\gamma}) \rangle \Big) \Bigg) \\ &+ \sum_u \Big(\langle \log p(\boldsymbol{\phi}_u|\sigma_u) \rangle + \langle \log p(\boldsymbol{\eta}_u|\sigma_\eta) \rangle \Big) \\ &+ \sum_i \Big(\langle \log p(\boldsymbol{\phi}_i|\sigma_i) \rangle + \sum_a \langle \log p(\boldsymbol{\phi}_{i,a}|\sigma_{i,a}) \rangle \\ &+ \langle \log p(\boldsymbol{\eta}_i|\sigma_\eta) \rangle \Big) + \langle \log p(\boldsymbol{\eta}|\sigma_\eta) \rangle \\ &+ \sum_a \langle \log p(\boldsymbol{\beta}_a|\sigma_\beta) \rangle + \sum_a \sum_r \langle \log p(\boldsymbol{\gamma}_{a,r}|\sigma_\gamma) \rangle \\ &- \sum_d \sum_{t \in d} \Big(\langle \log q(\boldsymbol{a}_t|\boldsymbol{\pi}_t) \rangle + \langle \log q(\boldsymbol{s}_t|\boldsymbol{\lambda}_t) \rangle \Big)\end{aligned} \tag{3.13}$$

where $\langle p(\mathbf{\Omega})\rangle$ denotes the expectation of the probability of p given the distribution $q(\mathbf{\Omega})$.

3.4.2 Learning the Parameters

In general, the learning procedure can be viewed as coordinate ascent in $\mathscr{L}$, i.e., alternatively optimizing one set of parameters while fixing the others.

Updating $\boldsymbol{\pi}$: We get the solution of $\boldsymbol{\pi}_t$ by setting $\frac{\partial \mathscr{L}_{[\boldsymbol{\pi}_t]}}{\partial \boldsymbol{\pi}_t} = 0$ with the constraint $\sum_a \boldsymbol{\pi}_t[a] = 1$.

$$\boldsymbol{\pi}_t[a] \propto \boldsymbol{\theta}_d[a] \prod_r \left(\boldsymbol{\varphi}_{d,a}[r]^{\boldsymbol{\lambda}_t[r]} \prod_j \boldsymbol{\alpha}_{a,r}[j]^{c_{t,j}\boldsymbol{\lambda}_t[r]} \right) \tag{3.14}$$

where $c_{t,j}$ is the frequency of the j-th word in sentence t. Since $c_{t,j}$ is sparse, the complexity of updating $\boldsymbol{\pi}_t$ is $O(c_t \cdot R \cdot A)$, where c_t is the number of words in sentence t. The total complexity for updating $\boldsymbol{\pi}$ in one EM iteration is $O(c \cdot R \cdot A)$, where c is the number of words of all the documents.

Updating $\boldsymbol{\lambda}$: The update procedure of $\boldsymbol{\lambda}$ is similar to that for $\boldsymbol{\pi}$.

$$\boldsymbol{\lambda}_t[r] \propto \prod_a \boldsymbol{\varphi}_{d,a}[r]^{\boldsymbol{\pi}_t[a]} \prod_j \boldsymbol{\alpha}_{a,r}[j]^{c_{t,j}\boldsymbol{\pi}_t[a]} \tag{3.15}$$

The complexity of updating $\boldsymbol{\lambda}$ in one EM iteration is also $O(c \cdot A \cdot R)$.

Updating $\boldsymbol{\phi}_u$: We can get $\mathscr{L}_{[\boldsymbol{\phi}_u]}$ by only retaining those terms in $\mathscr{L}$ that are a function of $\boldsymbol{\phi}_u$:

$$\begin{aligned} \mathscr{L}_{[\boldsymbol{\phi}_u]} = \sum_{d \in \mathscr{D}_u} \Bigg(& -\frac{(r_d - \sum_a \boldsymbol{\theta}_d[a] \boldsymbol{\phi}_u^\top \boldsymbol{\phi}_{i,a})^2}{2\sigma_r^2} \\ & + \sum_t \sum_a \sum_r \boldsymbol{\pi}_t[a] \boldsymbol{\lambda}_t[r] \log \boldsymbol{\varphi}_{d,a}[r] \Bigg) - \frac{\boldsymbol{\phi}_u^\top \boldsymbol{\phi}_u}{2\sigma_u^2} \end{aligned} \tag{3.16}$$

The derivative of $\mathscr{L}_{[\boldsymbol{\phi}_u]}$ with respect to $\boldsymbol{\phi}_u$ depends on $\boldsymbol{\phi}_u$, so we have to use a gradient ascent based method to update $\boldsymbol{\phi}_u$.

Updating $\boldsymbol{\phi}_{i,a}$:

$$\begin{aligned} \mathscr{L}_{[\boldsymbol{\phi}_{i,a}]} = \sum_{d \in \mathscr{D}_i} \Bigg(& -\frac{(r_d - \sum_a \boldsymbol{\theta}_d[a] \boldsymbol{\phi}_u^\top \boldsymbol{\phi}_{i,a})^2}{2\sigma_r^2} \\ & + \sum_{t \in d} \sum_a \sum_r \boldsymbol{\pi}_t[a] \boldsymbol{\lambda}_t[r] \log \boldsymbol{\varphi}_{d,a}[r] \Bigg) \\ & - \frac{\boldsymbol{\phi}_{i,a}^\top \boldsymbol{\phi}_{i,a}}{2\sigma_{i,a}^2} \end{aligned} \tag{3.17}$$

We also use a gradient ascent based method to update $\boldsymbol{\phi}_{i,a}$.

Updating $\boldsymbol{\eta}$:

$$\begin{aligned}\mathscr{L}_{[\boldsymbol{\eta}_0]} =& \sum_{d\in D}\Big(-\frac{(r_d-\sum_a \boldsymbol{\theta}_d[a]\mathbb{E}[r_{d,a}])^2}{2\sigma_r^2} \\ &+\sum_{t\in d}\sum_a \boldsymbol{\pi}_t[a]\log\boldsymbol{\theta}_d[a]\Big)-\frac{\boldsymbol{\eta}_0^\top\boldsymbol{\eta}_0}{2\sigma_\eta^2} \\ =&\boldsymbol{\pi}_D^\top\boldsymbol{\eta}_0-\frac{\boldsymbol{\eta}_0^\top\boldsymbol{\eta}_0}{2\sigma_\eta^2}-\sum_{d\in D}\frac{(r_d-\sum_a \boldsymbol{\theta}_d[a]\mathbb{E}[r_{d,a}])^2}{2\sigma_r^2} \\ &-\sum_{d\in D}N_d\log\left(\sum_{a'}\exp\left(\boldsymbol{\eta}_0[a']+\boldsymbol{\eta}_u[a']+\boldsymbol{\eta}_i[a']\right)\right)\end{aligned} \tag{3.18}$$

where $\boldsymbol{\pi}_D=\sum_{d\in D}\sum_{t\in d}\boldsymbol{\pi}_t$ and $N_d=\sum_{t\in d}\sum_a\boldsymbol{\pi}_t[a]=\sum_{t\in d}1$ is the number of sentences in d.

We apply gradient ascent method to optimize $\boldsymbol{\eta}_0$. The derivative with respect to $\boldsymbol{\eta}_0[a]$ is :

$$\begin{aligned}g(\boldsymbol{\eta}_0[a]) =&\boldsymbol{\pi}_D[a]-\sum_{d\in D}N_d\boldsymbol{\theta}_d[a]-\frac{\boldsymbol{\eta}_0[a]}{\sigma_\eta^2} \\ &+\sum_{d\in D}\frac{(r_d-\sum_a \boldsymbol{\theta}_d[a]\mathbb{E}[r_{d,a}])(\boldsymbol{\theta}_d[a])(1-\boldsymbol{\theta}_d[a])}{\sigma_r^2}\end{aligned} \tag{3.19}$$

The update formula of $\boldsymbol{\eta}_u$ and $\boldsymbol{\eta}_i$ is similar. The only difference is to replace $\mathscr{D}$ in Equation (3.19) with $\mathscr{D}_u$ and $\mathscr{D}_i$ respectively, where $\mathscr{D}_u$ is the set of reviews of user u and $\mathscr{D}_i$ is the set of reviews of item i.

Updating $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$:

$$\begin{aligned}\mathscr{L}_{[\boldsymbol{\beta}_a]} =& \boldsymbol{c}_a^\top\boldsymbol{\beta}_a-\frac{\boldsymbol{\beta}_a^\top\boldsymbol{\beta}_a}{2\sigma_\beta^2} \\ &-\sum_d\sum_{t\in d}\boldsymbol{\pi}_t[a]c_t\sum_r\boldsymbol{\lambda}_t[r]\log\left(\sum_l\exp\left(\boldsymbol{\beta}_a[l]+\boldsymbol{\gamma}_{a,r}[l]\right)\right)\end{aligned} \tag{3.20}$$

where $\boldsymbol{c}_a[j]=\sum_d\sum_{t\in d}\boldsymbol{\pi}_t[a]c_{t,j}$, and $c_t=\sum_j c_{t,j}$ denotes the number of words in sentence t.

We use Newton's method to optimize $\boldsymbol{\beta}_a$. The derivative with respect to $\boldsymbol{\beta}_a$ is :

$$g(\boldsymbol{\beta}_a)=\boldsymbol{c}_a-\sum_r C_{a,r}\boldsymbol{\alpha}_{a,r}-\frac{\boldsymbol{\beta}_a}{\sigma_\beta^2} \tag{3.21}$$

where $C_{a,r}=\sum_d\sum_{t\in d}\boldsymbol{\pi}_t[a]c_t\boldsymbol{\lambda}_t[r]$ represents the expected word counts for each (a,r) combination. The Hessian matrix is:

$$H(\boldsymbol{\beta}_a)=\sum_r C_{a,r}\boldsymbol{\alpha}_{a,r}\boldsymbol{\alpha}_{a,r}^\top-diag(\sum_r C_{a,r}\boldsymbol{\alpha}_{a,r}+\frac{1}{\sigma_\beta}\mathbf{1}) \tag{3.22}$$

The update formula for $\boldsymbol{\beta}_a$ is:

$$\boldsymbol{\beta}_a^{(t+1)}=\boldsymbol{\beta}_a^{(t)}-\boldsymbol{H}^{-1}(\boldsymbol{\beta}_a^{(t)})g(\boldsymbol{\beta}_a^{(t)}) \tag{3.23}$$

We use a linear algorithm for the Hessian matrices with special structure [47, 6, 12], which lets the complexity of computing $\boldsymbol{H}^{-1}(\boldsymbol{\beta}_a)g(\boldsymbol{\beta}_a)$ be $O(V)$ instead of $O(V^3)$.

We can also get the derivative and Hessian of $\boldsymbol{\gamma}_{a,r}$ as follows:

$$g(\boldsymbol{\gamma}_{a,r}) = \boldsymbol{c}_{a,r} - C_{a,r}\boldsymbol{\alpha}_{a,r} - \frac{\boldsymbol{\gamma}_{a,r}}{\sigma_{\gamma}^2} \tag{3.24}$$

where $\boldsymbol{c}_{a,r}[j] = \sum_d \sum_{t \in d} \boldsymbol{\pi}_t[a]\boldsymbol{\lambda}_t[r]c_{t,j}$.

$$H(\boldsymbol{\gamma}_{a,r}) = C_{a,r}\boldsymbol{\alpha}_{a,r}\boldsymbol{\alpha}_{a,r}^{\top} - diag(C_{a,r}\boldsymbol{\alpha}_{a,r} + \frac{1}{\sigma_{\gamma}}\mathbf{1}) \tag{3.25}$$

The complexity of updating $\boldsymbol{\gamma}_{a,r}$ is also linear in the size of the vocabulary.

Computational Complexity: To conclude, the complexity of one update iteration is $O(c \cdot A \cdot R + T \cdot A \cdot K + D \cdot K + (I+U) \cdot A + A \cdot R \cdot V)$, where c is the total number of words in the corpus, T is the number of sentences in the corpus, and D is the number of documents in the corpus. Usually K, A and R are small constants, so the complexity is linear to the size of the review dataset.

Implementation Notes: An important issue is how to initialize the model. We use the following initialization steps. Taking the TripAdvisor data set as an example, we initialize $\boldsymbol{\beta}_a$ using the names of the aspect, i.e., we set $\boldsymbol{\beta}_{room,room} = 1$ for the aspect *room*, and then learn the aspect distribution of each sentence only based on the initialized $\boldsymbol{\beta}$. Similar techniques are also used in [33]. The aspect ratings of each sentence are initialized using the overall rating of the review. The parameters $\{\sigma\}$ can also be learned using the coordinate ascent-like procedure. We set them manually in our implementation, e.g., we set $\sigma_r^2 = 1$, $\sigma_{r,a}^2 = 0.5$, $\sigma_{\eta} = 10$, etc. Some optimization techniques, e.g., L-BFGS [38] and backtracking line search [7], are applied to accelerate the gradient ascent updates.

3.5 Experimental Evaluation

In this section, we first describe the data sets we used in our experiments and then discuss the experimental results on different tasks.

3.5.1 Data Sets and Preprocessing

We use two review data sets for our experimental evaluation: the TripAdvisor *hotel* review data[7] and Yelp review data[8]. In the TripAdvisor data, besides the overall rating, users are also asked to provide the aspect ratings on 6 pre-defined aspects: *Location*, *Sleep Quality*, *Room*, *Service*, *Value* and *Cleanliness*, on a scale from 1 star to 5 stars. We use these ground-truth aspect ratings to evaluate our model on the task of aspect rating prediction. For Yelp data set, we extract a subset which only contains the reviews on *restaurants*.

[7] `http://www.cs.cmu.edu/~jiweil/html/hotel-review.html`

[8] `http://www.yelp.com/dataset_challenge`

Table 3.2: Dataset Statistics

	TripAdvisor	Yelp
# Users	9,419	6,944
# Items	1,904	3,315
# Reviews	66,637	115,290
Density	0.37%	0.50%
# Sentences Per Review	12.60 ± 8.64	11.67 ± 7.80
# Words Per Sentence	7.50 ± 3.76	6.47 ± 4.64

We use the following preprocessing procedure on both of the data sets. We first remove non-English reviews and reviews with less than 3 sentences or 20 words, and then iteratively remove users with less than 5 reviews and items with less than 5 reviews. For the text in reviews, we remove stop words and words that occur in less than 5 reviews, and stem the remaining words using the PorterStemmer[9]. After the preprocessing, we have a *hotel* review data set including 66,637 hotel reviews of 1,904 hotels and a *restaurant* review data set including 115,290 reviews of 3,315 restaurants. The detailed statistics are listed in Table 3.2.

We randomly split both of the data sets into training and test sets. Specifically, for each user, we randomly select 20% of his reviews as test examples (For users with less than 10 reviews, we randomly select 2 reviews as test examples) and put the rest reviews into the training sets. We train the models on the training data sets and test their performance on the test data sets. We use the model initialization method and parameters selection strategies as discussed in Section 4.

3.5.2 Quantitative Evaluation

Perplexity on Held-out Reviews

As in standard topic models, we use *perplexity* of the held-out test data sets to compare the generalization performance of FLAME with some other state-of-the-art models.

Evaluation Measure. *Perplexity* is a standard measure for topic models to measure how the model can generate future documents [6]. For review mining, a good aspect-based topic model should be able to predict what the reviewer will write in a new review, which leads to a lower perplexity. A strong correlation of the perplexity and accuracy of aspect-based topic models is shown in [35]. We use a lower bound on perplexity as in [16].

$$\text{Perplexity}(\mathscr{D}_{test}) = \exp\left(-\frac{\sum_d \langle \log p(\boldsymbol{w}_d|\boldsymbol{\Xi})\rangle - \langle p(\boldsymbol{\Delta}_d)\rangle}{\sum_d N_d}\right)$$

We compare FLAME with the basic LDA model [6] and the D-LDA model presented in [35]. D-LDA is a state-of-the-art aspect-based opinion mining model which can be seen as a generalization of several other models [56, 20]. For D-LDA, we also use the assumption that the words in one

[9] http://tartarus.org/martin/PorterStemmer/

Table 3.3: Perplexity on the held-out data sets

	TripAdvisor	Yelp
LDA-A	1012.80	767.24
LDA-AR	918.07	728.00
D-LDA	771.05	621.24
FLAME	**733.12**	**590.46**

sentence refer to the same aspect as in FLAME and other models [35, 56, 20]. In the aspect-based topic models, we actually use $A \times R$ latent topics, so we compare with LDA using both A topics (LDA-A) and $A \times R$ topics (LDA-AR).

For all the models, we use the same parameter settings and stopping criteria. We set $R = 5$ for all the aspect-based topic models. We train the models using the reviews in the training sets and evaluate the perplexity on the test sets. We test various numbers of latent aspects $A = \{6, 12, 24\}$. Since the relative results are similar, we choose $A = 6$ for discussion. Table 3.3 shows the perplexity on test data sets of FLAME and the comparison partners. We can see that D-LDA and FLAME, which are specifically designed for aspect-based opinion mining, significantly outperform basic LDA methods. FLAME achieves the best results among all the models on both of the data sets. We believe this is because FLAME can predict personalized aspect distribution as well as aspect rating distribution, which other models do not consider.

Aspect Rating Prediction on Held-out Reviews

Since we need the ground-truth aspect ratings to quantitatively compare FLAME with other methods, we evaluate the aspect rating prediction only on the TripAdvisor data set. In order to align the latent aspects to the pre-defined aspects in the TripAdvisor data set, we set A to be 6 and use the initialization techniques discussed in Section 4.

We use the evaluation measures in [61, 62] to evaluate different methods:

- Root Mean Square Error (RMSE) of the predicted aspect ratings compared with the ground-truth aspect ratings.
- Pearson Correlation inside reviews (ρ_A) to measure how well the predicted aspect ratings preserve the relative order of aspects within a review.

$$\rho_A = \frac{1}{D} \sum_{d=1}^{D} \rho(\boldsymbol{s}_d, \boldsymbol{s}_d^*)$$

  where $\boldsymbol{s}_d^*$ is the ground-truth aspect ratings for document d, and $\boldsymbol{s}_d$ is predicted aspect ratings.

- Pearson Correlation between personalized ranking of items ρ_I. For each user and each aspect, we rank the items by their predicted aspect ratings, and measure how the ranked lists preserve

Table 3.4: Aspect rating prediction on test set of TripAdvisor data

	PMF	LRR+PMF	FLAME
RMSE	**0.970**	1.000	0.980
ρ_A	N/A	0.110	**0.195**
ρ_I	0.304	0.177	**0.333**
$L_{0/1}$	0.210	0.238	**0.196**

the ground truth.

$$\rho_I = \frac{1}{U \cdot A} \sum_{u=1}^{U} \sum_{a=1}^{A} \rho(\boldsymbol{s}_{\mathscr{I}_u,a}, \boldsymbol{s}^*_{\mathscr{I}_u,a})$$

where $\mathscr{I}_u$ is the set of items in user u's test data, $\boldsymbol{s}_{\mathscr{I}_u,a}$ is the predicted aspect ratings on the set of items and $\boldsymbol{s}_{\mathscr{I}_u,a}$ is the ground-truth ratings.

- Zero-One Ranking loss ($L_{0/1}$) [25], which measures the percentage of mis-ordered pairs of items for each user. It is computed as follows:

$$\sum_u \sum_{i,j \in \mathscr{I}_u} \frac{1}{Z_u} \sum_{a=1}^{A} \mathbf{1}[(\boldsymbol{s}_{u,i,a} - \boldsymbol{s}_{u,j,a}) \cdot (\boldsymbol{s}^*_{u,i,a} - \boldsymbol{s}^*_{u,j,a}) < 0]$$

where Z_u is the number of pairs in user u's test set, $\boldsymbol{s}_{u,i,a}$ is the predicted rating of user u of item i on aspect a, and $\boldsymbol{s}^*_{u,i,a}$ is the ground-truth aspect rating. We do not choose nDCG since each user has few samples in the test data (2.2 test samples per user), the values of nDCG tend to be very close to 1 for all comparison partners.

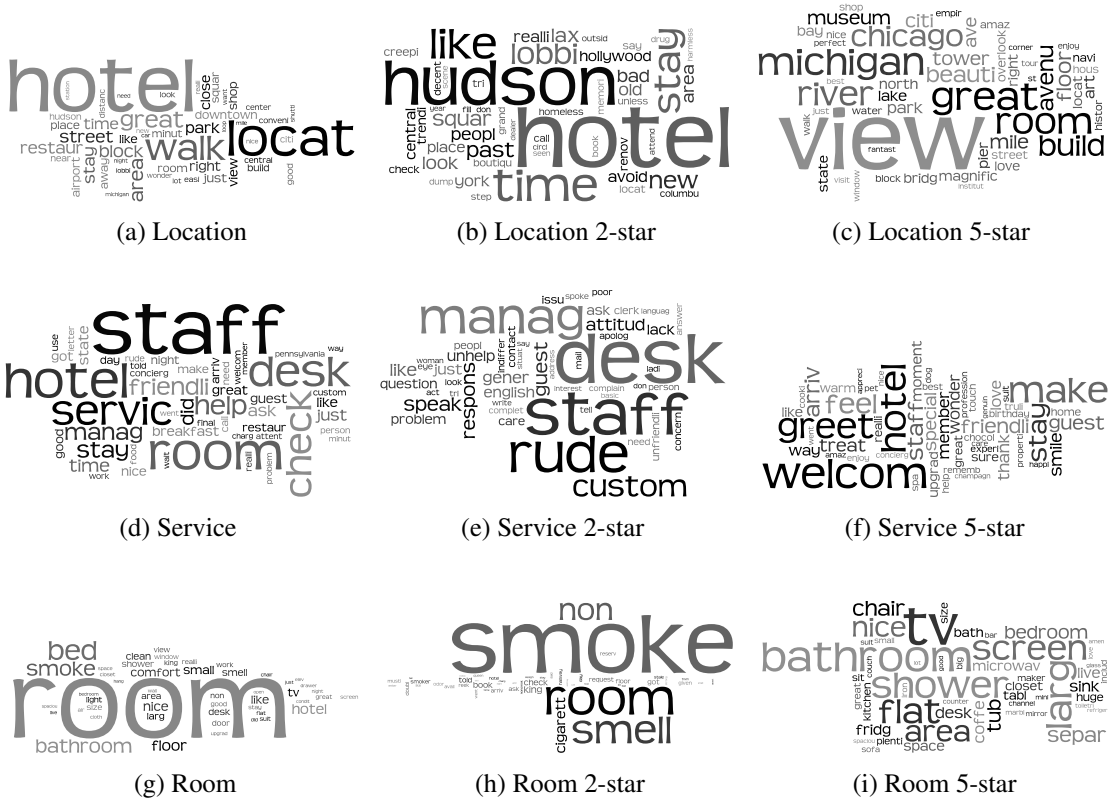
An intuitive solution of aspect rating prediction is just using the overall rating of the review as prediction. We use PMF [48] to predict the overall ratings of the reviews in the test set and use the predicted overall ratings as predictions for aspect ratings. To our best knowledge, [61, 62] are the only work that predict aspect ratings at review-level. However, they can only predict aspect ratings based on users' reviews. In order to predict the aspect ratings in test set, we first apply the LRR model [61] to extract aspect ratings for each review in the training set, and then use PMF [48] to train and to predict the aspect ratings in test set (we call it LRR+PMF). We use the code of LRR provided by the authors. Same training and testing strategies are applied to all the models for fair comparison. We also test the performance of with different values of the dimensions of latent factors. The relative results are similar, so we only choose $K = 10$ for discussion. Table 3.4 presents the results for aspect rating prediction on the test set. We also highlight the best performance in each measure in bold.

A general observation is that FLAME outperforms other baseline models on all the measures except RMSE. It has been discussed in [61] that RMSE is less important than other measures since it does not reflect how the relative order of the aspect ratings is preserved. ρ_A measures how a method preserves the relative order of aspect ratings within a review. PMF uses the same predicted ratings

for all aspects, so it is not applicable for ρ_A. ρ_I and $L_{0/1}$ are the most important measures for our task where we want to rank items based on the predicted aspect ratings. PMF outputs exactly the same item ranking lists for all aspects, thus it is not suitable for real-world applications. We can see that FLAME gets the best results on the two measures. The gain is especially significant compared to LRR+PMF, where there are about 90% improvement on ρ_I and 40% improvement on $L_{0/1}$.

Note that LRR+PMF does not achieve desirable performance. The reason is that it is a two-step approach that the errors induced in the first step have significant influence on the performance of the second step.

3.5.3 Qualitative Evaluation



(a) Location (b) Location 2-star (c) Location 5-star

(d) Service (e) Service 2-star (f) Service 5-star

(g) Room (h) Room 2-star (i) Room 5-star

Figure 3.3: Word-cloud visualization of top words with highest generating probability in $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$. Word size reflects the weight of each word.

In this subsection, we evaluate FLAME on the task of aspect identification. We perform qualitative analysis of the top words obtained by FLAME to see whether FLAME can produce meaningful aspects.

Figure 3.3 shows the word-cloud visualization of top words (after stemming) with the highest generating probability in the aspect-rating specific word distributions. The three rows are the top words in the topic distributions for the aspects *location*, *service* and *room*, respectively. The left column shows the top words in the aspect-word distributions $\boldsymbol{\beta}$ of the three aspects. The middle

and right columns show the top words in the aspect-rating-word distributions $\boldsymbol{\gamma}$. The middle column shows negative ones (2-star) and the right column shows positive ones (5-star). In general, the top words generated by FLAME represent meaningful and interpretable topics. We observe that the top words match our intuition, e.g., words like "location", "walk", "street" have higher weights in the word distribution of aspect *location*. The middle and right columns show the top words of the 2-star ($\boldsymbol{\gamma}_{a,2}$) and 5-star ($\boldsymbol{\gamma}_{a,5}$) word distributions of for the three aspects,. The aspect-rating specific word distribution can automatically learn the sentiment oriented words, e.g., words like "bad", "old", "creepy" and "homeless" have high weights in the 2-star word distribution of the aspect *location*, while the words like "view", "great", "perfect", "best" have high weights in the 5-star word distribution of *location*.

One contribution of FLAME is that the aspect-rating topics have sentiment polarities, i.e., 5-star topics are more positive than 4-star, and so on. This is different from previous work [56, 34, 36] where the latent ratings in these models are rating labels which do not correspond to sentiment polarities.

3.5.4 Further Applications

The detailed analysis on personalized latent aspect ratings enables a wide range of applications. Here we discuss three sample applications.

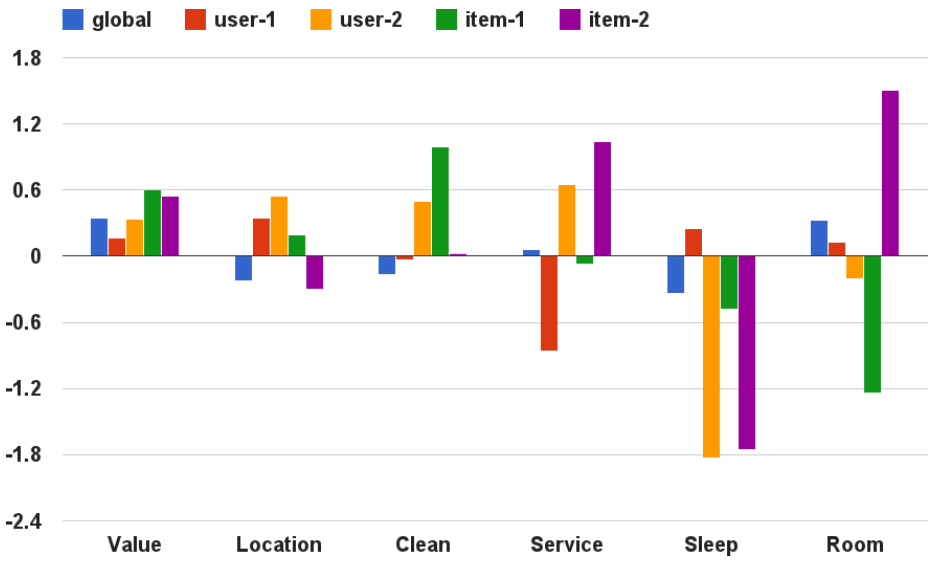


Figure 3.4: Aspect Weights. *Global* represents the values of $\boldsymbol{\eta}_0$. *user-1* and *user-2* are the aspect weights $\boldsymbol{\eta}_u$ of two randomly sampled users, and *item-1* and *item-2* are the values of $\boldsymbol{\eta}_i$ for two randomly sampled items.

Aspect Distribution: Since FLAME can infer the aspect weights for users and item, we can easily use the values of $\boldsymbol{\eta}_0$, $\boldsymbol{\eta}_u$ and $\boldsymbol{\eta}_i$ for the rating behavior analysis. Figure 3.4 shows some sample results on TripAdviosr data. From the histogram *Global* in the figure, we can see that *Value* and *Room* are the most discussed aspects, and most people rarely mention the aspect *Sleep*. Note

that the values of $\boldsymbol{\eta}_u$ indicates the biases of users deviating from the global aspect weights. We can see that *user-1* likes to comment on the *Location* and *Sleep*, while *user-2* cares more about the *Service*, *Clean* and *Location*. The two users have opposite weights for the aspects *Service* and *Sleep*. Thus, when they are searching for hotels, the aspects they care about are different. It indicates that letting users choose to rank items based on aspects which they cares about is very useful. The aspect weights of items can be used to help merchants to improve their services. If a specific aspect is discussed a lot and most of the reviews are negative, the merchant should think about how to improve this aspect.

Personalized Review Recommendation: As discussed in Section 1, facing a large number of reviews expressing different opinions, a user might have no idea of which reviews are reliable. FLAME can alleviate this problem by sorting the reviews by the similarities between reviewers with current user. A simple way of computing the similarities between users is to compute the distance between their latent factors. Since personalized review recommendation is hard to evaluate, we would like to leave it as a future work on some data sets with ground-truth of user feedback on reviews.

Recommendation Explanation: Traditional collaborative filtering methods only provide predicted scores for then items, but can not produce reasonable explanations with the recommendations. A recent work [76] has shown the possibility of using the aspect weights to generate some explanations. FLAME can produce more persuasive recommendation explanations by the predicted aspect ratings and some selected reviews written by similar users.

3.6 Conclusion

In this chapter, we introduced the problem of Personalized Latent Aspect Rating Analysis to model users' preferences on different aspects. We proposed a unified probabilistic model FLAME which combines aspect-based opinion mining and collaborative filtering. FLAME extracts aspect ratings from the review text, and predicts aspect ratings of an item that a user has not yet reviewed based on aspect ratings within other reviews of the item by other users with similar preferences. Our experimental evaluation on a hotel review data set and a restaurant review data set shows that FLAME can effectively solve the research problem. The qualitative evaluation shows that FLAME can automatically extract meaningful aspects and sentiment-oriented aspects. We also investigated the ability of FLAME on the task of generating future review text. Most importantly, our experiments on TripAdvisor data sets show that FLAME significantly outperforms state-of-the-art methods in terms of accuracy of aspect rating prediction.

Chapter 4

CDAE: On Better Top-N Recommendation

In this chapter, we study the second research question that we asked in Chapter 1: *If we look through a user's view (*i.e.*, a row in the matrix), what is personalized recommendation in principle*?

We present a new model-based collaborative filtering (CF) method for top-N recommendation called Collaborative Denoising Auto-Encoder (CDAE). CDAE assumes that whatever user-item interactions are observed are a *corrupted* version of the user's full preference set. The model learns latent representations of corrupted user-item preferences that can best reconstruct the full input. In other words, during training, we feed the model a subset of a user's item set and train the model to recover the whole item set; at prediction time, the model recommends new items to the user given the existing preference set as input. Training on corrupted data effectively recovers co-preference patterns. We show that this is an effective approach for collaborative filtering.

Learning from intentionally corrupted input has been widely studied. For instance, Denoising Auto-Encoders [57] train a one-hidden-layer neural network to reconstruct a data point from the latent representation of its partially corrupted version. However, to our best knowledge, no previous work has explored applying the idea to recommender systems.

CDAE generalizes several previously proposed, state-of-the-art collaborative filtering models (see Section 4.3.2). But its structure is much more flexible. For instance, it is easy to incorporate non-linearities into the model to achieves better top-N recommendation results. We investigate the effects of various choices for model components and compare their performance against prior approaches on three real world data sets. Experimental results show that CDAE consistently outperforms state-of-the-art top-N recommendation methods by a significant margin on a number of common evaluation metrics.

4.1 Background and Overview

4.1.1 Overview of Model-based Recommenders

Most machine learning models can be specified through two components: **model definition** and **objective function** during training. The model definition formulates the relationship between the input (*e.g.*, user ids, item ids, interactions, other features, etc.) and output (ratings or implicit feedback of items). The objective function is what the training process optimizes to find the best model parameters.

Recommender Models

In general, recommender models are defined as

$$\hat{y}_{ui} = \mathscr{F}_{\boldsymbol{\theta}}(u, i), \tag{4.1}$$

where $\hat{y}_{ui}$ is the predicted preference of user u on item i, and $\boldsymbol{\theta}$ denotes the model parameters we need to learn from training data.

Different choices of the function $\mathscr{F}_{\boldsymbol{\theta}}$ correspond to different assumptions about how the output depends on the input. Here we review 4 common recommender models.

Latent Factor Model (LFM). LFM models the preference $\hat{y}_{ui}$ as the dot product of latent factor vectors $\boldsymbol{v}_u$ and $\boldsymbol{v}_i$, representing the user and the item, respectively [23, 48]. [1]

$$\hat{y}_{ui} = \mathscr{F}^{LFM}_{\boldsymbol{v}}(u, i) = \boldsymbol{v}_u^\top \boldsymbol{v}_i \tag{4.2}$$

In addition, hierarchical latent factor models [1, 75] and the factorization machine [43] can model interactions between user or item side features.

Similarity Model (SM). The Similarity model [22] models the user's preference for item i as a weighted combination of the user's preference for item j and the item similarity between i and j. It is a natural extension of an item-based nearest neighbor model. The difference is that SM does not use predefined forms of item similarity (*e.g.*, *Jaccard*, *Cosine*). Instead, it learns a similarity matrix from data [37].

$$\hat{y}_{ui} = \mathscr{F}^{SM}_{\boldsymbol{S}}(u, i) = \sum_{j \in \mathscr{O}_u \setminus \{i\}} y_{uj} \cdot \boldsymbol{S}_{ji} \tag{4.3}$$

Factorized Similarity Model (FSM). The problem with the Similarity Model is that the number of parameters is quadratic in the number of items, which is usually impractical. A straightforward solution is to factorize the similarity matrix into two low rank matrices [22, 21].

$$\hat{y}_{ui} = \mathscr{F}^{FSM}_{\boldsymbol{p},\boldsymbol{q}}(u, i) = \left(\sum_{j \in \mathscr{O}_u \setminus \{i\}} y_{uj} \cdot \boldsymbol{p}_j \right)^\top \boldsymbol{q}_i \tag{4.4}$$

[1] To simplify notation, we assume that the latent factors are padded with a constant to model the bias.

LFSM (LFM+FSM). The above models can also be combined. For example, combining LFM and FSM results in the model SVD++ [22], which proved to be one of the best single models for the Netflix Prize.

$$\hat{y}_{ui} = \mathscr{F}_{\boldsymbol{p},\boldsymbol{q}}^{LFSM}(u,i) = \left(\sum_{j \in \mathscr{O}_u \setminus \{i\}} y_{uj} \cdot \boldsymbol{p}_j + \boldsymbol{p}_u \right)^\top \boldsymbol{q}_i \tag{4.5}$$

Objective Functions for Recommenders

Objective functions for training recommender models can be roughly grouped into two groups: point-wise and pair-wise[2]. Pair-wise objectives approximates ranking loss by considering the relative order of the predictions for pairs of items. Point-wise objectives, on the other hand, only depend on the accuracy of the prediction of individual preferences. Pair-wise functions are usually considered to be more suitable for optimizing top-N recommendation performance.

Regardless of the choice of a pair-wise or point-wise objective function, if one aims at making satisfactory top-N recommendations, it is critical to properly take unobserved feedback into account within the model. Models that only consider the observed feedback fail to account for the fact that ratings are *not* missing at random. These models are not suitable for top-N recommendation [40, 55].

Let $\ell(\cdot)$ denote a loss function and $\Omega(\boldsymbol{\theta})$ a regularization term that controls model complexity and encodes any prior information such as *sparsity*, *non-negativity*, or *graph regularization*. We can write the general forms of objective functions for recommender training as follows.

Point-wise objective function.

$$\sum_{(u,i) \in \mathscr{O}'} \ell_{point}(y_{ui}, \hat{y}_{ui}) + \lambda \Omega(\boldsymbol{\theta}). \tag{4.6}$$

Here $\mathscr{O}'$ denotes an augmented dataset that includes both observed and unobserved user-item pairs. The problem with using only observed user-item pairs is that users prefer to rate/view/purchase items they like than those they dislike. In this case, directly optimizing the point-wise objective function over $\mathscr{O}$ leads to a biased model towards to the observed sets [40, 55]. A common solution is to augment $\mathscr{O}$ with a subset of *negative* user-item pairs [3] from the unobserved set, and train the model on the augmented set $\mathscr{O}'$ while re-sampling the subset at the beginning of each epoch of the training phase. Several strategies for sampling negative user-item pairs are discussed in [40]. $\mathscr{O}'$ can also include duplicate samples to simulate feedback with different confidence weights (*e.g.*, [19]).

Pair-wise objective function.

$$\sum_{(u,i,j) \in \mathscr{P}} \ell_{pair}(y_{uij}, \hat{y}_{uij}) + \lambda \Omega(\boldsymbol{\theta}), \tag{4.7}$$

[2] Some models use list-wise objective functions [65, 54], but they are not as widely adopted as point-wise and pair-wise objectives.

[3] Here we use the term *negative* to denote missing feedback.

Table 4.1: Overview of related model-based recommenders.

Name	Model	Objective Function
MF [23] / PMF [48]	LFM	ℓ^{SL}_{point}
SVD++ [22]	LFSM	ℓ^{SL}_{point}
MMMF [45]	LFM	ℓ^{HL}_{point}
WSABIE [66]	LFM	ℓ^{HL}_{pair}
BPR-MF [44]	LFM	ℓ^{LL}_{pair}
SLIM [37]	SM	ℓ^{SL}_{point}
FISM [21]	FSM	ℓ^{SL}_{point}, ℓ^{SL}_{pair}
WRMF [19]	LFM	*weighted* ℓ^{SL}_{point}
COFI [65]	LFM	NDCG loss
CLiMF [53]	LFM	MRR loss

where $y_{uij} = y_{ui} - y_{uj}$, $\hat{y}_{uij} = \hat{y}_{ui} - \hat{y}_{uj}$, and $\mathscr{P}$ is a set of triplets sampled from $\mathscr{O}'$, each of which includes a user u and a pair of items i and j, where usually i is a positive item and j is a negative item sampled from the missing set.

For both pair-wise and point-wise objective functions, the choice of the loss function $\ell(\cdot)$ is important. Here we list a few commonly used loss functions for both point-wise and pair-wise objectives for implicit feedback[4].

- SQUARE LOSS: $\ell^{SL}(y,\hat{y}) = \frac{1}{2}(y-\hat{y})^2$;
- LOG LOSS: $\ell^{LL}(y,\hat{y}) = \log(1+\exp(-y\cdot\hat{y}))$;
- HINGE LOSS: $\ell^{HL}(y,\hat{y}) = \max(0, 1-y\cdot\hat{y})$;
- CROSS ENTROPY LOSS: $\ell^{CE}(y,\hat{y}) = -y\log(p) - (1-y)\log(1-p)$, where $p = \sigma(\hat{y}) = 1/(1+\exp(-\hat{y}))$.

For explicit feedback, the loss functions are similar but sightly different (*e.g.*,[25]).

Table 4.1 summarizes recent models for top-n recommendation that fit this framework. Also, several recent papers study position-aware pair-wise loss functions (*e.g.*, WARP [66, 67], CLiMF [53]). Any objective function that fits the described framework can be used along with any model we described above.

To summarize, the two key components of designing model-based recommenders are: 1) a suitable way to represent the relations between inputs and outputs. 2) a proper objective function and a proper way to deal with the relationship between observed and unobserved feedback.

[4]Note that, for LOG and HINGE losses, the value y for the negative examples should be -1 instead of 0.

4.1.2 Denoising Auto-Encoders

A classical auto-encoder [3] is typically implemented as a one-hidden-layer neural network that takes a vector $\boldsymbol{x} \in \mathbb{R}^D$ as input and maps it to a hidden representation $\boldsymbol{z} \in \mathbb{R}^K$ through a mapping function

$$\boldsymbol{z} = h(\boldsymbol{x}) = \sigma\left(\boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{b}\right),$$

where $\boldsymbol{W}$ is a $D \times K$ weight matrix and $\boldsymbol{b} \in \mathbb{R}^K$ is an offset vector. The resulting latent representation is then mapped back to a reconstructed vector $\hat{\boldsymbol{x}} \in \mathbb{R}^D$ through

$$\hat{\boldsymbol{x}} = \sigma\left(\boldsymbol{W}'\boldsymbol{z} + \boldsymbol{b}'\right).$$

The reverse mapping may optionally be constrained by *tied weights*, where $\boldsymbol{W}' = \boldsymbol{W}$.

The parameters of this model are trained to minimize the average reconstruction error:

$$\arg \min_{\boldsymbol{W}, \boldsymbol{W}', \boldsymbol{b}, \boldsymbol{b}'} \frac{1}{n} \sum_{i=1}^{n} \ell\left(\boldsymbol{x}_i, \hat{\boldsymbol{x}}_i\right), \tag{4.8}$$

where ℓ is a loss function such as the *square loss* or the *cross entropy loss* mentioned in the previous subsection.

The Denoising Auto-encoder (DAE) [57] extends the classical auto-encoder by training to reconstruct each data point $\boldsymbol{x}$ from its (partially) corrupted version $\tilde{\boldsymbol{x}}$. The goal of DAE is to force the hidden layer to discover more robust features and to prevent it from simply learning the identity function [57]. The corrupted input $\tilde{x}$ is typically drawn from a conditional distribution $p(\tilde{\boldsymbol{x}}|\boldsymbol{x})$. Common corruption choices are the additive Gaussian noise and the multiplicative mask-out/drop-out noise. Under mask-out/drop-out corruption, one randomly overwrites each of the dimensions of $\boldsymbol{x}$ with 0 with a probability of q:

$$\begin{aligned} P(\tilde{\boldsymbol{x}}_d = \delta \boldsymbol{x}_d) &= 1 - q \\ P(\tilde{\boldsymbol{x}}_d = 0) &= q \end{aligned} \tag{4.9}$$

To make the corruption unbiased, one sets the uncorrupted values to $\delta = 1/(1-q)$ times their original value.

4.2 Related Work

An overview of the model-based collaborative filter methods has been discussed in 4.1.1. In this section, we discuss the few related works on neural networks for recommender systems.

Restricted Boltzmann Machines (RBM) [49] is the first work that applies neural network models to recommender systems. However, RBM targets rating prediction, not top-N recommendation, and its loss function considers only the observed ratings. How to incorporate negative sampling, which would be required for top-N recommendation, into the training of RBM is technically challenging.

We do not compare with RBM in our experiments, but test several other neural network baselines that work for top-N recommendation (see Section 4.4.4).

We also notice that there is a concurrent work AutoRec [51] using the Auto-Encoder for rating prediction. The main differences are as follows: 1) AutoRec only considers the observed ratings in the loss function, which does not guarantee the performance for top-N recommendation. 2) They use the vanilla Auto Encoder model as the structure, while we have proved that introducing user factors in the model can greatly improve the performance. 3) AutoRec does not employ the denoising technique, which is a major part of our work.

Another related work is [60], which also uses the Auto-Encoder for recommender systems. This work studies the particular problem of article recommendation, and improves the well-known model Collaborative Topic Regression [59] by replacing its Topic Model component by a Bayesian Auto-Encoder, which is used for learning the latent feature representations for the articles. Different from this model, our model is generic and addresses the general top-N recommendation problem, and the inputs are user behaviors instead of item/article features.

4.3 Proposed Methodology

In this section, we introduce a new model – Collaborative Denoising Auto-Encoder (CDAE). The model learns correlations between the user's item preference by training on a corrupted version of the known preference set. A preference set is binary, *i.e.*, containing only information about whether an item is preferred or not. Therefore, as we will see, CDAE is uniquely suitable for top-N preference recommendations.

4.3.1 Collaborative Denoising Auto-Encoder

Similar to the standard Denoising Auto-Encoder, CDAE is also represented as a one-hidden-layer neural network. The key difference is that the input also encodes a latent vector for the user, which allows CDAE to be a much better recommender model, as we see in section 5.5. Figure 4.1 shows a sample structure of CDAE. CDAE consists of 3 layers, including the input layer, the hidden layer and the output layer.

In the input layer, there are in total $I+1$ nodes, where each of the first I nodes corresponds to an item, and the last node is a user-specific node (the red node in the figure), which means the node and its associated weights are unique for each user $u \in \mathscr{U}$ in the data. We refer to the first I nodes as *item input nodes*, and the last node as *user input node*. Given the historical feedback $\mathscr{O}$ by users on the item set $\mathscr{I}$, we can transform $\mathscr{O}$ into the training set containing U instances $\{\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_U\}$, where $\boldsymbol{y}_u = \{y_{u1}, y_{u2}, ..., y_{uI}\}$ is the I-dimensional feedback vector of user u on all the item in $\mathscr{I}$. $\boldsymbol{y}_u$ is a sparse binary vector that only has $|\mathscr{O}_u|$ non-zero values: $y_{ui} = 1$ if i is in the set $\mathscr{O}_u$, otherwise, $y_{ui} = 0$.

There are K nodes in the hidden layer and these nodes are fully connected to the nodes of the input layer. Here K is a predefined constant which is usually much smaller than the size of the input

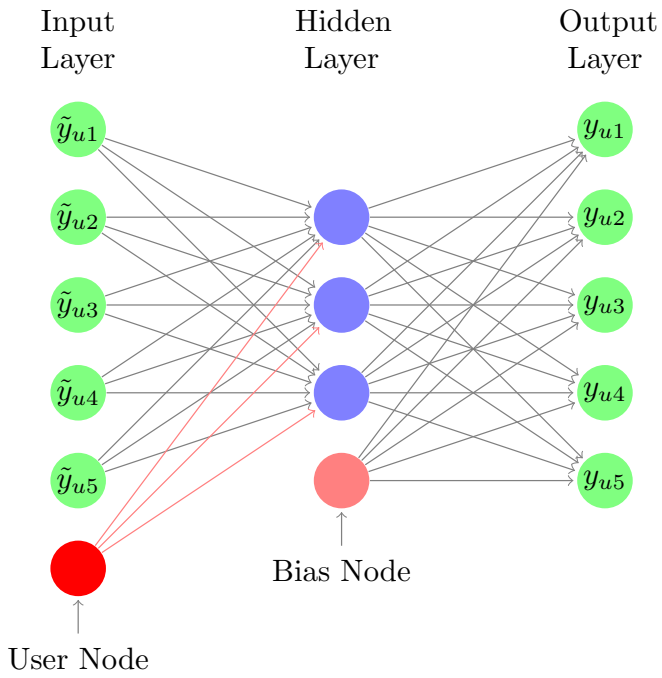


Figure 4.1: A sample CDAE illustration for a user u. The links between nodes are associated with different weights. The links with *red* color are user specific. Other weights are shared across all the users.

vectors. The hidden layer also has an additional node to model the bias effects (the pink node in the figure). We use $\boldsymbol{W} \in \mathbb{R}^{I \times K}$ to denote the weight matrix between the *item input nodes* and the nodes in the hidden layer, and $\boldsymbol{V}_u \in \mathbb{R}^K$ to denote the weight vector for the *user input node*. Note that $\boldsymbol{V}_u$ is a user-specific vector, *i.e.*, for each of the users we have one unique vector. From another point of view, $\boldsymbol{W}_i$ and $\boldsymbol{V}_u$ can be seen as the distributed representations of item i and user u respectively.

In the output layer, there are I nodes representing reconstructions of the input vector $\boldsymbol{y}_u$. The nodes in the output layer are fully connected with nodes in the hidden layer. The weight matrix is denoted by $\boldsymbol{W}' \in \mathbb{R}^{I \times K}$. We denote the weight vector for the bias node in the hidden layer by $\boldsymbol{b}' \in \mathscr{R}^I$.

Formally, the inputs of CDAE are the corrupted feedback vector $\tilde{\boldsymbol{y}}_u$ which is generated from $p(\tilde{\boldsymbol{y}}_u|\boldsymbol{y}_u)$ as stated in Equation 4.9. Intuitively, the non-zero values in $\boldsymbol{y}_u$ are randomly dropped out independently with probability q. The resulting vector $\tilde{\boldsymbol{y}}_u$ is still a sparse vector, where the indexes of the non-zero values are a subset of those of the original vector.

CDAE first maps the input to a latent representations $\boldsymbol{z}_u$, which is computed as follows:[5]

$$\boldsymbol{z}_u = h\left(\boldsymbol{W}^\top \tilde{\boldsymbol{y}}_u + \boldsymbol{V}_u + \boldsymbol{b}\right), \tag{4.10}$$

[5] Here we compute the hidden representation using the sum of the weight vectors. Other choices such as concatenation and max pooling are also possible.

Table 4.2: Sample Mapping Functions. Note that all the operations in this table are element-wise.

	$h(\boldsymbol{x})$	Gradient $\frac{\partial h}{\partial \boldsymbol{x}}$
Identity	$\boldsymbol{x}$	$\mathbf{1}$
Sigmoid	$\sigma(\boldsymbol{x})$	$\sigma(\boldsymbol{x})(\mathbf{1}-\sigma(\boldsymbol{x}))$
Tanh	$\tanh(\boldsymbol{x})$	$\mathbf{1}-\tan^2(\boldsymbol{x})$

where $h(\cdot)$ is an *element-wise* mapping function (e.g., *identity function* $h(\boldsymbol{x}) = \boldsymbol{x}$ or *sigmoid function* $h(\boldsymbol{x}) = \sigma(\boldsymbol{x}) = 1/(1+e^{-\boldsymbol{x}})$), and $\boldsymbol{b} \in \mathbb{R}^K$ is the offset vector.

At the output layer, the latent representation is then mapped back to the original input space to reconstruct the input vector. The output value $\hat{y}_{ui}$ for node i is computed as follows:

$$\hat{y}_{ui} = f\left(\boldsymbol{W}_i'^{\top} \boldsymbol{z}_u + b_i'\right), \tag{4.11}$$

where $\boldsymbol{W}' \in \mathbb{R}^{I \times K}$ and $\boldsymbol{b}'$ are the weight matrix and the offset vector for the output layer, respectively. $f(\cdot)$ is also a mapping function.

We learn the parameters of CDAE by minimizing the average reconstruction error:

$$\arg\min_{\boldsymbol{W},\boldsymbol{W}',\boldsymbol{V},\boldsymbol{b},\boldsymbol{b}'} \frac{1}{U}\sum_{u=1}^{U} \mathbb{E}_{p(\tilde{\mathbf{y}}_u|\mathbf{y}_u)}\left[\ell\left(\tilde{\mathbf{y}}_u, \hat{\mathbf{y}}_u\right)\right] + \mathscr{R}\left(\boldsymbol{W}, \boldsymbol{W}', \boldsymbol{V}, \boldsymbol{b}, \boldsymbol{b}'\right), \tag{4.12}$$

where $\mathscr{R}$ is the regularization term to control the model complexity. Here we use the squared $L2$ Norm.

$$\mathscr{R}(\cdot) = \frac{\lambda}{2}\left(\|\boldsymbol{W}\|_2^2 + \|\boldsymbol{W}'\|_2^2 + \|\boldsymbol{V}\|_2^2 + \|\boldsymbol{b}\|_2^2 + \|\boldsymbol{b}'\|_2^2\right) \tag{4.13}$$

We apply Stochastic Gradient Descent (SGD) to learn the parameters. Because the number of output nodes equals the number of items, the time complexity of one iteration over all users is $O(UIK)$, which is impractical when the number of users and the number of items are large. Instead of computing the gradients on all the outputs, we only sample a subset of the negative items $\mathscr{S}_u$ from $\bar{\mathscr{O}}_u$ and compute the gradients on the items in $\mathscr{O}_u \cup \mathscr{S}_u$. The size of $\mathscr{S}_u$ is proportional to the size of $\mathscr{O}_u$. So the overall complexity of the learning algorithm is linear in the size of $\mathscr{O}$ and the number of latent dimensions K. A similar method has been discussed in [15]. An alternative solution is to build a *Hierarchical Softmax* tree on the output layer [24], but it requires the loss function on the output layer to be softmax loss.

Recommendation. At prediction time, CDAE takes user u's existing reference set (without corruption) as input, and the items from the candidate set $\bar{\mathscr{O}}_u$ that have largest prediction values on the output layer are recommended to him/her.

4.3.2 Discussion

Components. CDAE is very flexible in that the mapping function $h(\cdot)$, point-wise or pair-wise objectives, the loss function $\ell(\cdot)$, and the corruption probability q can all be chosen to suit the application. Table 4.2 lists mapping function choices explored in this chapter. As for the loss function, all the choices discussed in Section 4.1.1, both *point-wise* and *pair-wise*, can be used. Different choices of these functions result in different variants of the model with different representation capabilities. Our experiments show that no single variant always produces the best results. One benefit of the proposed general framework is that one can try several variants and find the one that best fits the task.

Generalization of other models. CDAE is a generalization of latent factor models. The representations mentioned in Section 4.1.1 can all be interpreted as special cases of this framework.

Specifically, if we choose the *identity* mapping function for both $h(x)$ and $f(x)$ and not add noise to the inputs, the output value of $\hat{y}_{ui}$ in Equation 4.11 becomes:

$$\begin{aligned}\hat{y}_{ui} &= f\left(\boldsymbol{W}_i'^{\top}\boldsymbol{z}_u + b_i'\right) \\ &= \boldsymbol{W}_i'^{\top} h\left(\boldsymbol{W}^{\top}\tilde{\boldsymbol{y}}_u + \boldsymbol{V}_u + \boldsymbol{b}\right) + b_i' \\ &\cong \boldsymbol{W}_i'^{\top}\left(\sum_{j\in\mathcal{O}_u}\tilde{y}_{ui}\boldsymbol{W}_j + \boldsymbol{V}_u\right).\end{aligned} \tag{4.14}$$

In the last step we omit the bias term to make the comparison clearer. We can see that the representation in Equation 4.14 is equivalent to that in Equation 4.5, *i.e.*, the **LFSM** model.

If we set the corruption level q to 1, all the non-zero values in the input vector would be dropped out. We get the following prediction:

$$\hat{y}_{ui} = \boldsymbol{W}_i'^{\top}\boldsymbol{V}_u, \tag{4.15}$$

which is equivalent to the representation in Equation 4.2, *i.e.*, the LFM model. Alternatively, if we remove the *user input node* and its associated weights, the resulting model is equivalent to **FSM** in Equation 4.4:

$$\hat{y}_{ui} = \boldsymbol{W}_i'^{\top}\left(\sum_{j\in\mathcal{O}_u}\tilde{y}_{ui}\boldsymbol{W}_j\right). \tag{4.16}$$

Another possible mapping function is the *linear* function $h(\boldsymbol{x}) = \boldsymbol{U}^{\top}\boldsymbol{x}$, where $\boldsymbol{U}$ is a $K \times K$ transform matrix. If we use a user-specific matrix $\boldsymbol{U}_u \in \mathbb{R}^{K\times K}$ on the hidden layer, the representation becomes

$$\hat{y}_{ui} = \boldsymbol{W}_i'^{\top}\left(\boldsymbol{U}_u^{\top}\left(\sum_{j\in\mathcal{O}_u}\tilde{y}_{ui}\boldsymbol{W}_j\right)\right), \tag{4.17}$$

which is related to the Latent Collaborative Retrieval model proposed in [67].

Table 4.3: Dataset Statistics

	#users	#items	#dyads	density(%)
ML	69K	8.8K	5M	0.82
Netflix	37K	11K	4.8M	1.18
Yelp	9.6K	7K	243K	0.36

Summary. CDAE is a flexible framework for top-N recommendation. It generalizes several very popular existing methods. The proposed framework is naturally compatible with the denoising trick, which can further improve the results of recommendation.

4.4 Experimental Results

Our experimental evaluation consists of two parts. First, we study the effects of various choices of the components of CDAE. Second, we compare CDAE against other state-of-the-art top-N recommendation methods.

4.4.1 Data Sets and Experimental Setup

We use 3 popular data sets: MovieLens 10M (ML)[6], Netflix[7] and Yelp (from Yelp Dataset Challenge[8] in 2014). For each data set, we keep those with ratings no less than 4 stars and treat all other ratings as missing entries. Those ratings that are retained are converted to a y_{ui} score of 1. This processing method is widely used in previous work on recommendation with implicit feedback (e.g., [44, 74, 21]). We iteratively remove users and items with fewer than 5 ratings. For each user, we randomly hold 20% of the ratings in the test set, and put the other ratings in the training set. The statistics of the resulting data sets are shown in Table 4.3.

4.4.2 Implementation Details

We perform 5-fold cross validation on the training data sets to select the best hyperparameters for all the models, and then use the best hyperparameters to train models on the whole training data sets.

We use Stochastic Gradient Decent (SGD) to learn the parameters for both the proposed method and comparison partners. AdaGrad [11] is used to automatically adapt the step size during the learning procedures. We set $\beta = 1$ and try different step sizes $\eta \in \{1, 0.1, 0.01, 0.001\}$ and report the best result for each model.

[6] http://grouplens.org/datasets/movielens

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

[8] http://www.yelp.com/dataset_challenge

For negative sampling, we experiment with different numbers of negative samples and find that $NS = 5$ consistently produces good results. This means that, for each user, the number of negative samples is 5 times the number of observed ratings of this user.

4.4.3 Evaluation Metrics

In the case of top-N recommender systems, we present each user with N items that have the highest predicted values but are not adopted by the user in the training data. We evaluate different approaches based on which of the items are actually adopted by the user in the test data.

Precision and Recall. Given a top-N recommendation list $C_{N,\text{rec}}$, precision and recall are defined as

$$\begin{aligned} \text{Precision@}N &= \frac{|C_{N,\text{rec}} \bigcap C_{\text{adopted}}|}{N} \\ \text{Recall@}N &= \frac{|C_{N,\text{rec}} \bigcap C_{\text{adopted}}|}{|C_{\text{adopted}}|}, \end{aligned} \tag{4.18}$$

where C_{adopted} are the items that a user has adopted in the test data. The precision and recall for the entire recommender system are computed by averaging the precision and recall over all the users, respectively.

Mean Average Precision (MAP). Average precision (AP) is a ranked precision metric that gives larger credit to correctly recommended items in top ranks. AP@N is defined as the average of precisions computed at all positions with an adopted item, namely,

$$\text{AP@}N = \frac{\sum_{k=1}^{N} \text{Precision@}k \times \text{rel}(k)}{\min\{N, |C_{\text{adopted}}|\}}, \tag{4.19}$$

where $\text{Precision}(k)$ is the precision at cut-off k in the top-N list $C_{N,\text{rec}}$, and $\text{rel}(k)$ is an indicator function equaling 1 if the item at rank k is adopted, otherwise zero. Finally, Mean Average Precision (MAP@N) is defined as the mean of the AP scores for all users.

Usually, these metrics are consistent with each other, *i.e.*, if a model performs better than another model on one metric, it is more likely that it will also produce better results on another metric. Due to space limits, we mainly show the results of MAP@N with $N = \{1, 5, 10\}$ on several evaluation tasks since it takes the positions into consideration.

4.4.4 Analysis of CDAE Components

The main components of the proposed CDAE model include the types of the mapping functions, the loss function and the level of corruption. Different choices of these components result in different variants of the model that make different top-N recommendations. In this subsection, we study these variants on the 3 data sets.

For the mapping function, we show results for the *identity* function and *sigmoid* function on the hidden layer and the output layer. (Results for the *tanh* function are similar to those of the *sigmoid* function and hence omitted.) There are $2^3 = 8$ total combinations of choices for the mapping functions (on both layers) and the loss function. Among them, the logistic loss function requires $\hat{y}$ to be a value between 0 and 1, so it must be associated with a sigmoid function on the output layer. Note that the combination of the sigmoid function and the logistic loss is equivalent to the cross entropy loss discussed in Section 4.1.1. Therefore, we study 4 variants[9] of our model in this subsection. Table 4.4 describes the function choices for each variant.

Table 4.4: Four possible variants of the CDAE model.

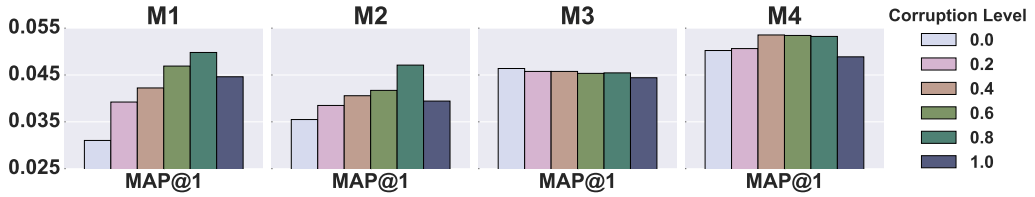
	Hidden Layer	**Output Layer**	**Loss Function**
M1	Identity	Identity	Square
M2	Identity	Sigmoid	Logistic
M3	Sigmoid	Identity	Square
M4	Sigmoid	Sigmoid	Logistic

In our extensive experiments, we observed that the pair-wise objective function did not perform much better than point-wise objectives for CDAE. One possible cause is that for the implicit feedback with binary ratings, point-wise loss functions are sufficient for separating those items preferred by the user and those not preferred. In other words, a well-designed point-wise loss can be discriminating enough to model the user's preference in these datasets, and the pair-wise loss is not needed. For this reason, results on pair-wise objective functions are omitted in the rest of this chapter. On a related note, as we show in section 4.4.5, the BPR model, which uses pair-wise loss, does not perform better than MF, which uses point-wise loss. Similar results have also been reported in [21]. This might be due to the same reason as for CDAE. Moreover, BPR is designed to optimize for the AUC, not top-N metrics such as precision, recall, or MAP. Hence, for multiple models for top-N recommendation, pair-wise loss functions may not be necessary for all data sets.

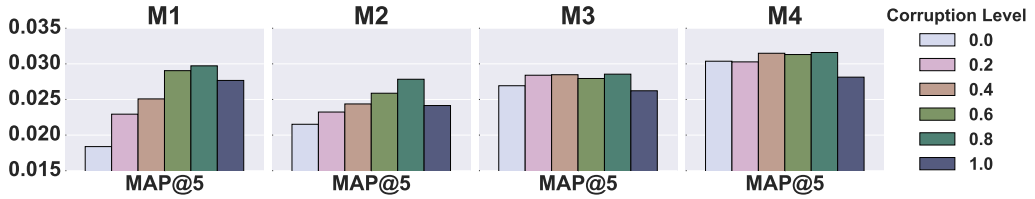
We train each of the four variants under varying corruption levels (q in Equation 4.9) from $\{0, 0.2, 0.4, 0.6, 0.8, 1\}$. The number of latent dimensions K is set to 50 unless otherwise stated. The results on the three data sets are shown in Figure 4.2, 4.3 and 4.4 respectively.

The general observation is that the best model depends on the data set. No single variant of CDAE always produces the best results. So one should choose the components (mapping function, objective function, loss function, and corruption level) depending on the data. Consider the two different extremes of corruption level: $q = 0$ (no input corruption) and $q = 1$ (complete input corruption). For variant M1, the results of $q = 0$ are much worse than those of $q = 1$. This indicates that simply summing up all the input vectors ($q = 0$) is insufficient for learning good representations, and is in fact worse than dropping out all of them ($q = 1$). Note that introducing non-linear functions can largely alleviate the problem, as evidenced in the results for the other three variants with $q = 0$. Adding noise on the input can also prevent this problem (see the results of M1 with

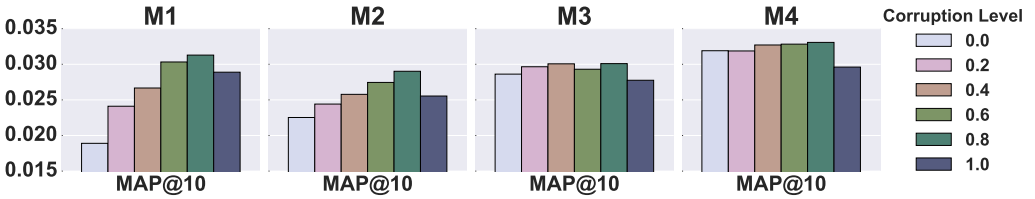
[9] We omit the results of another 2 variants (replacing the loss functions of M2 and M4 with square loss) since their performances are similar to those of M2 and M4 respectively.



(a) MAP@1

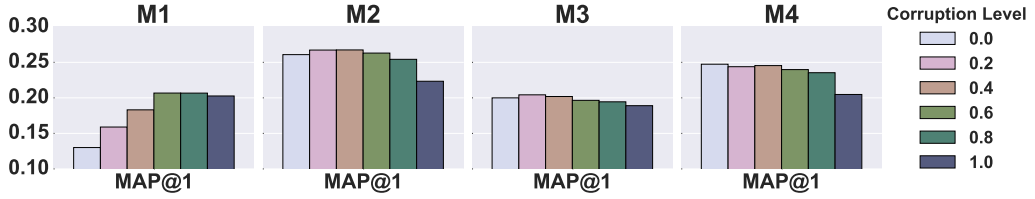


(b) MAP@5

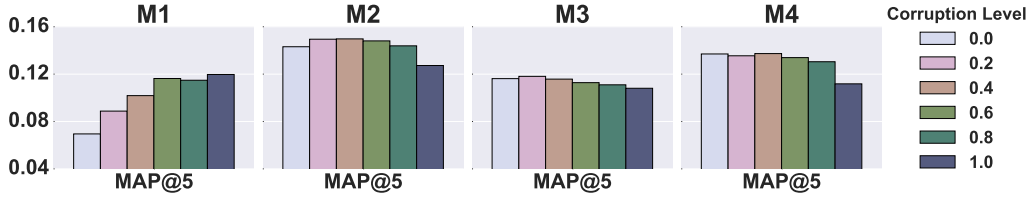


(c) MAP@10

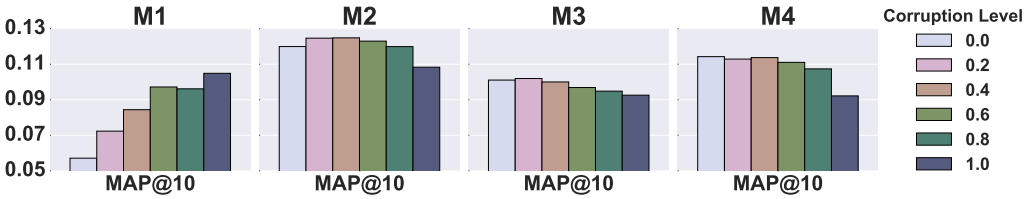
Figure 4.2: Model performance comparison on Yelp data.



(a) MAP@1



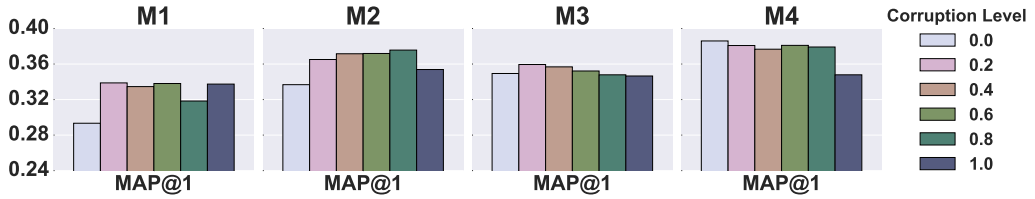
(b) MAP@5



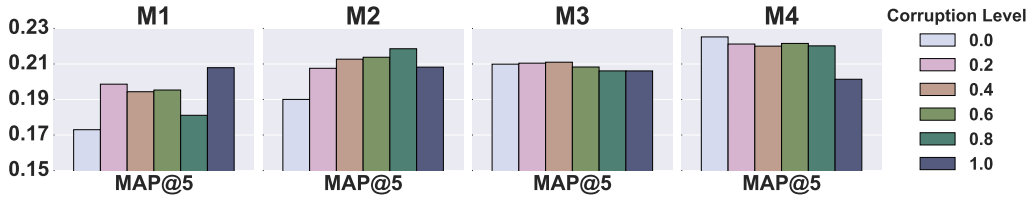
(c) MAP@10

Figure 4.3: Model performance comparison on Netflix data.

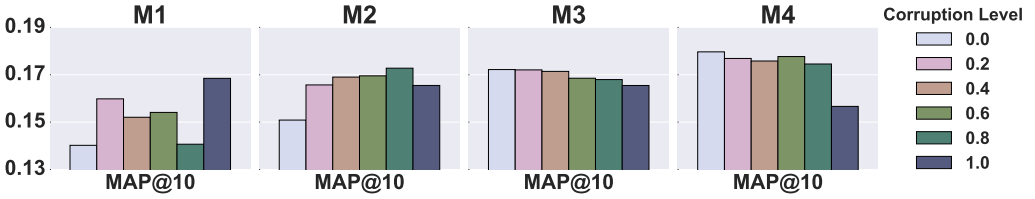
various corruption levels), which means that the denoising technique can help with learning more robust representations.



(a) MAP@1



(b) MAP@5



(c) MAP@10

Figure 4.4: Model performance comparison on MovieLens data.

Table 4.5: Comparison with DAE on the Yelp data.

	MAP@1		MAP@5		MAP@10	
Model	**DAE**	**CDAE**	**DAE**	**CDAE**	**DAE**	**CDAE**
M1	0.0275	0.0327	0.0164	0.0201	0.0172	0.0210
M2	0.0369	0.0420	0.0225	0.0241	0.0236	0.0254
M3	0.0443	0.0460	0.0270	0.0285	0.0289	0.0303
M4	0.0529	0.0528	0.0315	0.0319	0.0329	0.0334

The denoising technique appears beneficial especially for variants M1 and M2. On the Yelp data set, all four variants can be improved by adding relatively higher levels of noise (e.g., $q = 0.8$). Variant M2 is the best model for the Netflix data set, and setting $q = 0.2$ and $q = 0.4$ can slightly improve the results. On MovieLens data, setting $q = 0.8$ makes M2 almost as good as M4, the best model. However, in some cases, the best results are those without any input corruption ($q = 0$).

In general, M4 produces relatively better results on all three data sets. In particular, it achieves the best MAP scores on Yelp and MovieLens. This indicates that non-linear functions help to increase the representation capability of the model, thus improving the recommendations.

Comparison with DAE. A main difference between CDAE and classical DAE is the user-specific input between the input layer and the hidden layer, namely, the vector $\boldsymbol{V}_u$. We study two cases here – with the user-specific vectors (CDAE) and without the user-specific vectors (DAE). We conduct experiments on the three data sets, and get relatively similar results. We show the results on the Yelp data and on the MovieLens data in Table 4.5 and 4.6 respectively. We can see that for

Table 4.6: Comparison with DAE on the MovieLens data.

	MAP@1		MAP@5		MAP@10	
Model	**DAE**	**CDAE**	**DAE**	**CDAE**	**DAE**	**CDAE**
M1	0.2807	0.2933	0.1619	0.1730	0.1278	0.1402
M2	0.3134	0.3368	0.1785	0.1900	0.1408	0.1509
M3	0.3270	0.3494	0.1953	0.2099	0.1579	0.1722
M4	0.3625	0.3860	0.2123	0.2252	0.1684	0.1797

Table 4.7: Effects of using tied weights on MovieLens data. "TW" means using tied weights, while "NTW" means no tied weights.

	MAP@1		MAP@5		MAP@10	
Model	**TW**	**NTW**	**TW**	**NTW**	**TW**	**NTW**
M1	0.1739	0.2933	0.0983	0.1730	0.0763	0.1402
M2	0.3707	0.3368	0.2086	0.1901	0.1643	0.1509
M3	0.3482	0.3494	0.2044	0.2099	0.1669	0.1722
M4	0.3530	0.3860	0.2007	0.2252	0.1609	0.1797

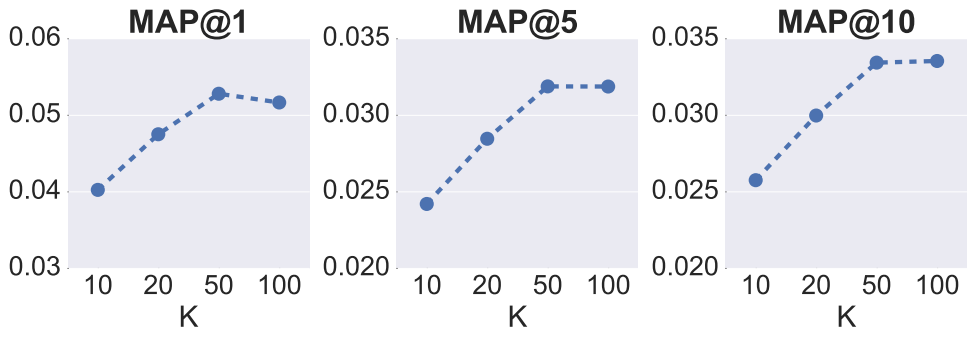
Table 4.8: Effects of using tied weights on Netflix data. "TW" means using tied weights, while "NTW" means no tied weights.

	MAP@1		MAP@5		MAP@10	
Model	**TW**	**NTW**	**TW**	**NTW**	**TW**	**NTW**
M1	0.1172	0.1301	0.0551	0.0695	0.0428	0.0571
M2	0.2567	0.2608	0.1418	0.1431	0.1177	0.1199
M3	0.1172	0.2000	0.0551	0.1162	0.0428	0.1011
M4	0.2287	0.2474	0.1260	0.1370	0.1066	0.1143

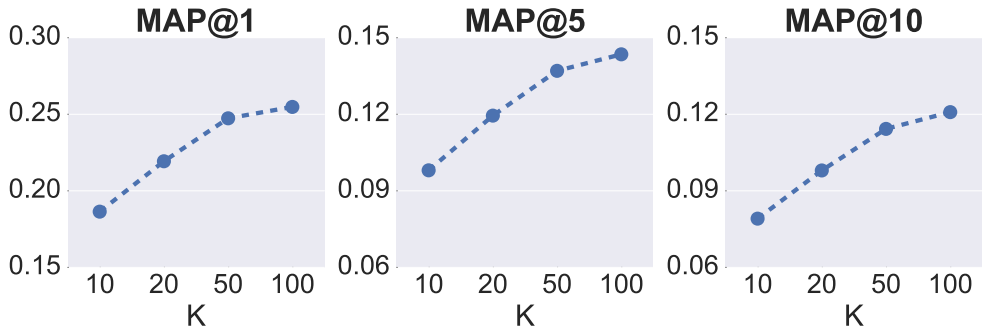
models M1 and M2, using user-specific vectors greatly improves the results on the Yelp data. As the performances of the models get better for M3 and M4, the gain becomes relatively smaller. For the MovieLens data, using user-specific vectors consistently outperforms the alternative choice.

Tied weights. We study the effects of using tied weights (TW) for the weight matrices, where we force $\boldsymbol{W} = \boldsymbol{W}'$. Results on MovieLens and Netflix data sets are shown in Table 4.7 and 4.8, respectively. We refer to the cases of "no tied weights" as NTW. The results on Yelp data are similar to those on Netflix data, so we omit them here. Other than variant M2 on MovieLens data, the results of NTW are much better than TW. On Netflix data, NTW consistently outperforms TW by at least 10%. On both data sets, the best MAP scores are from models with NTW (M4+NTW in Table 4.7 and M2+NTW in Table 4.8, respectively). Thus we do not recommended using tied weights for CDAE.

The number of latent dimensions. We study the effects of the number of latent dimensions K. Results on Yelp data and Netflix data are shown in Figure 4.5. From the figures, we can see that the performance increases with larger K, but only up to a point. When K becomes large enough, the performance no longer improves and can in fact decrease due to overfitting.



(a) Results on Yelp data.



(b) Results on Netflix data.

Figure 4.5: The effects of the number of latent dimensions.

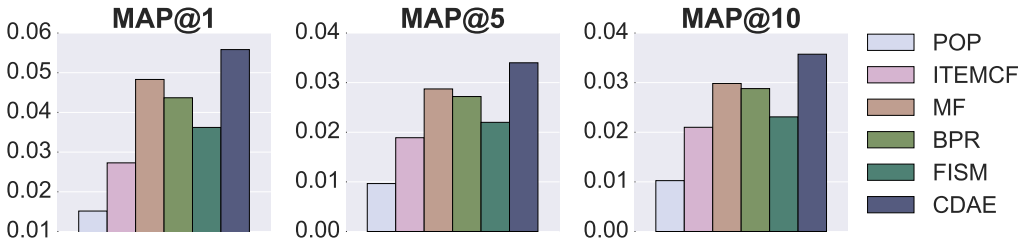


Figure 4.6: MAP scores with different N on the Yelp data set.

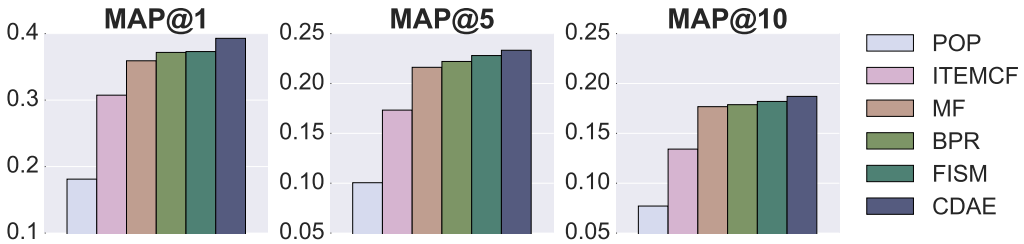


Figure 4.7: MAP scores with different N on the MovieLens data set.

4.4.5 Experimental Comparisons with Previous Models

In this section, we compare CDAE with a number of popular top-N recommendation methods. Note that comparisons against Denoising Auto-Encoder (DAE) and its non-denoising variant (when $q = 0$) are already discussed in Section 4.4.4.

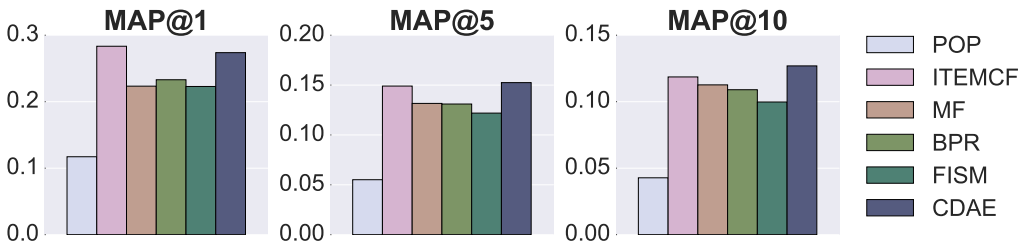


Figure 4.8: MAP scores with different N on the Netflix data set.

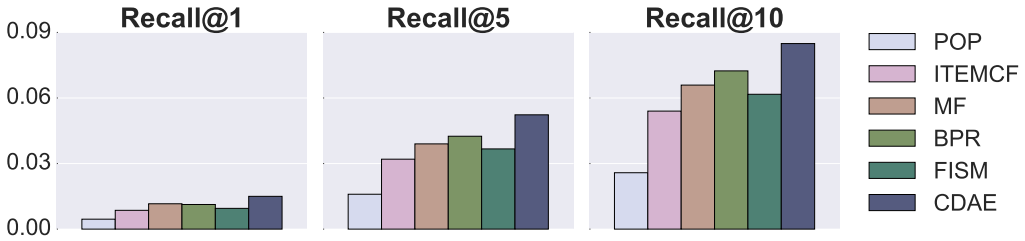


Figure 4.9: The Recall scores with different N on the Yelp data set.

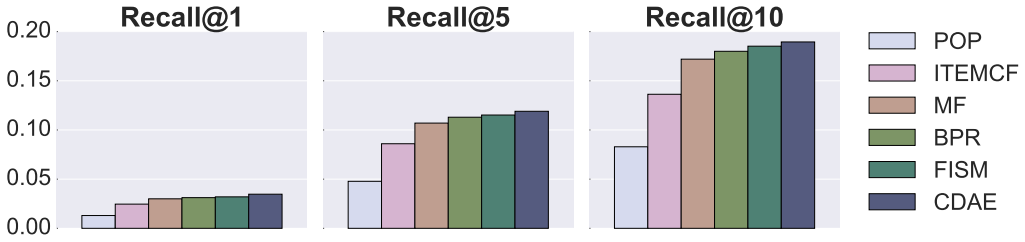


Figure 4.10: The Recall scores with different N on the MovieLens data set.

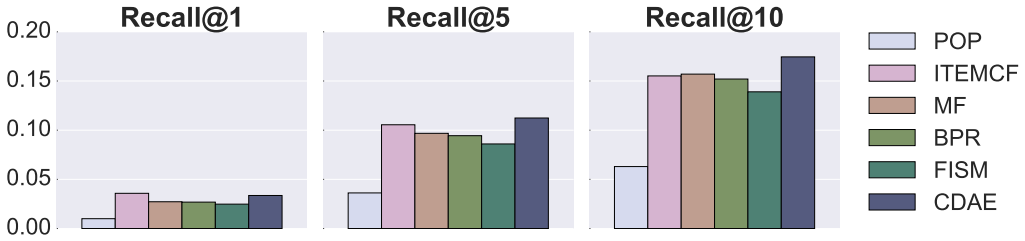


Figure 4.11: The Recall scores with different N on the Netflix data set.

These baseline methods are:

- **POP**: Items are recommended based on how many users have rated them.
- **ITEMCF** [50]: We use Jaccard similarity measure and set the number of nearest neighbor to 50.

- **MF (with negative sampling)** [23]: We train the Latent Factor Model with point-wise objective functions (with square, log, hinge and cross entropy losses) and negative sampling[10], and report the best results.

- **BPR** [44]: BPR is the state-of-the-art method for recommendation based on implicit feedback. As discussed in Section 4.1.1, BPR-MF is a Latent Factor Model with the pair-wise log loss function. In addition to the log loss used in the original paper [44], we also experiment with square and hinge loss functions and report the best results.

- **FISM** [21]: FISM is a variant of FSM with point-wise square loss function[11]. We also test log loss and hinge loss and report the best results.

For all the baseline methods, we carefully choose the hyperparameters by cross validation to ensure fair comparisons. We train the 4 variants of CDAE discussed in Table 4.4 with different corruption levels, and report the best results. For all the latent factor models (including MF, BPR, FISM and CDAE), we set the number of latent dimensions to 50 and use an additional dimension to model the bias. Other implementation details are as discussed in Section 4.4.2.

Figure 4.6, 4.7 and 4.8 show the MAP@N scores of all models on Yelp, MovieLens and Netflix, respectively. Since Recall is another widely used metric for comparing top-N recommendation models, we also include plots of the Recall scores on the three data sets in Figure 4.9, 4.10 and 4.11.

In general, the results of MAP and Recall are consistent, *i.e.*, the performance orderings of models are almost the same. One exception is on the Yelp data, where MF gets better MAP@N scores than BPR, but BPR has better Recall@N scores.

According to the results of MAP@10 and Recall@10, CDAE consistently outperforms other compared methods. On the Yelp data, CDAE outperforms the other methods with a large margin on all the evaluation metrics. The MAP@10 score and Recall@10 score of CDAE are at least 15% better than those of the second best model MF. For the Netflix data set, ITEMCF achieves much better results than other methods such as MF, BPR and FISM, particularly on the metrics MAP@1 and Recall@1. CDAE is the only model that can beat ITEMCF on metrics MAP@10 and Recall@10, where CDAE outperforms ITEMCF by around 10%.

It is surprising to see that BPR and FISM achieve lower MAP scores than MF on Yelp and Netflix data sets. The only data set on which they achieve better results is MovieLens, but the performance gains are not significant.

[10] We note that, for implicit feedback data, MF with negative sampling has the same objective function with WRMF [19] – both of them assign high confidence on the observed/positive feedback and low cofidence on the missing/negative feedback. We do not compare with WRMF because its computational speedup trick for ALS only works with squared loss.

[11] We also tried the pair-wise loss functions, but the results are not as good as for the point-wise functions. The same observation is reported in the original paper.

4.5 Conclusion

In this chapter, we presented the Collaborative Denoising Auto-Encoder (CDAE) for the top-N recommendation problem. CDAE learns distributed representations of the users and items via formulating the user-item feedback data using a Denoising Auto-Encoder structure. Several previous work can be seen as special cases of the proposed model. We conducted a comprehensive set of experiments on three data sets to study how the choice of the model components impacts the performance. We also compared CDAE against several other state-of-the-art top-N recommendation methods and the results show that CDAE outperforms the rest of the methods by a large margin.

The proposed model enables a wide range of future work on applying neural networks to recommender systems. Here we list some potential directions.

Deep Neural Network. The neural network structure used in this chapter is shallow. A straightforward extension is to stack the model as done in the stacked DAE [58]. Our preliminary experiments on the stacked CDAE do not show significant improvement over CDAE. We plan to investigate the reason and try to improve it. Also, the idea of marginalized DAE [9] might be able to speed up the training and improve the performance. It would also be interesting to consider applying other neural network structures such as Convolutional Neural Networks and Recurrent Neural Networks to this framework.

Feature-aware Recommendation. User and item features can be important for producing semantically meaningful models and dealing with the *cold-start* problem. It is worth exploring how to incorporate user and item features to improve the proposed model.

Context-aware Recommendation. In many applications, one might benefit from incorporating contextual information (such as time, location, browser session, etc.) into the recommendation process in order to recommend items to users in certain circumstances. We leave such extensions as future work.

Chapter 5

CCCF: On Better Utilizing the Matrix Structure

Typical Collaborative Filtering methods measure users' preferences by their historical behaviors over the entire item space. For example, the user-based nearest neighbors method measures the similarity between pairs of users by their preferences on all the items; Matrix Factorization decomposes the whole user-item matrix into two low-rank sub-matrices where the collective intelligences are represented as latent factors in the model. However, this assumption does not always hold, especially when the underlying system includes a large set of items of different types. For example, an Amazon user share similar tastes on books with a certain group of users, while having similar preferences with another group of users on movies. Therefore, it is more natural to model the users and the items using subgroups, where each subgroup includes a set of like-minded users and the subset of items that they are interested in, and each user/item can be assigned to multiple subgroups so that users can share their interests with different subsets of users on different subsets of items. Figure 5.1 plots an example of the underlying co-clusters in a user-item matrix. In this chapter, we will describe a new method CCCF which uses co-clustering to better utilized the matrix structure to get better recommendation for users.

5.1 Background and Overview

Partitioning users and items into subgroups for CF has been studied by several previous works, where user clustering [70], item clustering [39] and user-item co-clustering [14] methods have been proposed to boost the performance of collaborative filtering. However, in these methods each user/item is only allowed to be assigned to a single subgroup, so that they are not able to model the case where users have multiple interests. To address this problem, several other papers [52, 63, 64] extend the Mixed Membership Stochastic Blockmodels (MMSB) [2] to allow mixed memberships. However, these methods optimize the accuracy of rating/link prediction instead of the practically important problem of top-N recommendation.

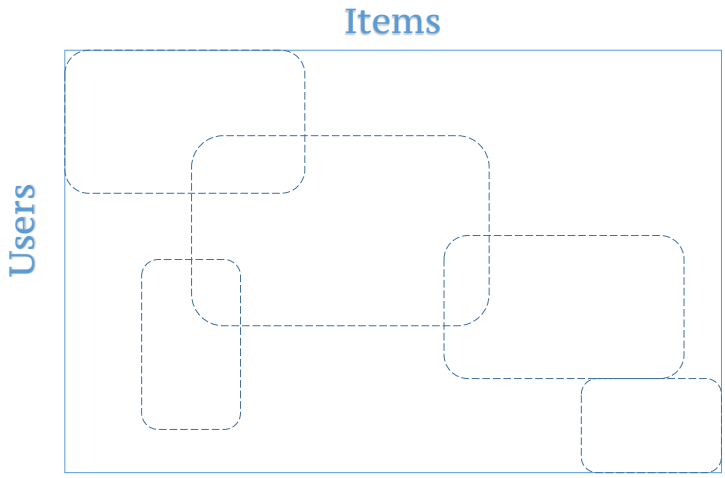


Figure 5.1: Illustration on the overlapping co-clusters.

In [69], the authors propose a unified framework for improving collaborative filtering via overlapping co-clustering, which can be employed for top-N recommendation. It works as follows: 1) Users and items are first grouped into multiple overlapping subgroups with different weights. 2) After obtaining the subgroups, any traditional CF method can be applied to each subgroup separately to make the predictions for the users on their unrated items in the same subgroup. 3) The final recommendation list for a user is aggregated from the results in the subgroups that she/he is involved in. The authors of [77] adopt a similar framework, and propose to partition the user-item matrix by permuting the rows and columns of the user-item matrix to form (approximate) Bordered Block Diagonal matrices.

However, the co-clustering objective function of [69] is based on a weighted graph cut problem, which partitions the underlying graph into non-overlapping subgroups and minimizes the total costs associated with the cut edges. Therefore, [69] does not *intrinsically* learn overlapping subgroups. Also, solving the weighted graph cut problem has a high computational complexity. In [77], assignments of users and items to subgroups are deterministic and have equal weight, *i.e.*, users are forced to have equal amount of interest in each subgroup that they are assigned to, thus affiliation strengths are not modeled.

We argue that in order to model users' shared interests in different subsets of items, we have to resort to overlapping co-clustering methods which have the following three properties:

- The proposed method should be able to model the strength of affiliations between users, items and subgroups by assigning weights to each user and item for each subgroup. As we shall see later in this work, these weights are critical when aggregating the results from subgroups to make the final recommendation.

Table 5.1: Mathematical Notations

Symbol	Value	Description
y_{ui}	$\{0,1\}$	user u's feedback on item i
ϕ_{uk}	$[0,1]$	latent membership of user u in subgroup k
ϕ_{ik}	$[0,1]$	latent membership of item i in subgroup k
θ_k	$[0,1]$	in-group preference strength of subgroup k
$z_{u\to i,k}$	$\{0,1\}$	membership indicator of user u in subgroup k when interacting with item i
$z_{i\to u,k}$	$\{0,1\}$	membership indicator of item i in subgroup k when interacting with user u

- The proposed method should be able to model a user's interest in an item through her/his affiliations with different subgroups.
- The proposed method should have the property that the more subgroups that a user u and an item i share, and the higher the affiliation strengths of u and i with the subgroups they share, the more likely u should be interested in i.

In this chapter, we propose a novel co-clustering method, called CCCF (CO-CLUSTERING FOR COLLABORATIVE FILTERING). In CCCF, each user/item can belong to multiple subgroups, and we use an affiliation strength score to denote the strength of the relationship between the user/item and the corresponding subgroup. CCCF models the probability of a user liking an item as a function based on subgroup affiliation strengths between users, items and their shared subgroups. If a user and an item share multiple subgroups, we assume that each such shared subgroup has an independent chance to trigger the link[1] between them [71]. As will be discussed in Section 5.4, the resulting CCCF model satisfies the three desired properties stated above.

5.2 Related Work

Using co-clustering based methods to improve CF has been studied by several previous works. [70] uses the K-means algorithm to cluster users into several subgroups and uses the subgroup assignments to smooth the unrated data for each individual user. [14] proposes a scalable CF framework based on a weighted co-clustering method.

[1] Here a link between a user and an item means the user likes the item. If there is no link between them, we treat it as missing.

Bayesian Co-Clustering (BCC) [52] adopts the Mixed Membership Stochastic Blockmodels (MMSB) [2] to the user-item bipartite graph. They assume that there are several latent user clusters and item clusters, and each user/item has mixed memberships to the latent clusters. The link between a user and an item is generated by the sum of the coefficient weights between the clusters they belong to. [63] proposes a collapsed Gibbs sampling and a collapsed variational Bayesian algorithm for BCC. [64] further extends BCC to a nonparametric model that can automatically learn the number of clusters from the data. In [5], the user and item latent factors are described by a nonparametric mixture of Gaussians based on their cluster allocations. [4] uses additive co-clustering to build concise model for matrix approximation. In these works, users and items are *separately* partitioned into user clusters and item clusters, which is totally different from co-clustering method used in this chapter which groups users and items into the same clusters in a holistic manner. Also, these works optimize either the likelihood of the probabilistic mixed membership model on all the user-item links [52, 63, 64], or the accuracy of the point-wise rating prediction [5, 4], which can not guarantee good top-N recommendation results as they are optimizing a different goal [44, 55]. On the other hand, our method can utilize state-of-the-art top-N recommendation models as the base CF method in subgroups.

The idea of divide-and-conquer has also been used in other related works for collaborative filtering [31, 26, 25]. Divide Factor Combine (DFC) [31] divides a large-scale matrix factorization task into smaller subproblems by random row/column selections, solves each subproblem in parallel using an arbitrary base matrix factorization algorithm, and combines the subproblem solutions using techniques from randomized matrix approximation. The authors of [26] propose a model called Local Low-Rank Matrix Approximation (LLORMA), leading to a representation of the observed matrix as a weighted sum of several local low-rank matrices. They further extend LLORMA to the Local Collaborative Ranking (LCR) model [25] to consider the relative orders of items. The main differences from ours are: 1) They partition the user-item matrix by either random row/column sampling [31] or random anchor points [26, 25], while our goal is to find locally focused subgroups. 2) They focus on the rating prediction [31, 26] and collaborative ranking [25] problems on observed ratings[2] and ignore the fact that ratings are missing not at random. It leads to unsatisfactory top-N recommendation results [44, 55]. 3) They rely on the matrix factorization techniques, while ours can apply any CF method in the subgroups. This is important because matrix factorization is not always the best choice for all the cases, which has also been shown in our experiments (see Section 5.5.3).

The closest works are [69, 77, 78], which propose models that cluster users and items into the same subgroups. The authors of [69] propose an overlapping user-item co-clustering method based on a weighted graph-cut optimization problem. The objective function is NP-Hard, and they propose to optimize a relaxed problem. However, the learning involves solving the top eigenvectors of a large squared matrix so it cannot scale to large data sets. In [77] and [78], the authors permute

[2] Although LCR [25] models the orders of items, it still only considers the ranking between *observed* ratings.

rows and columns of the user-item rating matrix to form (approximate) Bordered Block Diagonal (BBD) matrices, and then apply traditional CF to each BBD matrix.

The framework used in our thesis, as well as in [69, 77], has several benefits. 1) It partitions the matrix into several overlapping sub-matrices, which are denser than the original matrix, thus making the CF methods more feasible. 2) Any CF method can be used in each subgroup. Since the performance of different methods varies under different scenarios, this allows users to select their favorite CF base methods for the subgroups. 3) The training of the CF methods in subgroups can be trivially parallelized.

As discussed in Section 5.4.4, compared with previous works [69, 77], CCCF has various benefits including scalability, flexibility, interpretability and extensibility.

5.3 Problem Definition

Given a set of users $\mathscr{U} = \{u|u = 1,...,U\}$, a set of items $\mathscr{I} = \{i|i = 1,...,I\}$, as well as the log of users' historical preferences $\mathscr{O} = (u,i,y_{ui})$, the goal of recommender systems is to recommend to each user u a list of items that will maximize her/his satisfaction. Here, y_{ui} can be numeric ratings in the scale of, say, $[1,5]$ or binary values $\{0,1\}$.

In this chapter, we mainly focus on the case of *implicit* feedback, namely, we only have a partial information of the items that users have viewed/liked, and users' feedback on other items is missing. This is the most common scenario in practice. However, the proposed method can also be applied to other cases with slight modifications. For implicit feedback, all y_{ui} in $\mathscr{O}$ are 1; and for those $(u',i',y_{u'i'})$ triples not belonging to $\mathscr{O}$, the corresponding $y_{u'i'}$'s are missing. We use $\mathscr{O}'$ to denote the set of missing triples. The goal of top-N recommendation is to recommend each user a list of N items that she/he is most likely to like, *i.e.*, to pick the list of missing $y_{u'i'}$-s for every user which are most likely to be 1.

Some important notations used in this chapter are listed in Table 5.1. We use u to index a user, and i and j to index items, and use normal font symbol x to denote scalars and bold math symbols $\boldsymbol{x}$ to denote vectors. The k-th element of vector $\boldsymbol{x}_i$ is denoted by x_{ik}.

5.4 Proposed Methodology

The overall procedure of our method, also used by [69, 77], is as follows:

1. Cluster users and items into subgroups, where users and items can be assigned to multiple subgroups and each subgroup includes a group of like-minded users and a set of items that these users are particularly interested in. We present a scalable co-clustering method called CCCF in Section 5.4.1.

2. In each subgroup, we apply traditional collaborative filtering methods (*e.g.*, ITEMCF, MATRIX FACTORIZATION) to learn users' preferences over the items within this subgroup. Since

the learning of CF models in each subgroup is independent from each other, this step can be easily done in parallel (See Section 5.4.2).

3. For each user, we aggregate the recommendation results from all the subgroups that she/he is involved in. We discuss the aggregation strategy in Section 5.4.3.

The main differences from previous works [69, 77] lie in step 1 and 3, which will be explained in detail in Section 5.4.1 and 5.4.3, respectively. In Section 5.4.4, we elaborate the benefits of the proposed method compared to those related works.

5.4.1 Scalable Co-Clustering for CF

In this subsection, we present Co-Clustering for Collaborative Filtering (CCCF), a co-clustering model which assigns users and items into overlapping subgroups. Our goal is to find several latent subgroups, where each subgroup includes a set of like-minded users and the subset of items they are interested in. We assume that each user/item has a latent membership vector over subgroups. The memberships reveal users' latent interests on different types of items and items' latent properties that would be consumed by users, respectively. The links between users and items are generated based on the interactions between users, items and subgroups, *i.e.*, a user has larger chance to like an item if they both have strong affiliations to the same subgroups.

The two principles of designing CCCF are:

- A user has multiple types of interests so that she/he should be assigned to multiple subgroups with different strengths. Analogously, items are also allowed to be affiliated to multiple subgroups.

- The reason that a user likes an item is explained by the subgroup affiliations, *i.e.*, a user is more likely to like an item *if and only if* they belong to same subgroups. And we assume that the probability of a user liking an item depends on two ingredients: 1) The more overlapping subgroups a user shares with an item, the higher the probability that the user likes the item is. 2) A link between a user and an item can be explained by a dominant reason, *i.e.*, if a user and an item both have large affiliation strengths with a single subgroup, it is sufficient to form the link between them.

We assume there are K subgroups and each user/item has a probability of belonging to each subgroup. We denote by $\phi_{uk} \in [0,1]$ the affiliation strength of user u to subgroup k, and $\phi_{ik} \in [0,1]$ the affiliation strength of item i to subgroup k. We sample ϕ_{uk} and ϕ_{ik} from Beta distributions parameterized by predefined constants α_{k1} and α_{k2}.

$$\begin{aligned} \phi_{uk} &\sim \text{Beta}\left(\alpha_{k1}, \alpha_{k2}\right) \\ \phi_{ik} &\sim \text{Beta}\left(\alpha_{k1}, \alpha_{k2}\right) \end{aligned} \tag{5.1}$$

Intuitively, ϕ_{uk} represents the probability that user u likes the items in subgroup k. Analogously, ϕ_{ik} is the probability that item i is liked by the users in subgroup k. Items having large memberships in subgroup k share some latent properties recognized by a group of like-minded users.

The reasons why we use Beta distribution for ϕ_{uk} and ϕ_{ik} are: 1) In our work, we define $\boldsymbol{\phi}$ as membership variables with values in $[0,1]$, and we do not enforce $\boldsymbol{\phi}_u$ to form a probabilistic distribution over the set of all the subgroups as the Mixed Membership models (*e.g.*, [2, 52]) do. Instead, users and items can have large affiliation strengths with multiple subgroups, which has been shown to be more suitable for modeling overlapping subgroups [71, 72]. 2) Beta distribution is the conjugate prior of Bernoulli distribution, which we use for the indicator variable z later. This enables us to perform sampling steps more efficiently.

We assume that each subgroup k has an independent chance θ_k to trigger the link between user u and item i if both of them belong to this subgroup, and the overall probability relies on the rate that at least one of the K event succeeds.

Here θ_k acts as a parameter to model the strength of connectivity within each subgroup k, meaning that the larger θ_k is, the more likely it is to form links between users and items within this subgroup. This parameter can model the fact that in practice, some subgroups are more active than others. Also, if a user belongs to this subgroup, there is a large chance of her/him liking an item shared by users within this subgroup, whereas for the less active subgroups, the chance of her/him liking an item shared by users within the subgroup is lower.

We also place a Beta distribution as the prior of θ_k:

$$\theta_k \sim \text{Beta}\left(\beta_1, \beta_2\right). \tag{5.2}$$

Since our goal in this chapter is to find several focused subgroups, we'd like all the learned θ_k-s being close to 1. This prior knowledge can be incorporated into the model by, *e.g.*, setting $\beta_1 = 10$ and $\beta_2 = 1$.

For a pair of user u and item i, we use a parameter $z_{u\to i,k}$[3] to denote whether user u belongs to subgroup k or not when forming the link with item i. $z_{u\to i,k}$ is an indicator parameter drawn from a Bernoulli distribution parameterized by user u's group affiliation weight ϕ_{uk}. The parameter $z_{i\to u,k}$ which represents item i's affiliation with subgroup k for the link with user u is defined analogously.

$$\begin{aligned} z_{u\to i,k} &\sim \text{Bernoulli}\left(\phi_{uk}\right) \\ z_{i\to u,k} &\sim \text{Bernoulli}\left(\phi_{ik}\right) \end{aligned} \tag{5.3}$$

Then, the probability p_{uik} that subgroup k triggers a link between user u and item i is defined as

$$p_{uik} = \begin{cases} \theta_k, & \text{if } z_{u\to i,k} = 1 \text{ and } z_{i\to u,k} = 1 \\ 0, & \text{otherwise.} \end{cases} \tag{5.4}$$

[3] Here the notation $u \to i$ is used to denote user u's assignment when forming the link with the item i. Similar notations are also used in, *e.g.*, [2].

Note that, if either the user or the item do not belong to subgroup k, the probability of forming a link via this subgroup is 0.

The overall probability p_{ui} that user u likes item i is modeled as the overall success rate of the K independent events, each of which represents the probability of a subgroup k to trigger the link between u and i.

$$p_{ui} = 1 - \prod_k (1 - p_{uik}) = 1 - \prod_k (1 - \theta_k)^{z_{u \to i,k} z_{i \to u,k}} \tag{5.5}$$

The link between user u and item i is sampled as follows:

$$y_{ui} \sim \text{Bernoulli}\,(p_{ui})\,, \tag{5.6}$$

where $y_{ui} = 1$ means user u has a link with item i and $y_{ui} = 0$ means the link between them is missing.

We show the generative process in Algorithm 2. Note that in addition to using the subgroup affiliations to explain the observed links with $y_{ui} = 1$, the CCCF model also needs to consider the missing links (where $y_{ui} = 0$) properly to avoid non-meaningful subgroup affiliations. However, the number of missing links is usually almost as large as $U \cdot I$ due to the data sparsity, which makes it impractical for large data sets. To address this issue, we randomly sample a subset of the missing links during the inference, and alternate different subsets in different iterations. Specifically, we randomly select N_u missing links for each user u, where N_u is proportional to the number of u's observed links.

Parameter Estimation

Maximum likelihood estimation is intractable for this model due to the existence of hidden variables. We use a Markov Chain Monte Carlo (MCMC) method to estimate the parameters where we iteratively sample unknown variables from their conditional distributions given all the observations and the other variables fixed.

Denoting the set of items that user u has links with in the training data (including the observed links and randomly sampled negative links) by $\mathscr{T}_u$, we have the following sampling equation for $z_{u \to i,k}$ (latent variables $\boldsymbol{\phi}$ have been integrated out):

$$\begin{aligned} & p(z_{u \to i,k} = 1 | z_{\neg(u \to i,k)}, \boldsymbol{y}, \boldsymbol{\theta}, \alpha_{k1}, \alpha_{k2}) \\ \propto & \left(n_{uk}^{\neg i} + \alpha_{k1}\right) p(y_{ui} | \boldsymbol{z}_{\neg(u \to i,k)}, z_{u \to i,k} = 1, \boldsymbol{\theta}), \\ & p(z_{u \to i,k} = 0 | z_{\neg(u \to i,k)}, \boldsymbol{y}, \boldsymbol{\theta}, \alpha_{k1}, \alpha_{k2}) \\ \propto & \left(n_u - 1 - n_{uk}^{\neg i} + \alpha_{k2}\right) p(y_{ui} | \boldsymbol{z}_{\neg(u \to i,k)}, z_{u \to i,k} = 0, \boldsymbol{\theta}), \end{aligned} \tag{5.7}$$

where $n_{uk}^{\neg i} = \sum_{j \in \mathscr{T}_u \setminus \{i\}} \mathbf{1}(z_{u \to j,k} = 1)$ and $n_u = |\mathscr{T}_u|$

Since we sample different subsets of missing links during different iterations, a missing link might be new at iteration t. In this case, we do not have previous statistics of $z_{u \to i,k}$ and $z_{i \to u,k}$ to

Algorithm 2 Generative Process of CCCF.

Require: $\mathscr{O}$, $\mathscr{O}'$, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$

for all $k \in \{1,...,K\}$ **do**
  for all $u \in \mathscr{U}$ **do**
    Draw $\phi_{uk} \sim \text{Beta}(\alpha_{k1}, \alpha_{k2})$
  end for
  for all $i \in \mathscr{I}$ **do**
    Draw $\phi_{ik} \sim \text{Beta}(\alpha_{k1}, \alpha_{k2})$
  end for
  Draw $\theta_k \sim \text{Beta}(\beta_1, \beta_2)$
end for
for all $(u, i, y_{ui}) \in \mathscr{O} \cup \mathscr{O}'$ **do**
  for all $k \in \{1,...,K\}$ **do**
    Draw $z_{u \to i,k} \sim \text{Bernoulli}(\phi_{uk})$
    Draw $z_{i \to u,k} \sim \text{Bernoulli}(\phi_{ik})$
    if $z_{u \to i,k} = 1$ and $z_{i \to u,k} = 1$ **then**
      Set $p_{uik} = \theta_k$
    else
      Set $p_{uik} = 0$
    end if
  end for
  Set $p_{ui} = 1 - \prod_k (1 - p_{uik})$
  Draw $y_{ui} \sim \text{Bernoulli}(p_{ui})$
end for

sample their new values according to Equation 5.7. Inspired by the stochastic collapsed variational Bayesian inference method for Latent Dirichlet Allocation [13], we use some additional *burn-in* steps to update these statistics before doing the sampling.

Since it is intractable to directly sample the posterior of each θ_k, we update these parameters using Metropolis-Hasting steps, where in each iteration a candidate for the next sample is generated from a *"proposal"* distribution. The proposed sample is accepted with some probability, otherwise the previous sample is duplicated. To be specific, in each iteration we propose each θ_k's new value from $\theta_k \sim \text{Beta}(\beta_1, \beta_2)$ and accept this new value with probability

$$\min\left(1, \frac{P(\boldsymbol{y}|\boldsymbol{z}, \boldsymbol{\theta}^{(t)}) P(\boldsymbol{\theta}^{(t)}|\boldsymbol{\beta})}{P(\boldsymbol{y}|\boldsymbol{z}, \boldsymbol{\theta}^{(t-1)}) P(\boldsymbol{\theta}^{(t-1)}|\boldsymbol{\beta})}\right). \tag{5.8}$$

If the new value is rejected, the previous value is retained.

The overall inference procedure is shown in Algorithm 3. After we get the optimized parameters $\boldsymbol{\theta}$ and $\boldsymbol{z}$, the group memberships $\boldsymbol{\phi}$ can be computed by

$$\phi_{uk} = \frac{\sum_{i \in \mathscr{T}_u} z_{u \to i,k} + \alpha_{k1}}{n_u + \alpha_{k1} + \alpha_{k2}}, \tag{5.9}$$

Algorithm 3 Inference Procedure for CCCF

Require: Randomly initialize $\boldsymbol{z}$. Set each $\theta_k = 1$.
Ensure: $\boldsymbol{z}$, $\boldsymbol{\theta}$, $\boldsymbol{\phi}$

```
while not converged or maximum number of iterations do
    Randomly sample a subset of missing values O'' and set the training set as T = O ∪ O''.
    for all (u, i, y_ui) ∈ T do
        for all k ∈ {1, ..., K} do
            Sample z_{u→i,k} using Equation 5.7.
            Sample z_{i→u,k} similarly.
        end for
    end for
    while not converged do
        for all k ∈ {1, ..., K} do
            Update θ_k using Equation 5.8
        end for
    end while
end while
Compute φ_u and φ_i for all u and i using Equation 5.9 and 5.10.
```

$$\phi_{ik} = \frac{\sum_{i \in \mathscr{T}_i} z_{i \to u,k} + \alpha_{k1}}{n_i + \alpha_{k1} + \alpha_{k2}}. \tag{5.10}$$

5.4.2 Collaborative Filtering in Subgroups

After getting the group affiliation strengths, we assign to each subgroup k a threshold ε_k. If user u's (or item i's) affiliation strength ϕ_{uk} (ϕ_{ik}) is larger than ε_k, we assume that this user/item belongs to this subgroup. Then, *any* Collaborative Filtering algorithm can be applied to a subgroup to compute the prediction $\hat{y}^k_{ui}$, which represents user u's preference on item i based on subgroup k.

The choice of the underlying CF methods depends on a lot of factors, such as data sparsity, complexity, model interpretability and latency of recommendation. One benefit of the framework is that the method does not rely on a specific CF algorithm. Any CF method of interest can be used and the framework will likely improve their performances. In our experiments, we choose four most representative CF models as the base methods applied to the subgroups, and show that the framework improves their recommendation performances by a significant margin on four real world data sets.

To make this chapter self-contained, here we discuss the four CF methods which we use in our experiments. The reason why we choose these methods is that they cover a diverse set of methods ranging from the simplest solution to state-of-the-art models.

Popularity (POP).. Items are scored by the percentage of users who like them.

$$y_{ui} = \sum_{u'} \frac{y_{u'i}}{|\mathscr{U}|} \tag{5.11}$$

Though simple, POP is widely used in today's industry as it is extremely easy to implement. The disadvantage is that it is not a personalized method and all the users receive the same recommendations. This problem can be solved by the proposed framework. CCCF first partitions users and items into subgroups. The item popularity within a specific subgroup indicates the popularity among a small group of people. Using the recommendation method that we will discuss in Section 5.4.3, CCCF provides users personalized recommendations by aggregated ranking lists of items that are popular in the subgroups that users are particularly interested in.

ITEMCF. [50]. We use Jaccard correlation as the similarity measure and set the number of nearest neighbors to 50.

$$\hat{y}_{ui} = \sum_{j \in \mathscr{O}_u} s(i,j), \tag{5.12}$$

where $s(i,j)$ is the Jaccard similarity between the set of users who have liked item i and j, respectively.

MF (with negative sampling). [23, 48]. Matrix Factorization, or Latent Factor Model is the most popular Collaborative Filtering method for recommender systems. The preference of user u on item i is the dot product of the latent feature vectors of the user and the item, with some bias terms.

$$\hat{y}_{ui} = \alpha + b_u + b_i + \boldsymbol{u}_u^\top \boldsymbol{v}_i \tag{5.13}$$

The model parameters, latent factors and bias term, are learned by optimizing the following objective function.

$$\sum_{(u,i) \in \mathscr{T}} \ell(y_{ui}, \hat{y}_{ui}) + \lambda \mathscr{R}(\boldsymbol{b}, \boldsymbol{u}, \boldsymbol{v}), \tag{5.14}$$

where $\mathscr{R}$ is the regularization term.

For implicit feedback, we sample a subset of missing values as negative samples, and re-sample the subset for each learning iteration.

WARP. [66, 67]. Matrix factorization with the WARP loss [66] is the state-of-the-art method for top-n recommendation on implicit feedback data, which has been used by many recent works [17, 68]. The main difference between WARP and MF is the objective function. WARP models the relative orders between items, which is the goal of top-N recommendation.

Another way of making recommendation in the subgroup is just setting $\hat{y}^k_{ui} = \theta_k$. However, this makes a strong assumption that users in a subgroups have the same preferences for all the items within this subgroup, which will result in suboptimal recommendation results. We will discuss this case in Section 5.4.3 and compare it with other recommendation methods in Section 5.5.3.

5.4.3 Recommendation

The last but not the least question is to how to compute the final recommendation list for each user. In [69], the authors just keep several subgroups with the largest weights for each user and sum up

the recommendation scores from the selected subgroups. In [77], the proposed algorithm gets the final prediction of user u on an item i by averaging the predicted scores of u on i in all subgroups they share.

However, users and items belong to the subgroups with different weights, and the final recommendation should not ignore the weights. In this chapter, we propose to use an affiliation strength weighted aggregation function to compute the final recommendations:

$$\hat{y}_{ui} = \sum_k \phi_{uk} \cdot \phi_{ik} \cdot \theta_k \cdot \hat{y}^k_{ui} \tag{5.15}$$

This aggregation function matches our motivation of designing the co-clustering method: 1) The prediction $\hat{y}^k_{ui}$ has a large value if the user and the item both have large affiliation strengths with this subgroup; 2) The more subgroups the user and the item share, the larger their predicted score is. The comparison with other aggregation methods is discussed in Section 5.5.3.

We note that for some CF methods (*e.g.*, WARP [66], BPR [44]), the prediction score $\hat{y}^k_{ui}$ is only a relative measurement (*i.e.*, $\hat{y}_{ui} > \hat{y}_{uj}$ means user u prefers item i than j) and unbounded. Before aggregating the scores, we first scale all the predictions of a user to $[0,1]$.

In this following we also present three other strategies to aggregate the results.

CCCF-PR.. As discussed in Section 5.4.2, the most straightforward way is just using the in-group strength θ_k-s as the prediction scores for the subgroups and calculating the overall score as follows:

$$\hat{y}_{ui} = \sum_k \phi_{uk} \cdot \phi_{ik} \cdot \theta_k \tag{5.16}$$

Comparing with Equation 5.15, this strategy sets all $\hat{y}^k_{ui}$ to 1. From another point of view, it can be thought as applying the *random* prediction (all the items have the same prediction score 1) base method in the subgroups.

CCCF-AVG.. This is the strategy used in [78], where the average predicted score is taken as the overall score, ignoring the affiliation strengths of users and items.

$$\hat{y}_{ui} = \sum_k \hat{y}^k_{ui} / K \tag{5.17}$$

CCCF-MAX.. As discussed in Section 5, a user would like an item as long as the probability on one of the subgroup is large enough. Another strategies is to make the prediction using the maximum score among these subgroups.

$$\hat{y}_{ui} = \max_k \hat{y}^k_{ui} \tag{5.18}$$

5.4.4 Discussion

In this subsection, we discuss several properties of the proposed model.

- **Scalability.** The complexity of the proposed CCCF method is linear in the number of observed links. On the other side, the objective function of the method proposed in [69] is NP-hard, and their optimization method for the relaxed objective involves a step of solving the top eigenvectors of a squared matrix with size $M = U + I$. To our best knowledge, the computational complexity of the fastest eigenvalue decomposition solver is $O(M^{2.367})$ when the matrix has some special structures [41].
- **Flexibility.** The structure of the subgroups in CCCF is flexible to model any kind of overlapping subgroups where these subgroups can be densely overlapping, hierarchically nested or non-overlapping. The methods in [78] and [77] are not so flexible since they assume the subgroups to form (approximate) Bordered Block Diagonal matrices. The method proposed in [69] uses an objective function of a weighted graph cut problem, which is designed to partition the graph into non-overlapping parts, thus it does not intrinsically discover overlapping subgroups.
- **Interpretability.** CCCF is a probabilistic model where the resulting affiliation strengths have semantic meanings. The strengths are particularly important when aggregating the results from different subgroups to make final recommendation (see Section 5.4.3). Also, as shown in Section 5.5, we analyze the top items with the largest affiliation strengths of each subgroup and find that these items have similar properties (*e.g.*, locations, categories in the Yelp data).
- **Extensibility.** When we are able to get some useful features for users and items, it is easy for CCCF to leverage the features in the learning procedure as prior knowledge or supervisions (*e.g.*, [1], [73]). On one hand, it makes the subgroups even more interpretable by enforcing that the users/items in the same subgroup share some common features. On the other hand, features can help dealing with the cold-start problem – some users/items may have too few connections to be assigned to the right subgroups. It is not straightforward for the methods of [69] and [77] to achieve this goal.

5.5 Experiments

Our experiments are designed to answer the following two questions: 1) Can the proposed method improve the accuracy of recommendation? 2) How meaningful are the discovered subgroups?

5.5.1 Data Sets and Preprocessing

We use four publicly available data sets: Last.fm[4], MovieLens 10M[5], Netflix[6] and Yelp[7]. As discussed in Section 5.3, we are more interested in the *implicit* feedback case. We follow the preprocessing steps that have been used in many recent works (*e.g.*, [44, 74], etc.). For the data sets

[4] https://grouplens.org/datasets/hetrec-2011/
[5] https://grouplens.org/datasets/movielens/
[6] http://www.netflixprize.com/
[7] http://www.yelp.com/dataset_challenge/

Table 5.2: Data Statistics

	#users	#items	#dyads	#density(%)
Last.fm	1.8K	1.5K	50K	1.76
MovieLens	35K	5.6K	3.4M	1.73
Netflix	46K	13K	8M	1.33
Yelp	2.8K	2.8K	80K	1.02

with explicit ratings, we remove the ratings of less than 4 stars and convert the remaining ratings to 1. Cold-start users/items are not the focus of this chapter, so we remove users and items with less than 20 ratings. The statistics of the resulting data sets are shown in Table 5.2. For each user, we randomly hold 20% of her/his feedback in the test data and use the rest of her/his ratings as training data.

5.5.2 Evaluation Measures

In the case of top-N recommendation, we present each user with N items that have the highest predicted values and have not been adopted by the user in the training data. We evaluate different approaches based on which of the items are actually adopted by the user in the test data.

Precision and Recall.. Given a top-N recommendation list $C_{N,\text{rec}}$, precision and recall are defined as

$$\begin{aligned} \text{P@}N &= \frac{|C_{N,\text{rec}} \bigcap C_{\text{adopted}}|}{N} \\ \text{R@}N &= \frac{|C_{N,\text{rec}} \bigcap C_{\text{adopted}}|}{|C_{\text{adopted}}|}, \end{aligned} \tag{5.19}$$

where C_{adopted} are the items that a user has adopted in the test data. The precision and recall for the entire recommender system are computed by averaging the precision and recall over all the users, respectively.

F-measure.. The F-measure represents a trade-off between precision and recall. We consider the F_β metric, which is defined as

$$\text{F}_\beta\text{@}N = (1+\beta^2) \cdot \frac{\text{P@}N \times \text{R@}N}{\beta^2 \cdot \text{P@}N + \text{R@}N}. \tag{5.20}$$

where β is a weight to control the balance. In our experiments, we use $F1$ metric with $\beta = 1$. We also calculate the $F1$ score for each user and report the average score.

Mean Average Precision (MAP).. Average precision (AP) is a ranked precision metric that gives larger credit to correctly recommended items in higher positions. AP@N is defined as the average of precisions computed at all positions with an adopted item, namely,

$$\text{AP@}N = \frac{\sum_{k=1}^{N} \text{P@}k \times \text{rel}(k)}{\min\{N, |C_{\text{adopted}}|\}}, \tag{5.21}$$

where Precision(k) is the precision at cut-off k in the top-N list $C_{N,\text{rec}}$, and rel(k) is an indicator function equaling 1 if the item at rank k is adopted, otherwise it equals zero. Finally, Mean Average Precision (MAP@N) is defined as the mean of the AP@N scores of all users.

5.5.3 Accuracy of Top-N Recommendation

Table 5.3: CCCF vs. No Co-Clustering on the Last.fm data

	POP		ITEMCF		MF		WARP	
	NONE	CCCF	NONE	CCCF	NONE	CCCF	NONE	CCCF
P@10	0.070	0.134	0.173	0.174	0.152	0.156	0.178	0.187
R@10	0.090	0.176	0.223	0.250	0.204	0.217	0.253	0.264
F1@10	0.078	0.149	0.189	0.198	0.170	0.177	0.203	0.213
MAP@10	0.182	0.355	0.446	0.456	0.399	0.402	0.428	0.442

Table 5.4: CCCF vs. No Co-Clustering on the Netflix data

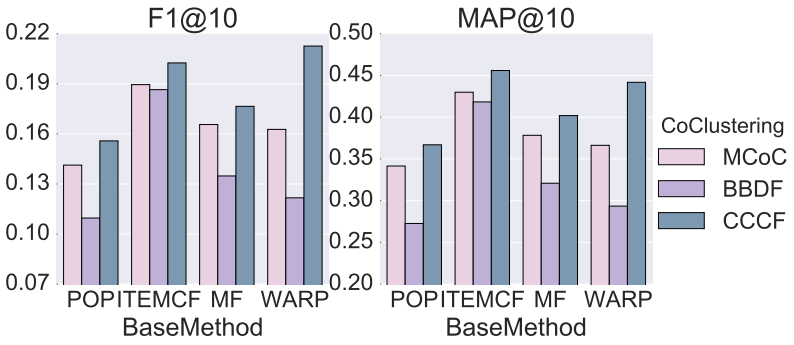
	POP		ITEMCF		MF		WARP	
	NONE	CCCF	NONE	CCCF	NONE	CCCF	NONE	CCCF
P@10	0.189	0.283	0.308	0.312	0.330	0.354	0.347	0.364
R@10	0.047	0.075	0.085	0.085	0.091	0.098	0.095	0.100
F1@10	0.070	0.109	0.123	0.123	0.132	0.142	0.138	0.145
MAP@10	0.348	0.471	0.493	0.505	0.528	0.551	0.538	0.556

Table 5.5: CCCF vs. No Co-Clustering on the Yelp data

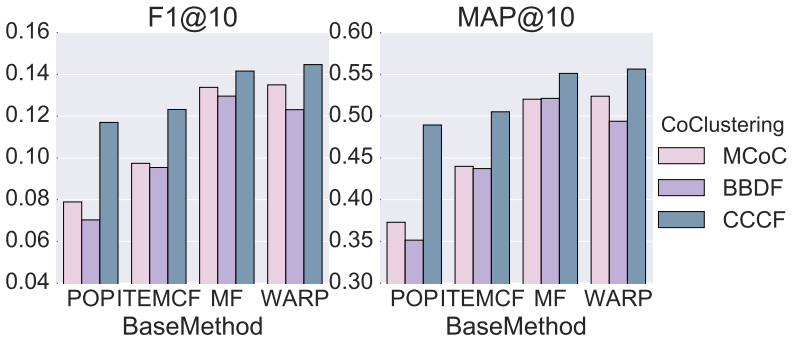
	POP		ITEMCF		MF		WARP	
	NONE	CCCF	NONE	CCCF	NONE	CCCF	NONE	CCCF
P@10	0.018	0.046	0.048	0.057	0.043	0.046	0.047	0.056
R@10	0.024	0.058	0.062	0.072	0.055	0.058	0.058	0.068
F1@10	0.018	0.046	0.048	0.058	0.044	0.046	0.047	0.055
MAP@10	0.050	0.124	0.142	0.159	0.124	0.131	0.137	0.152

Implementation Details

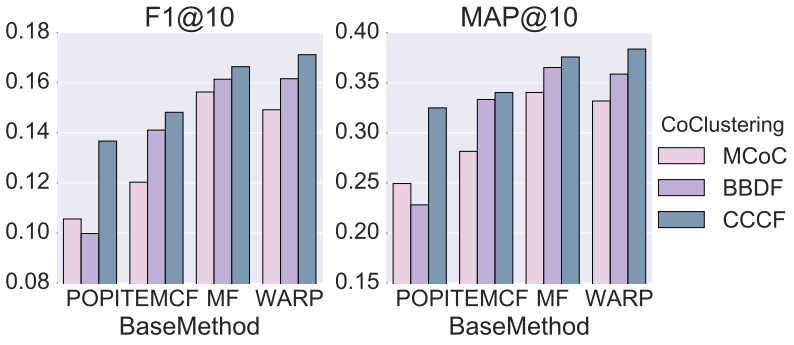
We choose 4 popular CF methods as the comparison partners, and these 4 methods also serve as the base methods applied to each subgroup (as discussed in Section 5.4.2). We first train CCCF on the training data, and then apply the above CF methods in each subgroup. At last, for each user, we predict her/his preferences on all the unrated items by aggregating the predictions from the subgroups the user is involved in, and pick the Top-N items with largest predicted values to form the recommendation list. We set the hyperparameters of CCCF by cross validation on the training data. We found that the performance is not so sensitive to the choices of α and β. We empirically set them as: $\alpha_{k1} = 1$, $\alpha_{k2} = 1$, $\beta_1 = 10$, $\beta_2 = 1$ and $\varepsilon = 0.1$. We will study how the choice of K impacts the performance in Section 5.5.3.



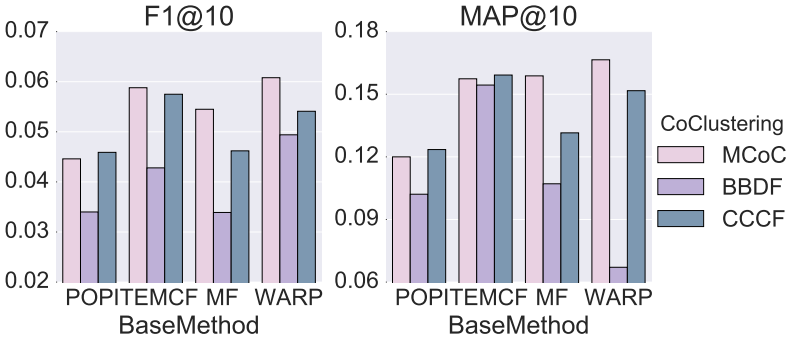
(a) Results on the Last.fm data



(b) Results on the Netflix data



(c) Results on the MovieLens data



(d) Results on the Yelp data

Figure 5.2: Comparison with other two co-clustering methods on the three data sets.

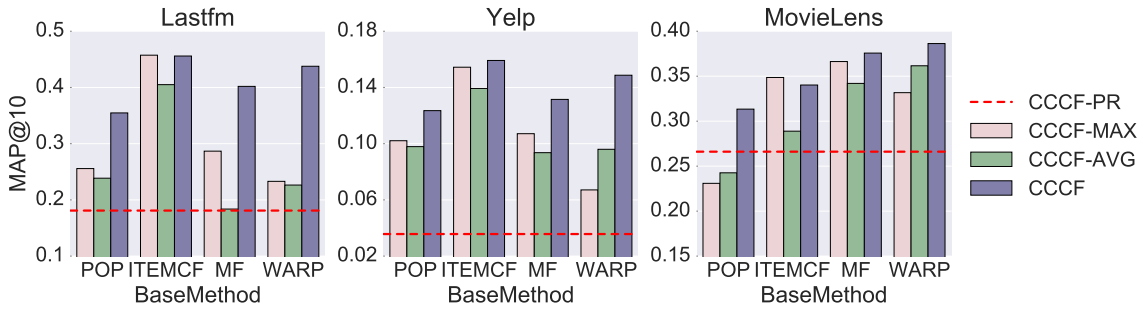
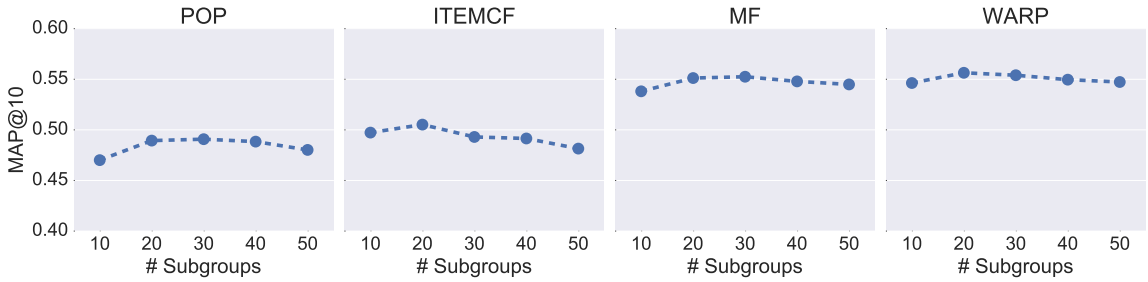
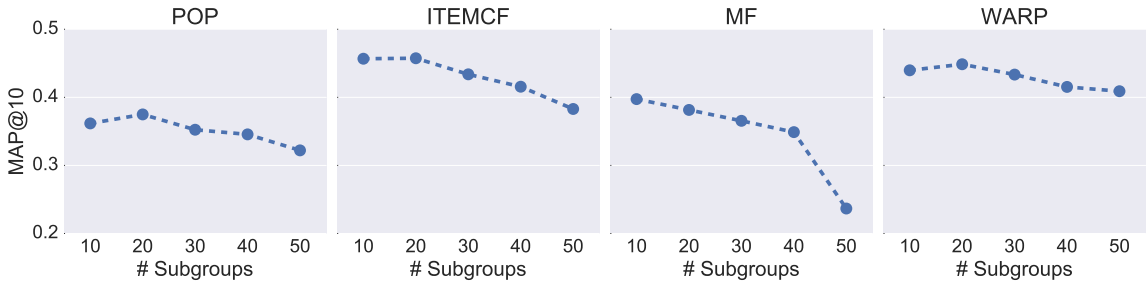


Figure 5.3: MAP@10 on the Last.fm and Yelp data with different strategies to aggregate the results from subgroups. CCCF-PR does not need a base method and we plot it using the dashed line.



(a) MAP@10 on the Netflix data



(b) MAP@10 on the Last.fm data

Figure 5.4: MAP@10 on the Last.fm and Netflix data with different numbers of subgroups.

CCCF vs. No Co-Clustering

We first study the performance of CCCF comparing with the CF methods that are directly applied on the training data without co-clustering.

The results are shown in Table 5.3, 5.4 and 5.5. The overall observation is that CCCF improves the base methods on most of the cases – the best scores on all the data sets are all from CCCF. For most of the cases, CCCF improves the corresponding base method by at least 10%. Especially, combining CCCF with simple recommendation methods like POP can produce fairly good results.

We note that the performance of these CF methods varies a lot on different data sets. For example, on the Yelp data set, we observe that ITEMCF performs better than MF and WARP; while WARP is the best method on the Netflix data set. This means that no single CF method can beat all

others at all times. The advantage of the proposed framework that we do not rely on the underlying base method, and users of our method are free to configure their favorite predictors in order to adapt to different scenarios.

CCCF vs. Other Co-Clustering Methods

We compare CCCF against two other co-clustering based methods MCoC [69] and BBDF [77], which are the most recent co-clustering methods for recommender systems, and adopt the same three-step framework. For MCoC and BBDF, we carefully choose the best parameters (*e.g.*, the number of top eigenvectors for MCoC, the density level in BBDF) to ensure fair comparisons. For both CCCF and MCoC, we set the number of subgroups to 10. The number of subgroups in BBDF is determined by the density parameter, which is chosen by cross validation.

Figure 5.2 shows the results for F1@10 and MAP@10 on the four data sets. The general observation is that CCCF outperforms the other two co-clustering methods in most cases. The only exception is on the Yelp data set, where both MCoC and CCCF achieve a large performance gain compared to BBDF, and MCoC obtains the best results on this data set. This is likely due to the fact that the users and items of the Yelp data are isolated by the geographical distance, *i.e.*, they are more likely to form subgroups by the cities they are in.

Surprisingly, in some cases, MCoC and BBDF even decrease the performance of some of the base methods on the Last.fm, Netflix and MovieLens data. As discussed in Section 5.4.4, MCoC is actually optimizing a non-overlapping clustering objective function. So on the data sets that do not have obvious clustering structures, the clustering result of MCoC fails to boost recommendation accuracy. The performance of BBDF is not as good as expected on all the data sets except MovieLens. One reason may be that it assumes the user-item matrix follows some specific structure, and not all the data sets have this property. For example, for the Yelp data set, users and items are more likely to form non-overlapping clusters, due to the geographical separations of users and items. In this case, it is impossible for BBDF to find a reasonable bordered-block sub-matrix, which produces unsatisfactory results.

CCCF vs. Local Low-Rank Matrix Factorization

In this subsection, we compare CCCF with recently proposed Local Low-Rank Matrix Factorization methods LLORMA[26] and LCR [25]. We note that the original LCR model described in their paper [25] is not applicable to the implicit feedback data. LCR optimizes the pairwise loss on the pairs of observed ratings. However, for implicit feedback where all the observed ratings are 1, pairwise comparisons make no sense. To make the comparison applicable, we modify LCR by applying the negative sampling strategy, which has also been used for BPR [44] and WARP [67] We set the number of anchor points of LLORMA and LCR same with the number of subgroups for CCCF, and select the hyperparameters (e.g., step size and regularization parameter) by cross validation.

Table 5.6: CCCF vs. Local Low Rank Matrix Factorization on the Yelp data

	P@10	R@10
LLORMA	0.0208	0.0306
LCR (with negative sampling)	0.0368	0.0543
WARP	0.0474	0.0582
CCCF + WARP	0.0556	0.0684

Table 5.7: CCCF vs. Local Low Rank Matrix Factorization on the Last.fm data

	P@10	R@10
LLORMA	0.042	0.057
LCR (with negative sampling)	0.148	0.201
WARP	0.178	0.253
CCCF + WARP	0.187	0.264

Here we show the results on the Yelp and Last.fm data sets in Table 5.6 and 5.7 (the results on other two data sets are similar). We can see that the results of LLORMA are far worse than those of other models. This validates the point that it is critical to consider missing ratings in the model. The modified LCR model is still not as good as WARP and CCCF+WARP. By analyzing the data and some intermediate results, we think that the main reasons might be as follows: 1) LCR/LLORMA chooses anchor points randomly, which might result in selecting some isolating anchor points or several anchor points from the same clusters, leaving a large number of users and items having small weights with all the anchor points. These users and items are rarely updated during learning due to the low weights. 2) The similarity kernel used in LLORMA is defined as the product of user similarity and item similarity. These two similarities are computed independently.

The co-clusters in CCCF act similarly as the anchor points in LCR. CCCF solves above two problems by jointly learning focused subgroups and the user/item subgroup affiliations. Combining the idea of CCCF with LCR to learn a joint Co-Clustering and Matrix Factorization model would be an interesting future work.

On the Choice of Aggregation Method

As discussed in Section 5.4.3, there are several alternative ways to aggregate recommendations from subgroups to make final recommendations: CCCF-PR, CCCF-MAX, CCCF-AVG. In this subsection, we compare these three alternatives with the method we proposed in Equation 5.15 (referred to as CCCF).

The MAP@10 results are shown in Figure 5.3. Here we use the same co-clustering results and CF base methods. The only difference between the comparisons is the recommendation strategy. For the base method ITEMCF, CCCF-MAX can provide similar performance as CCCF, but its results for other methods are not competitive. For the Yelp data, CCCF-PR is outperformed by other aggregation methods by a large margin. This means that the step of applying a CF method in the subgroup is necessary. For almost all the cases, CCCF outperforms alternatives, which indicates

Table 5.8: Items with largest affiliation strengths from 3 out of 10 subgroups on the Yelp data. We show their locations and the most representative categories that these items belong to.

Sample Cluster 1			Sample Cluster 2			Sample Cluster 3		
Business Name	**City**	**Description**	**Business Name**	**City**	**Description**	**Business Name**	**City**	**Description**
Gallo Blanco	Phoenix	Restaurants	Bread and Butter	Henderson	Restaurants	Las Vegas North Premium Outlets	Las Vegas	Shopping
Lux	Phoenix	Restaurants	Snow Ono Shave Ice	Las Vegas	Restaurants	Orleans Hotel & Casino	Las Vegas	Casinos & Hotels
Postino Central	Phoenix	Restaurants	Buldogis Gourmet Hot Dogs	Las Vegas	Restaurants	Fremont Street Experience	Las Vegas	Entertainment
Maizie's Cafe & Bistro	Phoenix	Restaurants	Sweet Tomatoes	Las Vegas	Restaurants	Conservatory & Botanical Garden	Las Vegas	Entertainment
Cherryblossom Noodle Cafe	Phoenix	Restaurants	Strip N Dip Chicken Strips	Las Vegas	Restaurants	Planet Hollywood Las Vegas Resort & Casino	Las Vegas	Casinos & Hotels
Pizza a Metro	Phoenix	Restaurants	Patisserie Manon	Las Vegas	Restaurants	Flamingo Las Vegas Hotel & Casino	Las Vegas	Casinos & Hotels
SideBar	Phoenix	Bars	Island Flavor	Las Vegas	Restaurants	New York - New York	Las Vegas	Casinos & Hotels
Churn	Phoenix	Coffee & Tea	Japanese Curry Zen	Las Vegas	Restaurants	Miracle Mile Shops	Las Vegas	Shopping
Carly's Bistro	Phoenix	Restaurants	Slidin' Thru	Las Vegas	Restaurants	Palms Casino Resort	Las Vegas	Casinos & Hotels
Federal Pizza	Phoenix	Restaurants	Art of Flavors	Las Vegas	Food	The Forum Shops	Las Vegas	Shopping

that the steps of applying CF methods in each subgroup and considering the affiliation strengths are both essential when aggregating the results.

On the Number of Subgroups

In this subsection, we study how the choice of the number of subgroups K impacts the recommendation result. Due to space limits, we only show the results on the Last.fm and Netflix data. The results on MovieLens and Yelp data show similar trends. The results are shown in Figure 5.4.

We observe that for the Netflix data, setting K to 20 is a good choice where all the four base methods get best results. The Netflix data is not so sensitive to the number of subgroups. For the Last.fm data, 10 and 20 are the best choices. Increasing the number decreases the performance.

We note that increasing K would not necessarily increase the recommendation performance. The user-item matrix is very sparse, and increasing the number of subgroups would keep splitting large subgroups into several more focused small overlapping groups. Taking an extreme case for example, we might learn a lot of small subgroups that all of the users and items have links with each other. This case produces perfect co-clusters, but we do not have *new* items to recommend to users, which would hurt the recommendation performance.

5.5.4 Analysis of the Co-Clustering Results

In this section, we conduct a qualitative analysis of the co-clustering results to provide anecdotal evidence that the discovered subgroups are meaningful.

The goal of our method is to find focused subgroups, where the items in the same subgroup share some common latent properties that attract a group of like-minded users. Users' interests are usually not explicit, and we only have limited information about users in the data sets due to privacy issues. It is easier and more straightforward to study the item properties because we have more detailed information about items.

Table 5.8 shows the items with largest affiliation strengths from 3 sample subgroups from 10 total subgroups. We also present their locations and categories to study what these items are. We can see that the items from each subgroup share similar geographical and semantic properties. The top items from the first subgroups are all businesses from the city *Phoenix*, and most of them are restaurants. We can regard this subgroup as "Restaurants in *Phoenix*". Similarly, the top items from the second subgroup are also restaurants, but they are all in the greater Las Vegas area (*Henderson* is a city adjacent to *Las Vegas*). The third subgroup contains hotels, casinos or attractions in *Las Vegas*. The users from this subgroup are more likely to be tourists of *Las Vegas*, while the users of the second subgroup may also be the *Las Vegas* locals.

5.6 Conclusion

In this chapter, we proposed the scalable co-clustering method CCCF to improve the performance of CF-based methods for Top-N recommendation. The intuition of our model is that users may have different preferences over different subsets of items, where these subsets of items may overlap, and we explore subgroups of users who share interests over similar subsets of items through overlapping clustering. Experimental results on four real world data sets demonstrate that CCCF can improve four representative CF base methods (POP, ITMECF, MF, WARP) by co-clustering, and it also outperforms other co-clustering based methods (MCoC and BBDF). Qualitative analysis on the data sets also shows that the discovered subgroups are semantically meaningful.

Chapter 6

Conclusion

6.1 Summary

Collaborative Filtering is the most popular method for recommender systems. User preference learning is the key ingredient of Collaborative Filtering methods to serve satisfactory recommendations. In this thesis, we proposed several new models that help us better understand user preferences in order to meet users' need. The contributions of this thesis can be summarized as follows:

- In Chapter 3, we proposed a probabilistic model FLAME, which combines the efforts from the Aspect-based Opinion Mining and Collaborative Filtering, to help us better understand the reasons behind a rating. FLAME is one of the earliest works for fine-grained user preference learning from reviews for recommender systems.
- In Chapter 4, we proposed a neural network model CDAE, which generalizes several popular models and has more flexible structure, thus making better recommendation results. CDAE is one of the earliest works that successfully apply and adopt neural network models for recommender systems.
- In Chapter 5, we proposed a scalable co-clustering model CCCF, which first discovers locally focused subgroups from the user-item matrix, and then performs collaborative filtering models within each subgroup. CCCF solves several limitations of previous co-cluster methods, and enjoys several benefits including scalability, flexibility, interpretability and extensibility.

6.2 Future Directions

The research of this thesis suggests many promising future directions. Here we list some of them.

Opinion Mining and Collaborative Filtering

In Chapter 3, we mainly focused on solving the aspect-based opinion mining problem for the items from a specific domain, such as hotels or restaurants that we used for experiments. One

interesting future work is whether we could adopt FLAME to the data sets where we have various categories of items with different aspects. Also, some websites like TripAdvisor provide the option of rating some pre-defined aspects. Although these aspect ratings are typically incomplete, they should be helpful to partially guide the learning of latent aspect ratings. It is worth exploring a semi-supervised extension of FLAME.

Deep Learning for Recommender Systems

Deep Learning has gained great success in other fields such as computer vision and nature language processing. However, few work has been done on transferring its success to recommender systems. CDAE is only a one-hidden-layer neural network. We have tried to extend it to a deep structure by stacking the auto-encoders, but did not get significant improvement. Several other works such as [60] use deep learning as an intermediate step to learn the feature representations of users and items. The power of deep learning is its end-to-end nature. Thus, end-to-end deep learning frameworks for recommender systems would be the next hot topic.

Co-Clustering for Recommender Systems

A future work for this direction is to combine the idea of CCCF with Locally Collaborative Ranking (LCR) [25], where when applying the CF methods to the subgroups, the group affiliation strengths could also considered. Our model is complementary to LCR in the sense that it can replace the randomized anchor points selection and define the distance function using the joint user-item affiliation strengths. Also, it would be interesting to incorporate user and item features into the CCCF model in order to further improve the clustering results and to make them more interpretable. Besides, the number of subgroups of CCCF is a pre-defined hyperparameter. How to automatically determine the best number of subgroups using a Bayesian nonparametric method is also worth exploring.

Cold-start Problem

For new users and items, we do not have enough historical behavior data to apply collaborative approaches to learn the representations of them. This is a common problem for every existing recommender system, and also poses challenges on the methods we proposed in this thesis.

Recently, Multi-armed bandit[1] methods have been widely used to solve cold-start problem for recommender systems. The multi-armed bandit problem models an agent that simultaneously attempts to acquire new knowledge (*exploration*) and optimize his or her decisions based on existing knowledge (*exploitation*). An interesting future direction is to apply Multi-armed bandit algorithms to the problems we studied in this thesis.

[1] `https://en.wikipedia.org/wiki/Multi-armed_bandit`

Bibliography

[1] Deepak Agarwal and Bee-Chung Chen. Regression-based latent factor models. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 19–28, 2009.

[2] Edoardo M Airoldi, David M Blei, Stephen E Fienberg, and Eric P Xing. Mixed membership stochastic blockmodels. In *Advances in Neural Information Processing Systems*, pages 33–40, 2009.

[3] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems*, pages 153–160, 2006.

[4] Alex Beutel, Amr Ahmed, and Alexander J Smola. Accams: Additive co-clustering to approximate matrices succinctly. In *Proceedings of the 24th International Conference on World Wide Web*, pages 119–129, 2015.

[5] Alex Beutel, Kenton Murray, Christos Faloutsos, and Alexander J Smola. Cobafi: collaborative bayesian filtering. In *Proceedings of the 23rd international conference on World wide web*, pages 97–108, 2014.

[6] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003.

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

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

[9] Minmin Chen, Kilian Q Weinberger, Fei Sha, and Yoshua Bengio. Marginalized denoising auto-encoders for nonlinear representations. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1476–1484, 2014.

[10] Tianqi Chen, Hang Li, Qiang Yang, and Yong Yu. General functional matrix factorization using gradient boosting. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 436–444, 2013.

[11] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

[12] Jacob Eisenstein, Amr Ahmed, and Eric P Xing. Sparse additive generative models of text. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1041–1048, 2011.

[13] James Foulds, Levi Boyles, Christopher DuBois, Padhraic Smyth, and Max Welling. Stochastic collapsed variational bayesian inference for latent dirichlet allocation. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 446–454. ACM, 2013.

[14] Thomas George and Srujana Merugu. A scalable collaborative filtering framework based on co-clustering. In *ICDM'05*. IEEE, 2005.

[15] Xavier Glorot, Yoshua Bengio, and Yann N Dauphin. Large-scale learning of embeddings with reconstruction sampling. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 945–952, 201.

[16] Matthew Hoffman, Francis R Bach, and David M Blei. Online learning for latent dirichlet allocation. In *advances in neural information processing systems*, pages 856–864, 2010.

[17] Liangjie Hong, Aziz S Doumith, and Brian D Davison. Co-factorization machines: modeling user interests and predicting individual decisions in twitter. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 557–566. ACM, 2013.

[18] Minqing Hu and Bing Liu. Mining and summarizing customer reviews. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '04, pages 168–177, New York, NY, USA, 2004. ACM.

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

[20] Yohan Jo and Alice H Oh. Aspect and sentiment unification model for online review analysis. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 815–824. ACM, 2011.

[21] 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.

[22] 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.

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

[24] Stanislas Lauly, Hugo Larochelle, Mitesh Khapra, Balaraman Ravindran, Vikas C Raykar, and Amrita Saha. An autoencoder approach to learning bilingual word representations. In *Advances in Neural Information Processing Systems*, pages 1853–1861, 2014.

[25] 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.

[26] Joonseok Lee, Seungyeon Kim, Guy Lebanon, and Yoram Singer. Local low-rank matrix approximation. In *Proceedings of The 30th International Conference on Machine Learning*, pages 82–90, 2013.

[27] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2014.

[28] Fangtao Li, Chao Han, Minlie Huang, Xiaoyan Zhu, Ying-Ju Xia, Shu Zhang, and Hao Yu. Structure-aware review mining and summarization. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 653–661. Association for Computational Linguistics, 2010.

[29] Bing Liu. Opinion mining and sentiment analysis. In *Web Data Mining*, pages 459–526. Springer, 2011.

[30] Bing Liu, Minqing Hu, and Junsheng Cheng. Opinion observer: analyzing and comparing opinions on the web. In *Proceedings of the 14th international conference on World Wide Web*, WWW '05, pages 342–351, New York, NY, USA, 2005. ACM.

[31] Lester W Mackey, Michael I Jordan, and Ameet Talwalkar. Divide-and-conquer matrix factorization. In *Advances in Neural Information Processing Systems*, pages 1134–1142, 2011.

[32] Julian McAuley and Jure Leskovec. Hidden factors and hidden topics: understanding rating dimensions with review text. In *Recsys*, 2013.

[33] Julian McAuley, Jure Leskovec, and Dan Jurafsky. Learning attitudes and attributes from multi-aspect reviews. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 1020–1025. IEEE, 2012.

[34] Samaneh Moghaddam and Martin Ester. Ilda: interdependent lda model for learning latent aspects and their ratings from online product reviews. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, SIGIR '11, pages 665–674, New York, NY, USA, 2011. ACM.

[35] Samaneh Moghaddam and Martin Ester. On the design of lda models for aspect-based opinion mining. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, CIKM '12, pages 803–812, New York, NY, USA, 2012. ACM.

[36] Samaneh Moghaddam and Martin Ester. The flda model for aspect-based opinion mining: addressing the cold start problem. In *Proceedings of the 22nd international conference on World Wide Web*, WWW '13, pages 909–918, 2013.

[37] Xia Ning and George Karypis. SLIM: Sparse linear methods for top-N recommender systems. In *Proceedings of the 11th IEEE International Conference on Data Mining*, pages 497–506, 2011.

[38] Jorge Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of computation*, 35(151):773–782, 1980.

[39] Mark O'Connor and Jon Herlocker. Clustering items for collaborative filtering. In *Proceedings of the ACM SIGIR workshop on recommender systems*, volume 128, 1999.

[40] Rong Pan, Yunhong Zhou, Bin Cao, Nathan Nan Liu, Rajan Lukose, Martin Scholz, and Qiang Yang. One-class collaborative filtering. In *Proceedings of the Eighth IEEE International Conference on Data Mining*, pages 502–511, 2008.

[41] Victor Y. Pan and Zhao Q. Chen. The complexity of the matrix eigenproblem. In *Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing*, STOC '99, pages 507–516. ACM, 1999.

[42] Christian Posse. Key lessons learned building recommender systems for large-scale social networks. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 587–587, New York, NY, USA, 2012. ACM.

[43] Steffen Rendle. Factorization machines with libfm. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 3(3):57, 2012.

[44] 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 (UAI)*, pages 452–461, 2009.

[45] 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.

[46] Francesco Ricci, Lior Rokach, and Bracha Shapira. *Introduction to recommender systems handbook*. Springer, 2011.

[47] Gerd Ronning. Maximum likelihood estimation of dirichlet distributions. *Journal of statistical computation and simulation*, 32(4):215–221, 1989.

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

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

[50] 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.

[51] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th International Conference on World Wide Web Companion*, WWW'15 Companion, pages 111–112, 2015.

[52] Hanhuai Shan and Arindam Banerjee. Bayesian co-clustering. In *ICDM'08*, pages 530–539. IEEE, 2008.

[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] Yue Shi, Martha Larson, and Alan Hanjalic. List-wise learning to rank with matrix factorization for collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 269–272. ACM, 2010.

[55] Harald Steck. Training and testing of recommender systems on data missing not at random. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 713–722. ACM, 2010.

[56] Ivan Titov and Ryan McDonald. Modeling online reviews with multi-grain topic models. In *Proceedings of the 17th international conference on World Wide Web*, pages 111–120. ACM, 2008.

[57] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[58] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *The Journal of Machine Learning Research*, 11:3371–3408, 2010.

[59] Chong Wang and David M. Blei. Collaborative topic modeling for recommending scientific articles. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '11, pages 448–456. ACM, 2011.

[60] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. Collaborative deep learning for recommender systems. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 1235–1244. ACM, 2015.

[61] Hongning Wang, Yue Lu, and Chengxiang Zhai. Latent aspect rating analysis on review text data: a rating regression approach. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 783–792. ACM, 2010.

[62] Hongning Wang, Yue Lu, and ChengXiang Zhai. Latent aspect rating analysis without aspect keyword supervision. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 618–626. ACM, 2011.

[63] Pu Wang, Carlotta Domeniconi, and Kathryn Blackmond Laskey. Latent dirichlet bayesian co-clustering. In *Machine Learning and Knowledge Discovery in Databases*, pages 522–537. Springer, 2009.

[64] Pu Wang, Kathryn B Laskey, Carlotta Domeniconi, and Michael I Jordan. Nonparametric bayesian co-clustering ensembles. In *SDM*. SIAM, 2011.

[65] Markus Weimer, Alexandros Karatzoglou, Quoc V Le, and Alex J Smola. COFI RANK-maximum margin matrix factorization for collaborative ranking. In *Advances in Neural Information Processing Systems*, pages 1593–1600, 2007.

[66] Jason Weston, Samy Bengio, and Nicolas Usunier. WSABIE: scaling up to large vocabulary image annotation. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*, pages 2764–2770, 2011.

[67] Jason Weston, Chong Wang, Ron Weiss, and Adam Berenzweig. Latent collaborative retrieval. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 9–16, 2012.

[68] 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.

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

[71] Jaewon Yang and Jure Leskovec. Community-affiliation graph model for overlapping network community detection. In *ICDM'12*, pages 1170–1175. IEEE, 2012.

[72] Jaewon Yang and Jure Leskovec. Overlapping community detection at scale: A nonnegative matrix factorization approach. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, pages 587–596. ACM, 2013.

[73] Jaewon Yang, Julian McAuley, and Jure Leskovec. Community detection in networks with node attributes. In *ICDM'13*, pages 1151–1156. IEEE, 2013.

[74] Shuang-Hong Yang, Bo Long, Alexander J Smola, Hongyuan Zha, and Zhaohui Zheng. Collaborative competitive filtering: learning recommender using context of user choice. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 295–304. ACM, 2011.

[75] Liang Zhang, Deepak Agarwal, and Bee-Chung Chen. Generalizing matrix factorization through flexible regression priors. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, RecSys '11, pages 13–20. ACM, 2011.

[76] Yongfeng Zhang, Guokun Lai, Min Zhang, Yi Zhang, Yiqun Liu, and Shaoping Ma. Explicit factor models for explainable recommendation based on phrase-level sentiment analysis. In *Proceedings of the 37th international ACM SIGIR conference on Research and development in Information Retrieval*, SIGIR '14, Gold Coast, Australia, 2014. ACM.

[77] Yongfeng Zhang, Min Zhang, Yiqun Liu, and Shaoping Ma. Improve collaborative filtering through bordered block diagonal form matrices. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pages 313–322. ACM, 2013.

[78] Yongfeng Zhang, Min Zhang, Yiqun Liu, Shaoping Ma, and Shi Feng. Localized matrix factorization for recommendation based on matrix block diagonal forms. In *Proceedings of the 22Nd International Conference on World Wide Web*, pages 1511–1520. ACM, 2013.

[79] Yongfeng Zhang, Min Zhang, Yi Zhang, Yiqun Liu, and Shaoping Ma. Understanding the sparsity: Augmented matrix factorization with sampled constraints on unobservables. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 1189–1198. ACM, 2014.

[80] Wayne Xin Zhao, Jing Jiang, Hongfei Yan, and Xiaoming Li. Jointly modeling aspects and opinions with a maxent-lda hybrid. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pages 56–65, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.