From Micro to Macro: Understanding User Preferences from Different Perspectives for Personalized Recommendation

Yao Wu

School of Computing Science
Simon Fraser University
wuyaow@sfu.ca

March 3, 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 report, we first present an overview of several state-of-the-art Collaborative Filtering models. Afterwards, we discuss three perspectives to rethink the recommendation problem and some related work that attempts to improve recommender systems from these perspectives. Finally, we point out some potential opportunities for the future work.

Contents

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 report. Recommender systems have contributed a large proportion of the traffic and revenue to a large number of on-line services like LinkedIn, Amazon, Pandora, etc.

Recommender systems use a number of different methodologies. We can generally classify these methods into two categories [9].

- **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, then recommending some more videos with the same genre to him is a straightforward solution.
- **Collaborative Filtering** methods are based on analyzing a large amount of user behaviors and predicting what users will 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 approach relies on the high quality contents that could accurately describe users and items in order to make good recommendations. Collaborative Filtering approach does not rely on this and therefore it is capable of recommending complex items like movies and musics without requiring an understanding of the items themselves. In this report, we focus our discussion on the Collaborative Filtering approaches as they have been shown to be able to produce more accurate recommendations and have been widely adopted in most successful recommender systems like Amazon, Netflix, YouTube and Pandora.

This report is organized as follows. We first present some background knowledge in Section 2, and then give an overview of several state-of-the-art Collaborative Filtering models used for recommender systems in Section 3. Afterwards, in Section 4, we discuss three perspectives to rethink the recommendation problem and some related work that attempts to improve recommender systems from these perspectives. Section 5 concludes this report with a short summarization of our work and discussion of future work.s

2. Background

2.1. Preliminaries

In a recommender system application, there are two main classes of entities – *users* and *items*. Users have preferences for certain items, and these preferences must be teased out of the data. The input data is usually represented as a utility matrix, giving for each user-item pair, a value that represents what is known about the degree of preference of that user for that item.

Based on the types of the values in the utility matrix, the input data can be classified into two groups:

- **Explicit feedback**. Users explicitly express their feedback on items via ratings or reviews. Explicit feedback can also be from an ordered set, e.g., integers in $[1,5]$, representing the number of stars that the user gave as a rating for that item, or binary values in $\{\pm 1\}$, representing whether the user likes or dislikes that item.
- **Implicit feedback**. In real-world scenarios most feedback is not explicit but implicit. Implicit feedback is tracked automatically, like monitoring clicks, view times, purchases,etc. Thus it is much easier to collect, because the user has not to express his taste explicitly. In fact implicit feedback is already available in almost any information system – e.g. web servers record any page access in log files

Table 1: A sample utility matrix, representing users' ratings of movies on a 1–5 scale, with 5 the highest rating.

	Avatar	**The Godfather**	**The Shawshank Redemption**	**Transformers**	**Star Wars**
Alice		5	4		
Bob	3			4	5
Chris		5	4		
David	4		1	5	5

Each user only knows or has consumed a small subset of all the available items. Therefore, the utility matrix is 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 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.

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

In the rest of the report, 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 $\boldsymbol{X}_{ij}$.

2.2. Tasks and Evaluation Measures

Before we start discussing the methodologies, it is essential to make our target clear, i.e., what goal the recommender systems should achieve. In this subsection, we discuss three popular evaluation protocols in literature, and argue that top-N recommendation is the one closest to real world scenarios.

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 work 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):

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

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

where $\mathcal{T}$ 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., [8, 32]) 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, NDCG, MAP, etc.

Top-N Recommendation Although Rating Prediction and Item Ranking have been widely used in previous work, they are not suitable for the real world applications. The most critical issue is that they only train and test the models on the observed set in the utility matrix. However, users have selection bias, i.e., users prefer to rate/purchase/view the items they like rather than the items they dislike. This phenomenon is called *Ratings Missing not at Random* [26]. Models trained only on the observed set are biased, and we can not directly apply them to the missing set because the data distribution is different.

In recent years, top-N recommendation has become more standard approach to evaluate the performances 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.

3. Overview of Model-based Collaborative Filtering Methods

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.

3.1. Recommender Models

In general, recommender models are defined as

$$\hat{y}_{ui} = \mathcal{F}_{\boldsymbol{\theta}}(u, i), \tag{3.1}$$

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 $\mathcal{F}_{\boldsymbol{\theta}}$ 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 $\boldsymbol{\nu}_u$ and $\boldsymbol{\nu}_i$, representing the user and the item, respectively [7, 22]. [1]

$$\hat{y}_{ui} = \boldsymbol{\nu}_u^\top \boldsymbol{\nu}_i \tag{3.2}$$

Similarity Model (SM). The Similarity model [6] 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 [15].

$$\hat{y}_{ui} = \sum_{j \in \mathcal{O}_u \setminus \{i\}} y_{uj} \cdot \boldsymbol{S}_{ji} \tag{3.3}$$

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 [5, 6].

$$\hat{y}_{ui} = \left(\sum_{j \in \mathcal{O}_u \setminus \{i\}} y_{uj} \cdot \boldsymbol{p}_j \right)^\top \boldsymbol{q}_i \tag{3.4}$$

LFSM (LFM+FSM). The above models can also be combined. For example, combining LFM and FSM results in the model SVD++ [6], which proved to be one of the best single models for the Netflix Prize.

$$\hat{y}_{ui} = \left(\sum_{j \in \mathcal{O}_u \setminus \{i\}} y_{uj} \cdot \boldsymbol{p}_j + \boldsymbol{p}_u \right)^\top \boldsymbol{q}_i \tag{3.5}$$

3.1.1. Feature-aware Models Above models 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 [2] 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 chose 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.

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

[2] 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 all of them.

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

$$\hat{y}_{ui} = \boldsymbol{x}_u^\top \boldsymbol{W} \boldsymbol{y}_i. \tag{3.6}$$

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 overfitting 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.

$$\hat{y}_{ui} = \boldsymbol{x}_u^\top \boldsymbol{U}^\top \boldsymbol{V} \boldsymbol{y}_i. \tag{3.7}$$

Some advanced feature transformations [2] and non-linear kernels [34] can be applied to enhance the power of the model. We can also use one-hot encoding [3] representation of user/items ids as input feature, and then the Latent Factor Model can be thought as a special case. Factorization Machine [18] can also be formulated as a special case of Bilinear Model.

3.1.2. 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 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:

$$\hat{y}_{uic} = \boldsymbol{v}_u \bullet \boldsymbol{v}_i \bullet \boldsymbol{v}_c \tag{3.8}$$

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

3.2. Objective Functions for Recommenders

Objective functions for training recommender models can be roughly grouped into two groups: point-wise and pair-wise[4]. 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 [17, 26].

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.

$$\sum_{(u,i)\in\mathcal{O}'} \ell_{point}(y_{ui}, \hat{y}_{ui}) + \lambda\Omega(\boldsymbol{\theta}). \tag{3.9}$$

Here $\mathcal{O}'$ denotes an augmented dataset that includes both observed and unobserved user-item pairs. As we discussed in Section 2, 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 $\mathcal{O}$ leads to a biased model towards to the observed sets [17, 26]. A common

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

[4] Some models use list-wise objective functions [25, 32], but they are not as widely adopted as point-wise and pair-wise objectives.

Table 2: Overview of related model-based recommenders.

Name	Model	Objective Function
MF [7] / PMF [22]	LFM	ℓ^{SL}_{point}
SVD++ [6]	LFSM	ℓ^{SL}_{point}
MMMF [20]	LFM	ℓ^{HL}_{point}
WSABIE [33]	LFM	ℓ^{HL}_{pair}
BPR-MF [19]	LFM	ℓ^{LL}_{pair}
SLIM [15]	SM	ℓ^{SL}_{point}
FISM [5]	FSM	ℓ^{SL}_{point}, ℓ^{SL}_{pair}
WRMF [4]	LFM	*weighted* ℓ^{SL}_{point}
COFI [32]	LFM	NDCG loss
CLiMF [24]	LFM	MRR loss

solution is to augment $\mathcal{O}$ with a subset of *negative* user-item pairs [5] from the unobserved set, and train the model on the augmented set $\mathcal{O}'$ 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 [17]. $\mathcal{O}'$ can also include duplicate samples to simulate feedback with different confidence weights (*e.g.*, [4]).

Pair-wise objective function.

$$\sum_{(u,i,j)\in\mathcal{P}} \ell_{pair}(y_{uij}, \hat{y}_{uij}) + \lambda\Omega(\boldsymbol{\theta}), \tag{3.10}$$

where $y_{uij} = y_{ui} - y_{uj}$, $\hat{y}_{uij} = \hat{y}_{ui} - \hat{y}_{uj}$, and $\mathcal{P}$ is a set of triplets sampled from $\mathcal{O}'$, 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[6].

- 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.*,[8]).

Table 2 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 [33, 34], CLiMF [24]). 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.

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

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

4. Understanding User Preference from Different Perspectives

As we can see above, learning user preferences is the core question of Collaborative Filtering. Most of the models discussed above are based on the latent factor models – a low-rank matrix approximation approach that assumes that there are several general patterns of user preferences (e.g., ranks in the utility matrix), and a user's preferences on all items is a linear combination of these patterns. In this section, we would like to ask several questions that above methods might fail to give satisfactory answers and then discuss several related work that attempts to solve these problems.

Let us go back to the utility matrix in Table 1. We would ask several questions from three perspectives:

- **Element-wise Perspective**. If we zoom into each element of the matrix, is a numeric number good enough to describe user's preference?
- **Row-wise Perspective**. If we look through a user's view (*i.e.*, a row in the matrix), what is personalized recommendation in principle?
- **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?

We will discuss these questions in detail and some related work for each of them.

4.1. Element-wise Perspective

Most of the models we discussed in Section 3 take the numeric ratings as inputs, assuming that users with the same ratings share the same taste. However, two users might give 4 starts to a same item with different reasons – one might like a restaurant's food while the other like its service. Simply treating the feedback as a number is not good enough to accurately describe users' preferences. Thus, here comes an important research question: *can we understand the reasons behind a rating?*

Fortunately, for a number of on-line services, besides these numeric ratings, users are usually asked to provide textual reviews on the items that they have rated/consumed. Users might explain why then give the ratings by sharing their opinions on different aspects of the items. Reviews have become an pretty important source for users to check before making decisions. For example, TripAdvisor has plenty of user reviews on most hotels around the world, and users would like to check out the comments made by other people before booking a hotel. A challenge here is that aspect-based sentiment scores are not *explicitly* specified by users, but *implicitly* expressed in the review texts. Therefore classic Collaborative Filtering models can not be directly applied here, and it calls for a method that is able to learn the hidden aspects and sentiments from the reviews.

Aspect-based opinion mining [11] has attracted a lot of attention recently because of the huge amount of valuable information hidden in user reviews. 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 4.1 shows a sample review from Amazon. 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 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 [12], uncovering the latent aspects of the review sentences [14], inferring the latent aspect ratings and aspect weights of each review document [28, 29], aspect-based review summarization for products [10, 12], 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

Figure 4.1: A sample review on Amazon

Aspect Rating Analysis Model (LARAM) proposed in [28, 29] 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, 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. 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.

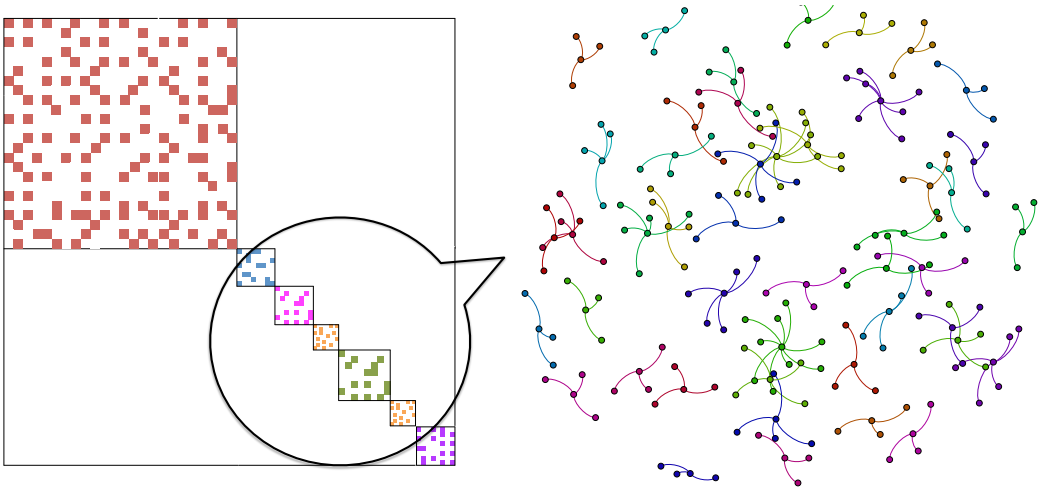
HTF [13] is the first work that tries to understand user ratings in recommender systems by utilizing review texts. HFT considers latent rating factors of an item as the properties that the item possesses, and assumes that if a product exhibits a certain property (higher latent rating factor value), this will correspond to a particular topic being discussed frequently (higher probability in topic distribution) [13]. HTF first aggregates all the reviews of an item into a single document, and uses a similar method as Collaborative Topic Regression model [27] 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) [40] 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.

4.2. Row-wise Perspective

In this subsection, we look through the matrix from a row-wise perspective, *i.e.*, for a user. As mentioned in Section 2.2, we are more interested in the top-N recommendation task, and here we focus our discussion on implicit feedback. Interesting thing about implicit feedback systems is that only positive observations are available. The non-observed user-item pairs – *e.g.*, a user has not bought an item yet – are a mixture of real negative feedback (the user is not interested in buying the item) and missing values (the user might want to buy the item in the future). A common approach is that for each user, the items with positive implicit feedback are generally more relevant than the items in the missing set, so they should rank higher (have larger prediction scores) than others.

From a user's perspective, we are given a subset of items that he likes, and need to find the whole set of his favorite items. We have already discussed several state-of-the-art models for top-N recommendation in Section 3. So we do not go to details in this subsection and refer the readers to Section 3 for an overview



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

Figure 4.2: Structures of Yelp dataset [43]. In the left is the exampled structure of the rating matrix, and in the right is the real structure of the scattered blocks.

of related work.

4.3. Matrix-wise Perspective

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 [21]; 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 [7]. 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 [7] can be found in [43]. By permuting the rows and columns of the rating matrix, it can be rearranged into a BDF structure with 53 diagonal blocks, with a dominating block and 52 scattered blocks, which is shown in Figure 4.2. 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. We note that these subgroups might be overlapping with each other, namely, users and items can belong to multiple subgroups. This scenario does not only exist in the location-based services, but also can be observed from other general domain recommender systems. 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. Partitioning users and items into subgroups for CF has been studied by several previous works, where user clustering [39], item clustering [16] and user-item co-clustering [3] 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 [23, 30, 31] extend

[7] https://www.yelp.com/dataset_challenge

the Mixed Membership Stochastic Blockmodels (MMSB) [1] to allow mixed memberships. However, these methods optimize the accuracy of rating/link prediction instead of the practically important problem of top-N recommendation.

In [38], 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 [41] 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. And they propose a general Collaborative Filtering framework based on these structures to make rating predictions. They also proposed a unified model LFM [42] that learns ABBDF and Latent Factor Model simultaneously.

The framework used in [38, 41] 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.

5. Conclusion

In this report, we presented an overview of the model-based approaches for recommender systems, and proposed to rethink the recommendation problem from three perspectives. We discussed some related work that attempts to improve the recommender systems from these perspectives.

We have done some preliminary work on these three perspectives: FLAME [36] was proposed to understand fine-grained user preference by analyzing the latent aspects and sentiments hidden in the textual reviews. By thinking the recommendation as a reconstruction problem, we proposed CDAE [35] to produce better top-N recommendation for users. CCCF [37] studies the problem from the whole user item matrix point of view, and improves the recommender system by a divide-and-conquer approach.

Still, there are several challenging problems in this direction. Here we list some of them. 1) For the problem discussed in Section 4.1, how to accurately extract opinions from the texts is an unsolved problem. The techniques in this direction are far from perfect. 2) For the personalized recommendation techniques discussed in **??**, the ultimate goal is to learn a generic model that could model the emerging user choices over time. Using online learning techniques such as Multi-armed Bandits is worth exploring. 3) For the matrix-wise perspective, how to design a unified model that can simultaneously learn the latent submatrix structure and the local models is a challenging and interesting problem.

References

[1] E. M. Airoldi, D. M. Blei, S. E. Fienberg, and E. P. Xing. Mixed membership stochastic blockmodels. In *Advances in Neural Information Processing Systems*, pages 33–40, 2009.

[2] T. Chen, H. Li, Q. Yang, and Y. Yu. General functional matrix factorization using gradient boosting. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 436–444, 2013.

[3] T. George and S. Merugu. A scalable collaborative filtering framework based on co-clustering. In *ICDM'05*. IEEE, 2005.

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

[5] S. Kabbur, X. Ning, and G. 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.

[6] Y. 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.

[7] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[8] J. Lee, S. Bengio, S. Kim, G. Lebanon, and Y. Singer. Local collaborative ranking. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 85–96. ACM, 2014.

[9] J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of massive datasets*. Cambridge University Press, 2014.

[10] F. Li, C. Han, M. Huang, X. Zhu, Y.-J. Xia, S. Zhang, and H. 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.

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

[12] B. Liu, M. Hu, and J. 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.

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

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

[15] X. Ning and G. 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.

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

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

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

[19] S. Rendle, C. Freudenthaler, Z. Gantner, and L. 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.

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

[21] F. Ricci, L. Rokach, and B. Shapira. *Introduction to recommender systems handbook*. Springer, 2011.

[22] R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. In *Advances in neural information processing systems*, pages 1257–1264, 2007.

[23] H. Shan and A. Banerjee. Bayesian co-clustering. In *ICDM'08*, pages 530–539. IEEE, 2008.

[24] Y. Shi, A. Karatzoglou, L. Baltrunas, M. Larson, N. Oliver, and A. Hanjalic. Climf: learning to maximize reciprocal rank with collaborative less-is-more filtering. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 139–146. ACM, 2012.

[25] Y. Shi, M. Larson, and A. 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.

[26] H. 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.

[27] C. Wang and D. 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.

[28] H. Wang, Y. Lu, and C. 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.

[29] H. Wang, Y. Lu, and C. 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.

[30] P. Wang, C. Domeniconi, and K. B. Laskey. Latent dirichlet bayesian co-clustering. In *Machine Learning and Knowledge Discovery in Databases*, pages 522–537. Springer, 2009.

[31] P. Wang, K. B. Laskey, C. Domeniconi, and M. I. Jordan. Nonparametric bayesian co-clustering ensembles. In *SDM*. SIAM, 2011.

[32] M. Weimer, A. Karatzoglou, Q. V. Le, and A. J. Smola. COFI RANK-maximum margin matrix factorization for collaborative ranking. In *Advances in Neural Information Processing Systems*, pages 1593–1600, 2007.

[33] J. Weston, S. Bengio, and N. 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.

[34] J. Weston, C. Wang, R. Weiss, and A. Berenzweig. Latent collaborative retrieval. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 9–16, 2012.

[35] Y. Wu, C. DuBois, A. X. Zheng, and M. Ester. Collaborative denoising auto-encoders for top-n recommender systems. In *Proceedings of the 9th ACM International Conference on Web Search and Data Mining*, pages 153–162, 2016.

[36] Y. Wu and M. Ester. FLAME: A probabilistic model combining aspect-based opinion mining and collaborative filtering. In *Proceedings of the 8th ACM International Conference on Web Search and Data Mining*, pages 199–208, 2015.

[37] Y. Wu, X. Liu, M. Xie, M. Ester, and Q. Yang. CCCF: Improving collaborative filtering via scalable user-item co-clustering. In *Proceedings of the 9th ACM International Conference on Web Search and Data Mining*, pages 73–82, 2016.

[38] B. Xu, J. Bu, C. Chen, and D. 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.

[39] G.-R. Xue, C. Lin, Q. Yang, W. Xi, H.-J. Zeng, Y. Yu, and Z. 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.

[40] Y. Zhang, G. Lai, M. Zhang, Y. Zhang, Y. Liu, and S. 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.

[41] Y. Zhang, M. Zhang, Y. Liu, and S. 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.

[42] Y. Zhang, M. Zhang, Y. Liu, S. Ma, and S. 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.

[43] Y. Zhang, M. Zhang, Y. Zhang, Y. Liu, and S. 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.