Acknowledgements

First and foremost, we have to thank our thesis supervisors, Mr. Mai Le Tung and Mr Le Ngoc Thanh. Without their assistance and dedicated involvement in every step throughout the process, this thesis would have never been accomplished. We would like to thank you very much for your support and understanding over these past four years.

We would also like to show gratitude to our committee, including Prof. Vu DUONG, Assoc.Prof. Minh-Triet TRAN, Dr. Ba-Tien DINH, Quoc-Hoang VU and Dr. Thai-Son TRAN and all of our lecturers and teaching assistants in all four academic year at University of Science. Dr. Ba-Tien DINH was our first-year lecturer at HCM University of Science. His teaching style and enthusiasm for the topic made a strong impression on us and we have always carried positive memories of his classes with us.

Getting through our thesis required more than academic support, and we have many, many people to thank for supporting and, at times, having to tolerate us over the past year. We cannot begin to express our gratitude and appreciation for their friendship. Viet-Hoang NGUYEN QUANG, Phong-Quoc DANG, and Tri NGUYEN have been unwavering in their personal and professional support during the time we spent at the University. For many memorable evenings out and in, we must thank everyone above as well as Quang-Minh Trinh, Phuc-Quang LE and Quang-Thang TAT. We would also like to thank Hue-Dang Nguyen who helped us find data on the internet and lent his house

when we need a place to work.

Most importantly, none of this could have happened without our family. To our parents, our brothers and sisters – who always be besides to encourage us with their love and support. Every time we was ready to quit, they did not let us and we are forever grateful. This thesis stands as a testament to their unconditional love and encouragement.

Finally, we would like thank to each other - a strong passionate research team - for spending, sharing spirit, patience, idea, algorithm, methods, time, space, result, resource, sad and happiness through many difficulties of the thesis together.

Table of contents

List of Figures

List of Tables

Abstract

Recommender systems were developed to help users deal with information. These systems have become an important part of e-commerce. With the tremendous growth of e-commerce, scalability is one of the biggest challenges for recommender systems. To address the scalability problem, we propose *Incremental SVD++* method which enables online integration of new ratings. *Incremental SVD++* is based on the state-of-the-art recommendation algorithm SVD++. To our knowledge, our work is the first to extend SVD++ with incremental updates. Our method improves performance of classic SVD++, while maintaining the recommendation quality.

Chapter 1

Introduction

1.1 Recommender systems

The explosive growth of the Internet and electronic commerce gives users many benefits. The penetration of the Internet and technology into all areas of human activity has further boosted the volume of online information resources. Every single day in this information era, we create 2.5 quintillion bytes of data, mean about 2.5 billion gigabytes, according to the Big Data research of IBM[1]. The amount of information available on the Internet has become immense and is still increasing exponentially. On one hand, the abundance of online information may guarantee that users are able to find what they are looking for. On the other hand, this abundance also makes the useful information difficult to find. Users are facing an explosion of choices. The availability of choices, instead of producing a benefit, started to reduce the comfort of users. It is understood that while the choice is good, more choice is not always better. Every day, users have difficulty deciding which movie to watch, which book to read, which course to take and where to go for a travel. Therefore, information overload became a big challenge.

[1] http://www-01.ibm.com/software/data/bigdata/what-is-big-data.html

To deal with this challenge, Recommender systems were born. Since the appearance of the first research in the mid-1990s [2–4], recommender systems have become an important research area. These systems attempt to suggest information/items that a user may be interested in. The suggestions provided are aimed at helping users to cope with the information and choice over-abundance.

1.1.1 Applications of recommender systems

Recommender systems have become one of the most powerful and popular tools in electronic commerce. These systems have been successfully applied and played a pivot role in many online services. Internet Movie Database (IMDb)[2] and Movie-Lens[3] recommend movies to users. Amazon[4] and Taobao[5] have personalized recommendations based on a user's past purchase history, rating data and review data. Recommender systems also have been in many dating services such as LibimSeTi[6] and Perfect Match[7].

Figure 1.1 shows a screen shot of recommendations on Amazon.

[2]http://www.imdb.com/
[3]https://movielens.org/
[4]http://www.amazon.com/
[5]http://www.taobao.com
[6]http://www.libimseti.cz/
[7]http://www.perfectmatch.com/

FIGURE 1.1
RECOMMENDATIONS OF AMAZON

With these applications, E-commerce websites (e.g. Amazon, CDNow[8]) can build customer loyalty, increase profits and boost item-cross selling. In fact, it is reported that 35% product sales of Amazon come from recommendations, 38% more click-throughs on Google news are generated by recommendations, and over two thirds of the rented movies of Netflix[9] are recommended [5].

1.1.2 Types of recommender systems

According to how recommendations are made, recommender systems are usually classified into 3 types [6,7]: *content-based filtering*, *collaborative filtering*, and *hybrid recommender systems*. Fig. 2 shows the anatomy of different recommendation techniques.

[8]http://CDNow.com/
[9]http://www.netflix.com/

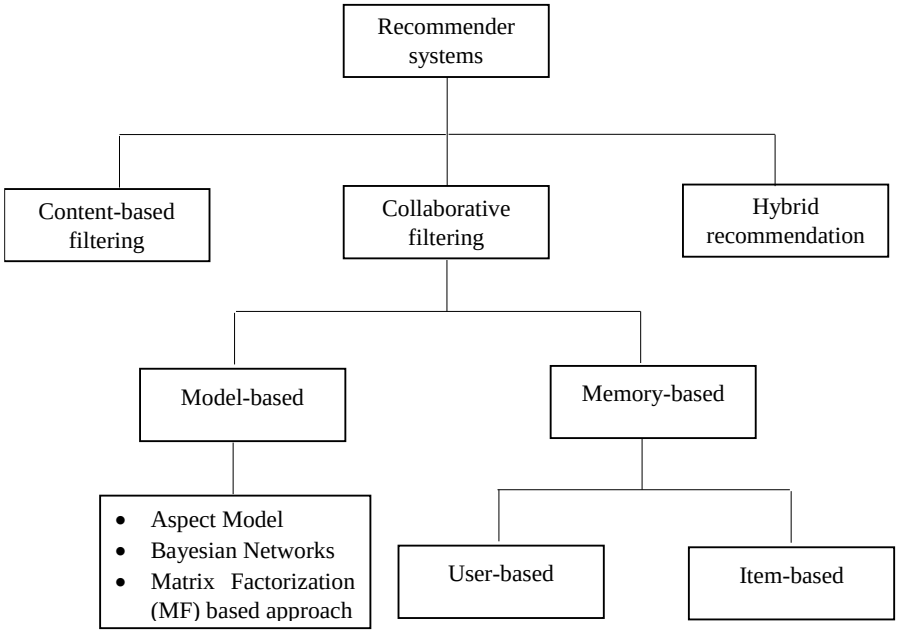

FIGURE 1.2
RECOMMENDATION TECHNIQUES

Content-based filtering recommends an item to a user by matching up the features of the item with the preferences of the user. The item features and the user's preferences are learnt by analyzing the profiles. For example, a movie profile might describe its genre, the participating actors, directors. User profiles could include demographic information such as gender, age, hobbies. The features of the content as well as the preferences of the user have to be learnt. This is usually very hard because the inherent diversity of contents of items. Content-based filtering has been successfully applied in areas such as news recommendation and music recommendation.

Collaborative filtering relies only on user past behaviours (previous transactions or product ratings). In addition, relying directly on user behavior allows uncovering complex and unexpected patterns that would be difficult or impossible to profile using known data attributes [8]. There are two main approaches of collaborative filtering: memory-based approach and

model-based approach. Memory-based approaches are focused on computing the relationships between users or items. Model-based approaches process observed ratings to build a predefined compact model in the training phase which is then used to make predictions.

Hybrid recommender combines multiple recommendation techniques together to eliminate the limitations of pure approaches [9].

1.2 Problem statement

Despite significant growth in the past decades, recommender systems still face many challenging problems. One of the biggest problems is the trade-off between accurate estimation prediction and the time required to calculate them. To address this problem, we need to consider two following goals simultaneously.

The first goal is that how to improve the quality of the recommendations for the customers. The accuracy of recommendations is the first criteria to evaluate a recommender system. For example, a recommender system recommends a list of products for a user, Joe. But Joe finds out that he does not like the suggested products, so Joe will be unlikely to use that recommender system again. He does not care about how fast the recommendations are produced or how diversity the recommendations are.

The Second goal is how to improve the scalability of the recommender systems. The tremendous growth of customers and products in recent years poses two key challenges for recommender systems: large-scale data and not being scalable to the demands of real world applications. For example, a recommender system recommends a list of products for a user, Joe. He likes all the suggested products, but he must wait thirty minutes to get the list of recommendations. Joe will be unlikely to use that recommender system again.

These are conflicts goals which require trade-offs. If an algorithm spends

less time for estimating prediction, it will be more scalable but produce worse quality.

Many researchers suggest that Singular Value Decomposition-based (SVD-based) methods may be a solution to improve the quality of recommender systems. SVD-based methods produced results that were better than a traditional CF algorithm most of the time when applied to the MovieLens data set [10].

A new approach of SVD was presented by Simon Funk in the context of the Netflix Prize [11]. This method used an approximate way to compute the low-rank approximation of the matrix by minimizing the squared error loss. Since then, many other SVD-based models have been created. One of them is SVD++ model, which is desribed by Yehuda Koren, the winner of Netflix Prize. SVD++ model relies on both explicit feedback and implicit feedback [8]. Implicit feedback is an especially valuable information source for users who do not provide much explicit feedback. It indirectly reflect opinion through observing user behavior. Hence, SVD++ uses implicit feedback as a secondary source of information. This model improves prediction accuracy to some extent.

Despite the high accuracy of recommendations, the matrix factorization step of SVD-based approaches is computationally very expensive because it takes a lot of memory and time to factorize a matrix. This limitations make SVD-based approach less suitable for large scale deployment in e-commerce system.

In order to deal with this, many e-commerce systems prefer to compute the model offline and feed the database with updated information periodically [12]. These systems provide recommendations to users quickly, based on pre-computed models. However, without considering the data submitted between two offline computations, these recommendations are not produced with the highest possible degree of confidence.

Let's look at a scenario:

A young man, Harry loved the classical music. Every day, he listened

to music on ABC online music website. this website recommended to him many classical songs. One day, he heard a rock songs and very loved it. He wanted to listen more rock music. He opened ABC website. Of course, it still recommended him classical songs. So he searched it by himself. After listening many songs on ABC website, he reloaded the page to get new recommendations but it was full of classical songs. And in the day after, ABC website suggested him a list of classical songs. What happened to the recommendation of ABC website?

The problem described in the scenario is happened with many recommender systems in real-world applications.

1.3 Thesis contribution

In this thesis, we present studies that improve recommender systems by solving the above mentioned problems. We propose *Incremental SVD++* method which extends SVD++ with incremental updates. Our method suitable for the dynamic scenario faced by recommender systems today. New users and new items join the system constantly. Our method significant improves scalability of classic SVD++, while maintaining its recommendation quality. The pre-computed model is updated incrementally at the time of rating activity and recommended items are modified based on newest data. Experimental results have shown a significant increase in training speed without a loss in accuracy.

The rest of the paper is organized as follows.

Chapter 2 reviews the background knowledge of recommender systems.More specifically, we present content-based filtering, collaborative filtering and hybrid recommender. We focus more on collaborative filtering, which is the dominating method in recommender systems nowadays.

Chapter 3 introduces several matrix factorization techniques like SVD and SVD++. We also propose a method incremental SVD++.

Chapter 4 presents our experimental procedure and results of the proposed methods on the MovieLens dataset.

Chapter ?? is the short summarization of Incremantal SVD++ and the developments of a simple application of Incremantal SVD++.

Chapter 2

Background

2.1 Recommender systems

Recommender systems are a subclass of information filtering system that collects users' preferences over time and try to make predictions on which items that a user might like in the future. Put simply, the aim of recommender systems is predicting ratings for the items that a user has not seen before. Once we can estimate a user's ratings for all unrated items, we can recommend the items which received the highest predicted value.

Recommender systems can broadly be classified into two types: content-based filtering and collaborative filtering. Both techniques have their advantages and disadvantages. Hybrid approaches, combining collaborative filtering and content-based filtering could be more effective in some cases. They can eliminate the limitations of pure techniques.

Figure 2.1 shows the principle of theses two techniques.

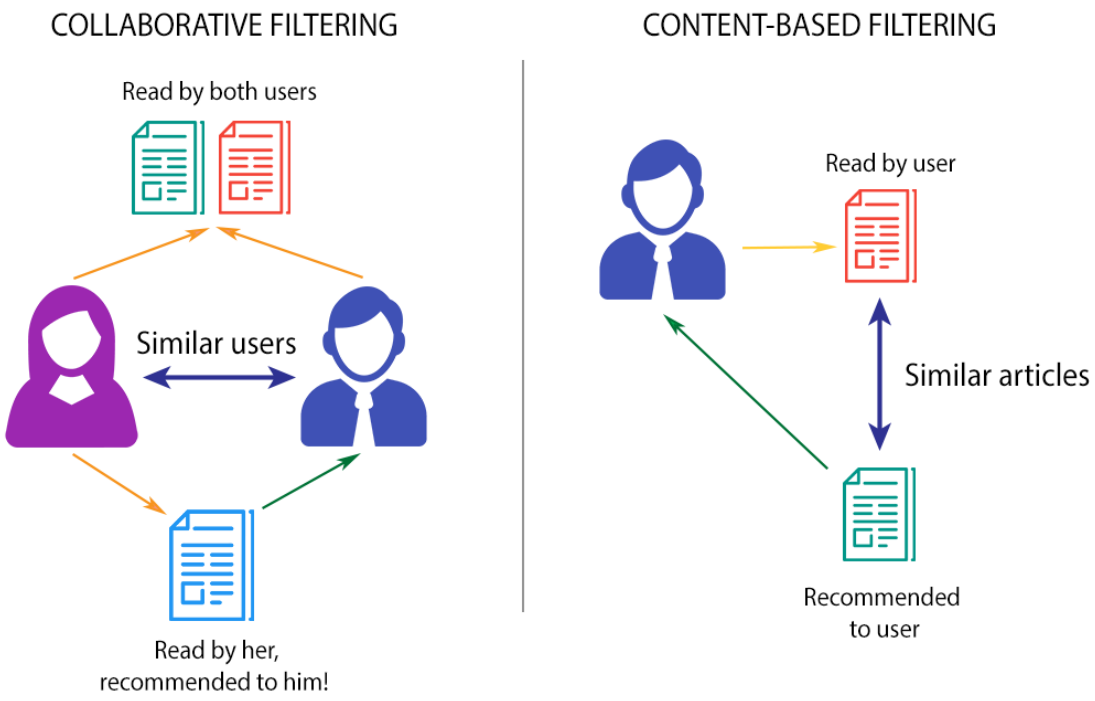

Figure 2.1
The principle behind collaborative and content-based filtering
Source: The Marketing Technologist.

2.2 Content-based filtering

In content-based filtering (CbF) approaches, a user is suggested items similar to those he liked or preferred in the past. Content-based filtering techniques make recommendations by comparing a user profile with the features of each item. Each user has a profile which describes his preferences.

For example, a user Joe has watched and rated the following movies:

Movies	Green Lantern	American Pie	Hangover	Saw	...
Rating	8	9	10	4	...

List of watched movies

The list of movies and their attribute-values:

	Action	Adventure	Children	Comedy	Crime	Drama
Green Lantern	1	1	0	1	0	0
American Pie	0	0	0	1	0	1
Hangover	0	1	0	1	0	1
Saw	1	0	0	0	1	0
Home Alone	1	1	1	1	0	1
...	...	...	...	...	...	...

Attribute-values of movies

The build of a user's profile does not depend on other users' behavior. The features of the items rated by a user are assumed to reflect the user's preferences. So content-based recommendation algorithms build a user's profile based on these features. Therefore, profile of Joe are built based on attribute-values of rated movies, i.e., Green Lantern, American Pie, Hangover,Saw, etc.

	Action	Adventure	Children	Comedy	Crime	Drama
Joe	0.6	0.5	-0.3	0.2	0.2	-0.4

Profile of Joe

Figure 2.2 shows the high level architecture of a content-based recommender. There are three major components, each component handles a step in the recommendation process [1].

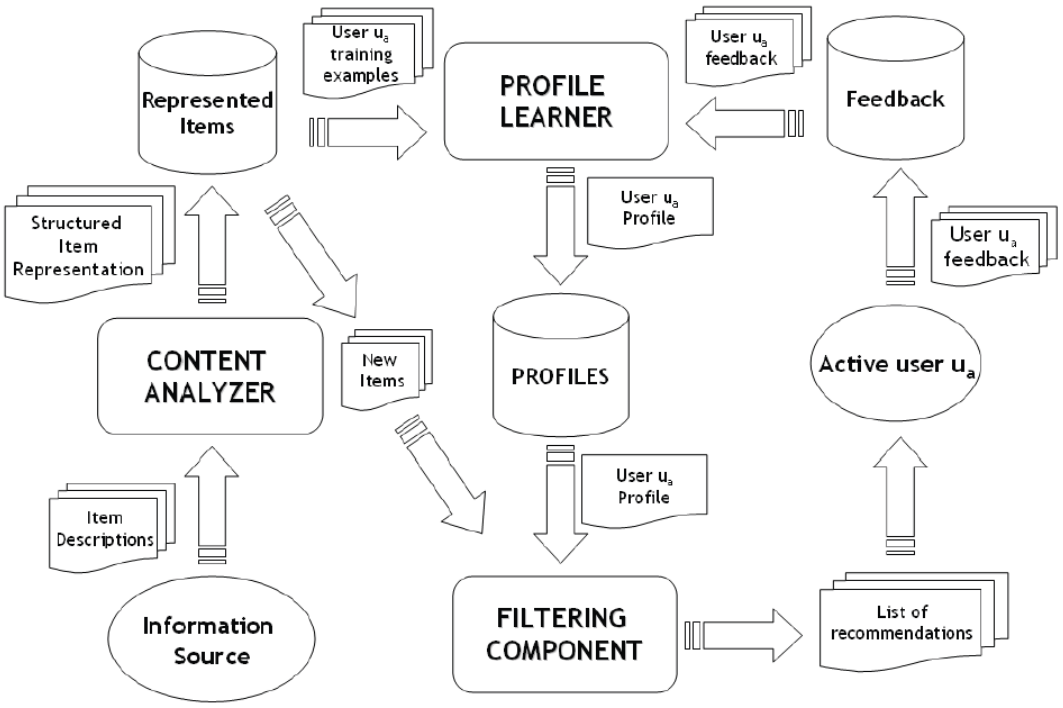

FIGURE 2.2
HIGH LEVEL ARCHITECTURE OF A CONTENT-BASED RECOMMENDER [1]

Content Analyzer performs pre-processing step that transforms the raw content of an item (e.g. documents, Web pages, news, product descriptions, etc.) into the form suitable for the next processing steps, like feature representation. These represented items are the input to the Profile Learner and Filtering Component

Profile Learner collects data related to the past preferences of a user and tries to construct the user profile. The user profile could be a prototypical item feature vector that represents the user's preferences.

Filtering Component computes relevance scores between items and given a user, based on the items' features and user profile. Then this component produces the list of recommendations based on these scores.

In this process, feedback of the active user is collected in order to construct and update the user's profile. Content analyzer step is very important. With

the suitable item representation, content-based filtering method produces high accuracy recommendations. Content analyzer usually involves domain dependent experts who select and devise the features suitable for the description of the items. For example, the features used for music recommendation are very different from the features used for news article recommendation. In addition, the content analyzer usually requires domain knowledge.

An advantage of content-based filtering is that explanations on how the recommender system works can be provided. Explanation on why a particular item is recommended can help user decide whether to take further actions.

Another advantage is that content-based filtering can recommend items with no previous ratings. With this advantage, content-based filtering has seen wide application in news recommendation which has the distinct feature of extremely large set of items. the items to recommend are breaking news that have no or little previous data. To recommend a news article content-based filtering analyzes the content of the article and match it with the preferences of a user, without the need to collect users' ratings on this article. On the other hand, content-based filtering is unable to handle the new users which has no or few preferences available. With no or few previous preferences, we cannot construct a reliable user profile, a key component in successful recommendation. So content-based filtering solves half of the cold-start problem.

The biggest disadvantage of content-based filtering is inability at discovering new unexpected items. The system does not recommend these items that are different from anything that the user has seen before. Sometimes this might become problem because the user might want to try something new.

2.3 Collaborative filtering

Collaborative filtering(CF) became one of most successful recommender technique since this approach was mentioned and described by Paul Resnick and Hal Varian in 1997 [13]. The underlying opinion of the CF approaches is that users who shared preferences in the past tend to share similar preferences in the future. With this opinion, we assume that there is a low-rank structure of the user item rating matrix and mathematical techniques can be used to fill this matrix and thus make recommendations.

Figure 2.3 shows the schematic diagram of the collaborative filtering process. There are two main tasks:

Predict task computes ratings that the target user gives to unrated items.

Recommend task produces the best ranked list of n items for the target user's need.

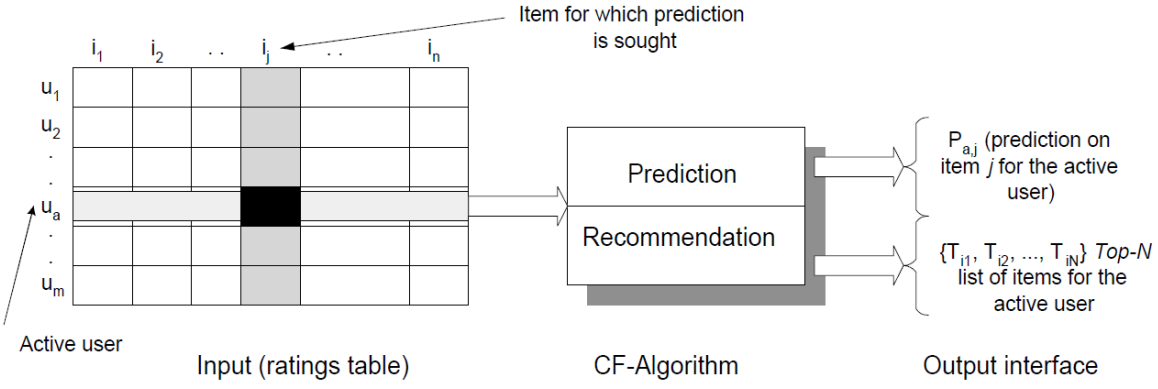

FIGURE 2.3
THE COLLABORATIVE FILTERING PROCESS.

Collaborative filtering models rely only on the past user behavior (i.e. previous transactions and ratings) without requiring the creation of explicit profiles.

	Green Lantern	American Pie	Hangover	Saw
u_1	5	3		1
u_2	4			1
u_3	1	1		5
u_4	1			4
u_5		1	5	4

TABLE 2.1
SAMPLE RATINGS MATRIX (ON A 5-STAR SCALE)

Typically, the data processed by a CF-based recommender system can be illustrated as in Table 2.1. This is a rating matrix of 5 users and 4 items. Each row represents a distinct user and each column represents one item, which is a movie here. Each cell contains the rating of the movie from 1 to 5 as provided by the users; 1 being the lowest rating and 5 being the highest one. Since all the users haven't seen or rated all of the movies, some cells remain vacant because of lack of information.

CF approaches take the ratings of the target user on the seen items and use them to predict the ratings for this user for the unseen items. Then, items are recommended in a descending order according to their predicted ratings.

Collaborative filtering techniques can be divided into two categories: memory-based and model-based. Although simple and easy to implement, memory-based methods have the limitations in prediction accuracy and time consuming to make a prediction.

Despite the success of CF techniques, these techniques suffer from various problems such as cold-start problem of new-user/new-item, data sparsity problem, and scalability.

In the following of this section,we first describe the notations that will be used throughout this thesis. We then review two approaches of collaborative filtering, including memory-based methods and model-based methods.

2.3.1 Notations

The notations used in this thesis are described as follows. Suppose that we are given a set of M users $U = \{u_1, u_2, ..., u_M\}$ and a set of N items $I = \{i_1, i_2, ..., i_M\}$. Users' rating on the items are arranged in a $N \times M$ matrix R where entry $r_{u,i}$ denote the rating that user u gives to item i and $r_{u,i} = 0$ if u have not rated i. The set of all items that user u have rated is denoted by I_u and the set of all users who have rated item i is denoted by U_i. Alternatively, we denote the set of all observed triplet $(u, i, r) \in K$ where u is the user id, i is the item id and r is the rating given by u to i. The whole set is denoted by K. Other notations used which are model specific will be discussed in the context.

Rating oriented collaborative filtering tries to produce prediction for the rating that is likely to be assigned to i by u, i.e. $r_{u,i}$. We denote the prediction by $\hat{r}_{u,i}$. Instead of giving prediction of ratings, ranking oriented collaborative filtering output a rank π of the items in decreasing order of preference.

2.3.2 Memory-based methods

The idea of memory-based methods (also called neighborhood-based) is that the rating predictions for a user directly depend on the ratings, on that item, of similar users (i.e. neighbors). Approaches are further divided into two techniques, user-based and item-based.

Similarity measures

One of the most important factors that affect the result of memory-based methods for collaborative filtering is how the similarity between users or items is measured. Note that the similarity measure is between two vectors. Let $s(u, v)$ denote the similarity measure between u and v. Several different similarity functions have been proposed and evaluated in the literature.

Pearson correlation

Pearson correlation coefficient is a similarity measure between two vectors. In essence, it measures the correlation between two vectors with respect to the product of their standard deviation. The correlation is computed by the following:

$$s\left(u,v\right)=\frac{\sum_{i\in I_u\cap I_v}\left(r_{u,i}-\bar{r}_u\right)\left(r_{v,i}-\bar{r}_v\right)}{\sqrt{\sum_{i\in I_u\cap I_v}\left(r_{u,i}-\bar{r}_u\right)^2}\sqrt{\sum_{i\in I_u\cap I_v}\left(r_{v,i}-\bar{r}_v\right)^2}} \tag{2.1}$$

Cosine similarity

It measures the cosine value of the angle between two vectors in high dimensional space. Its definition is given in equation 2.2

$$s\left(u,v\right)=\frac{r_u.r_v}{\left\|r_u\right\|.\left\|r_v\right\|}=\frac{\sum_i r_{u,i}r_{v,i}}{\sqrt{\sum_i {r^2}_{u,i}}\sqrt{\sum_i {r^2}_{vi}}} \tag{2.2}$$

User-based methods

In user-based methods, the prediction of an item for the target user is based on his/her similar users' ratings on it. The unknown ratings $\hat{r}_{u,i}$ is predicted using a set of other users' rating for item i. Let $s\left(v,u\right)$ denotes the similarity measure between 2 users u and v. N_u is the set of K neighbors of user u and U_i is all the users who have rated i. The prediction can be calculated as:

$$\hat{r}_{u,i}=\frac{\sum\limits_{v\in N_u\cap U_i}s\left(u,v\right).r_{v,i}}{\sum\limits_{v\in N_u\cap U_i}\left|s\left(u,v\right)\right|} \tag{2.3}$$

Equation 2.1 gives a simple method to calculate the prediction. However, there are several issues associated with this method. The first problem is that users may have very different rating behaviors. Some users will tend to give higher or lower ratings than others. A conservative user might never give more

than 3 while an optimism user would rate the items on the scale from 3 to 5. That is, the ratings given by a user could be biased. It is easy to see that this simple method could give rating prediction wildly deviate from the true rating. As a simple solution for this is to adjust a user's ratings with his mean rating [**?**]. This would give a slightly modified prediction:

$$\hat{r}_{u,i} = \bar{r}_u + \frac{\sum\limits_{v \in N_u \cap U_i} s\left(u,v\right).(r_{v,i} - \bar{r}_v)}{\sum\limits_{v \in N_u \cap U_i} |s\left(u,v\right)|} \tag{2.4}$$

In equation 2.4, $\bar{r}_u$ denotes the mean ratings given by u:

$$\bar{r}_u = \frac{\sum_{i \in I_u} r_{u,i}}{|I_u|} \tag{2.5}$$

Several different approaches have been proposed to alleviate the bias problem by normalizing the rating before calculating the similarity.

The time complexity of user-based approaches is $O(N^2xMxK)$ for the neighborhood model construction and $O(K)$ for the rating prediction. The space complexity is O(NxK) [5].

Item-based methods

Item-based methods are very similar with user-based methods. The prediction of an item for a user is based on the user's ratings on its similar items (neighbors). The assumption is that users would assign similar scores for similar items. Let $s\left(i,j\right)$ denotes the similarity measure between 2 item i and j. N_i is the set of K neighbors of item i and I_u is all the items that user u has rated. The prediction of item i for user u can be calculated as:

$$\hat{r}_{u,i} = \bar{r}_i + \frac{\sum\limits_{j \in N_i \cap I_u} s\left(i,j\right).\left(r_{u,j} - \bar{r}_j\right)}{\sum\limits_{j \in N_i \cap I_u} \left|s\left(i,j\right)\right|} \tag{2.6}$$

In equation 2.6, $\bar{r}_i$ denotes the mean ratings on item i:

$$\bar{r}_i = \frac{\sum_{u \in U_i} r_{u,i}}{|U_i|} \tag{2.7}$$

Item-based methods could produce more accurate prediction and efficient than user based-methods [14].

The time complexity of item-based approaches is $O(M^2xNxK)$ for the neighborhood model construction and $O(K)$ for the rating prediction. The space complexity is O(MxK) [5].

2.3.3 Model-based methods

Model-based approaches provide a systematic way to train a predefined compact model in that explains observed ratings, which is then used to make predictions. The general idea is to build a model offline and use that model for online rating. For example, the system can build cluster model to divide users into k groups based on similarities. Then in order to determine the community of a user, we need to solve a classification problem. These approaches potentially offers the benefits of both speed and scalability.

The model building process is performed by different machine learning algorithms such as Bayesian network, clustering, and rule-based approaches. The state-of-the-art model-based methods include restricted Boltzmann machines [15], SVD++ [16], Probabilistic Matrix Factorization (PMF) [17], and multi-domain collaborative filtering [18], graphical models [19], pair-wise tensor factoriza-

tion [20], and matrix factorization with social regularization [21], etc.

There are several advantages of model-based collaborative filtering methods:

Scalability: In model-based algorithms, precomputed models are much smaller than the actual dataset. So the model is used efficiently even for very large datasets. This improves scalability of the systems.

Prediction speed: Model-based collaborative filtering methods can quickly produce the recommendation for they use pre-computed model. This advantage is very importance to commercial applications such as online E-commerce websites.

Despite these advantages, model-based methods have a limitation. Building a model is often a time- and resource-consuming process, it is executed periodically. Therefore, it is difficult to add new data to systems, making them inflexible.

2.4 Hybrid recommendation approaches

Content-based and collaborative filtering approaches have been widely used in commercial and research areas, but they still have many limitations. Therefore, the hybrid approach has been introduced to avoid the limitations of pure approaches. Hybrid recommender systems combine multiple techniques of collaborative approaches and contentbased approaches together to produce its output. Hybrid approaches are classified into seven different types [22]

Weighted: the score of a item is a combination of the scores provided by different recommendation components using a linear combination or a voting scheme.

Switching: this is a special case of the weighted type considering binary weights. The system chooses only one recommendation technique among the others and applies it.

Mixed: recommendations from several components are available, and are presented together at the same time by means of certain ranking or combination strategy.

Feature combination: the features used by different recommenders are integrated into a single data source, which is exploited by a single recommender.

Feature augmentation: the output of a recommender is used as an additional input feature for the next recommender.

Cascade: the recommendation is performed as a sequential process. The next recommender refines the recommendations given by the previous one.

Meta-level: the model generated by a recommender is used as the input for the next one. As stated in [23]: "this differs from feature augmentation: in an augmentation hybrid, we use a learned model to generate features for input to a second algorithm; in a meta-level hybrid, the entire model becomes the input."

Chapter 3

Incremental SVD++

3.1 Singular Value Decomposition based models

Before delving into models, we first describe the notations used in this chapter. Suppose that we are given a set of M users $U = \{u_1, u_2, ..., u_N\}$ and a set of N items $I = \{i_1, i_2, ..., i_M\}$. Users' rating on the items are arranged in a $M \times N$ matrix R where entry $r_{u,i}$ denote the rating that user u gives to item i and $r_{u,i} = 0$ if u have not rated i. The predicted value of $r_{u,i}$ is denoted by $\hat{r}_{u,i}$. The set of all items that user u have rated is denoted by I_u and the set of all users who have rated item i is denoted by U_i. Alternatively, we denote the set of all observed triplet $(u, i, r) \in K$ where u is the user id, i is the item id and r is the rating given by u to i. The whole set is denoted by K.

3.1.1 Fundamentals of Matrix factorization

In recent years, many SVM algorithms such as SVM, boosting, ... was exploited by researchers to solve recommendation problems. One of the most popular algorithms is latent factor models. The underlining assumption of latent factor model is that we can find some hidden factors which could link users and items.

And that helps to explain the ratings.

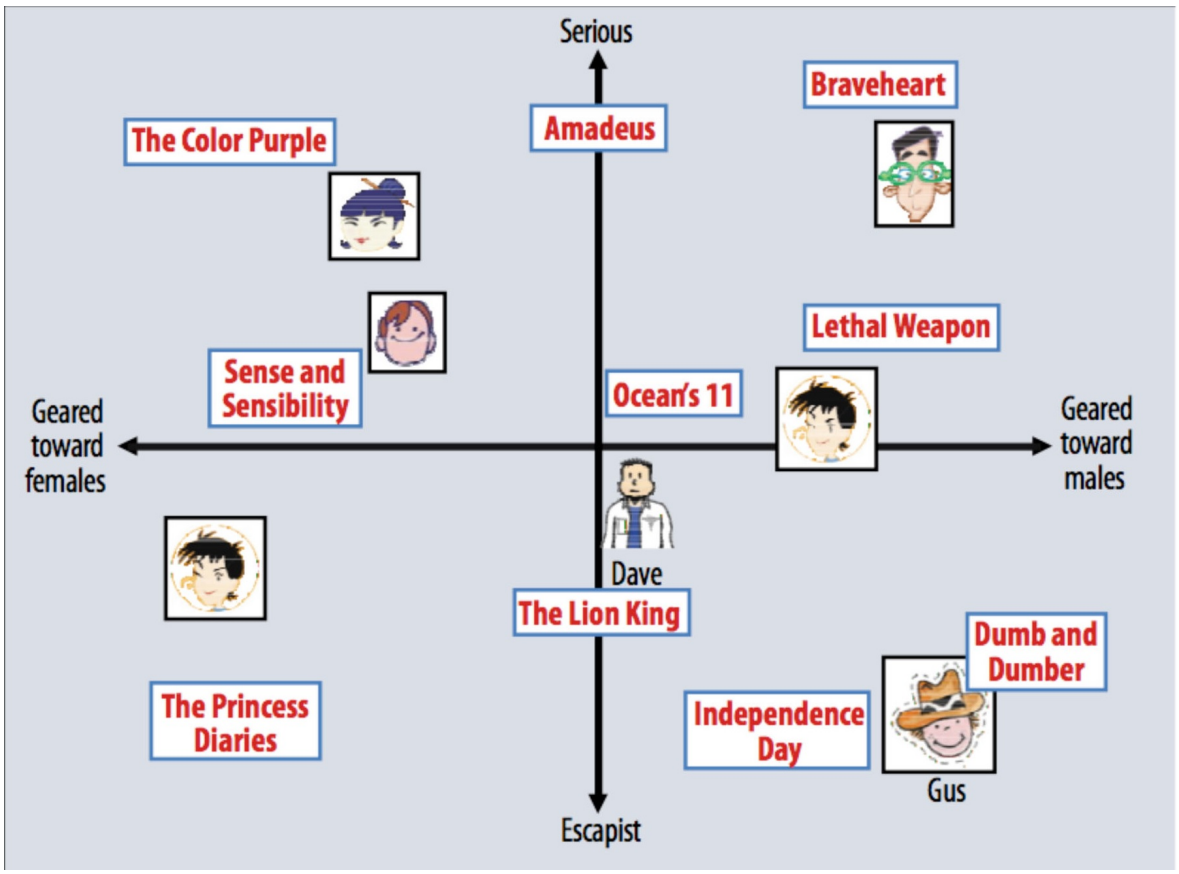

FIGURE 3.1
A SIMPLIFIED ILLUSTRATION OF THE LATENT FACTOR APPROACH, WHICH CHARACTERIZES BOTH USERS AND MOVIES USING TWO AXES—MALE VERSUS FEMALE AND SERIOUS VERSUS ESCAPIST.

SOURCE: NETFLIX PRIZE DIAGRAM [16]

Some of the most successful realizations of latent factor models are based on matrix factorization. Matrix factorization based methods use low-rank matrix to approximate user item rating matrix R. They learn two low-rank matrices, one captures users' latent features - P and the other capture items' latent features- Q^T.

The product of P and Q^T approximately equals to R.

$$R \approx P \times Q^T = \widehat{R} \tag{3.1}$$

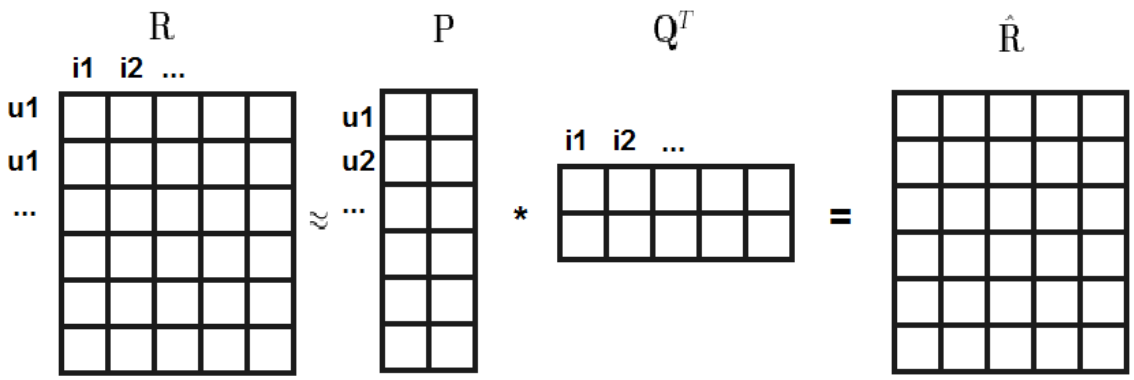

FIGURE 3.2
A ILLUSTRATION OF THE MATRIX FACTORIZATION

The approximation is to minimize the error between the predicted and the original rating matrix.

Matrix factorization techniques

There are two standard approaches to learn P and Q^T: stochastic gradient descent(SGD) and alternating least squares (ALS).

Stochastic gradient descent(SGD)

Stochastic gradient descent (SGD) is widely used in many machine learning problems. SGD try to find minimums or maximums by iteration. In an SGD (Stochastic Gradient descent) approach, for each example in the training set the error $e_{u,i} \stackrel{\text{def}}{=} r_{u,i} - \hat{r}_{u,i}$ is computed and the parameters (p_u and q_i) are updated by a factor in the opposite direction of the gradient.

$$p_u \leftarrow p_u + \gamma\left(e_{u,i}q_i - \lambda p_u\right)$$

$$q_i \leftarrow q_i + \gamma\left(e_{u,i}p_u - \lambda q_i\right)$$

Alternating Least Squares(ALS)

ALS represents a different approach to optimizing the loss function. The values in P are fixed when solving Q^T and vice versa. This allows for an optimal solving as it makes the optimization problem quadratic.

In this section we describe several matrix factorization techniques like SVD and SVD++.

3.1.2 Baseline predictors

Some users will tend to give higher or lower ratings than others. And some items received higher or lower ratings than others. So we consider two parameter: one parameter b_u for each user and one b_i for each item. The parameters b_u and b_i indicate the observed deviations of user u and item i from the average. For example, the average rating over all movies is 3.7 stars. Furthermore, *Titanic* is better than an average movie, so it receives 0.5 stars above the average. On the other hand, *Joe* is a critical user, who tends to rate 0.3 stars lower than the average. Thus, 0.5 and -0.3 reflects the deviations of *Titanic* and *Joe* from the global average 3.7.

The overall average rating is denoted by μ. Prediction is done by the rule:

$$\hat{r}_{u,i} = \mu + b_u + b_i \tag{3.2}$$

The parameters b_u and b_i are estimated by solving the least squares problem:

$$min_{b_*} \sum_{(u,\, i) \in K} (r_{u,i} - \ \mu + \ b_u + \ b_i)^2 + \ \lambda \left({b_u}^2 + \ {b_i}^2\right) \tag{3.3}$$

A simple gradient descent technique was applied successfully to solving (3.3). We loop through all ratings in the training set. The associated prediction error is denoted by $e_{u,i} \stackrel{\text{def}}{=} r_{u,i} - \hat{r}_{u,i}$. For a given training case $r_{u,i}$, we modify the parameters by moving in the opposite direction of the gradient, yielding:
$b_u \leftarrow b_u + \ \gamma.\,(e_{u,i} - \ \lambda.b_u)$

$b_i \leftarrow b_i + \ \gamma.\,(e_{u,i} - \ \lambda.b_i)$
where γ is the learning rate.

3.1.3 Regularized Singular Value Decomposition (Regularized-SVD)

Singular value decomposition (SVD) is one type of matrix factorization. Matrix factorization models transform both users and items into the same latent feature space of dimensionality f. The prediction is done by taking an inner product in that space:

$$\hat{r}_{u,i} = q_i^T p_u \tag{3.4}$$

where each item i is associated with a vector $q_i \in \mathbb{R}^f$, and each user u is associated with a vector $p_u \in \mathbb{R}^f$. For a given item i, the elements of q_i measure the extent to which the item possesses those factors; for a given user u, the elements of p_u measure the extent of interest the user has in items that are high on the corresponding factors. Therefore, their dot product $q_i^T p_u$ denotes the overall interest of the user in characteristics of the item.

For example, we apply matrix factorization methods on computing the rating that user u_2 gives to item *Apollo*13.

	Titanic	Mummy	Apollo 13	Spider Man
u_1	5	3		1
u_2	4	1	?	1
u_3	1			5
u_4	1	1		4
u_5			5	4

RATINGS MATRIX $R(m \times n)$, $m = 5$, $n = 4$

Therefore, the rating that user u_2 gives to item *Apollo*13 is:

$\hat{r}_{u2,Apollo13} = 1.96$ x $1.54 + 0.27$ x $2.17 = 3.6$

We learn the values of the factor vectors (p_u and q_i) by minimizing the regularized squared error function associated with (3.4)

	f_1	f_2
u_1	2.44	0.23
u_2	1.96	0.27
u_3	0.46	2.16
u_4	0.46	1.75
u_5	0.46	1.77

USERS FEATURES MATRIX $P_{m\times k}$, $k = 2$

	Titanic	Mummy	Apollo 13	Spider Man
f_1	1.95	1.14	1.54	0.22
f_2	0.06	0.18	2.17	2.13

ITEMS FEATURES MATRIX ${Q^T}_{k\times n}$, $k = 2$

$$min_{q_*,p_*} \sum_{(u,\,i)\in K} \left(r_{u,i} - \ q_i^T p_u\right)^2 + \ \lambda \left(\ \|q_i\|^2 + \ \|p_u\|^2 \right) \tag{3.5}$$

K is the set of the (u, i) pairs for which $r_{u,i}$ is known.

The first term $\left(r_{u,i} - \ q_i^T p_u\right)^2$ strives to find q_i's, p_u's that fit the given ratings. The regularizing term – $\lambda \left(\ \|q_i\|^2 + \ \|p_u\|^2 \right)$ – avoids overfitting by penalizing the magnitudes of the parameters.

A simple gradient descent technique was applied successfully to solving (3.5). We loop through all ratings in the training set. The associated prediction error is denoted by $e_{u,i} \stackrel{\text{def}}{=} \ r_{u,i} - \hat{r}_{u,i}$. For a given training case $r_{u,i}$, we modify the parameters by moving in the opposite direction of the gradient, yielding:

$q_i \leftarrow q_i + \ \gamma \left(e_{u,i} p_u - \lambda q_i\right)$

$p_u \leftarrow p_u + \ \gamma \left(e_{u,i} q_i - \lambda p_u\right)$

where γ is the learning rate.

3.1.4 SVD with Bias Terms(Bias-SVD)

We add biases to the regularized SVD model. Therefore, in this model, a rating is predicted by the rule:

$$\hat{r}_{u,i} = \mu + b_u + b_i + q_i^T p_u \tag{3.6}$$

To learn the model parameters, we should also minimize the regularized squared error:

$$min_{q_*,p_*,\ b_*} \sum_{(u,\ i)\in K} \left(r_{u,i} - \mu + b_u + b_i + q_i^T\ p_u\right)^2 + \lambda \left({b_u}^2 + {b_i}^2 + \|q_i\|^2 + \|p_u\|^2\right) \tag{3.7}$$

Similarly, we use the gradient descent method to get the update rule for each parameter:

$b_u \leftarrow b_u + \gamma.(e_{u,i} - \lambda_1.b_u)$

$b_i \leftarrow b_i + \gamma.(e_{u,i} - \lambda_1.b_i)$

$p_u \leftarrow p_u + \gamma.(e_{u,i}.q_i - \lambda_2.p_u)$

$q_i \leftarrow q_i + \gamma.(e_{u,i}.p_u - \lambda_2.q_i)$

where $e_{u,i} \overset{\text{def}}{=} r_{u,i} - \hat{r}_{u,i}$ and γ is the learning rate.

3.1.5 SVD with implicit feedback (SVD++)

Explicit and implicit feedback

Recommender systems rely on various types of input such as explicit feedback and implicit feedback.

Explicit feedback includes explicit input by users regarding their interest in items. For example, Amazon collects star ratings for products and TiVo users indicate their preferences for TV shows by hitting thumbs-up/down buttons. It is difficult to obtain explicit feedback from a population of users because it requires effort from the users and also, users are not always ready to supply enough information.

On the other hand, implicit feedback is abundant. Implicit feedback indirectly reflect opinion through observing user behaviors [24]. Types of implicit

feedback include purchase history, browsing history, search patterns, or even mouse movements. For example, a user who purchased many books by the same author probably likes that author or a user listens to a track 5 times may has an interest in that track.

Explicit feedback is generally more accurate than implicit feedback in representing the user's interests. Also, explicit feedback can be positive or negative, whereas implicit feedback is only positive. A combination of explicit feedback and implicit feedback can minimize their weaknesses and perform a better recommendation.

SVD with implicit feedback (SVD++)

SVD++ is another extension of SVD method. SVD++ uses implicit feedback as a secondary source of information. It also combines bias terms into the system. So this model produces more accurate prediction than other factor models.

For simplicity, we consider the rated history of users as a part of input data. This type of implicit feedback can be extracted from ratings matrix by reducing the ratings matrix into a binary matrix. $p_{u,i}$, which indicates the preference of user u to item i, are derived by binarizing the $r_{u,i}$ values:

$$q_{u,i} = \begin{cases} 1 & \text{if } r_{u,i} > 0 \\ -0 & \text{if } r_{u,i} = 0 \end{cases} \tag{3.8}$$

The set of all items for which u provided an implicit preference is denoted by N_u. To keep generality, each user u is associated with two sets of items I_u and N_u.

To this end, a second set of item factors is added, relating each item i to a factor vector $y_i \in \mathbb{R}^f$. Those new item factors are used to characterize users

based on the set of items that they rated. The exact model is as follows:

$$\hat{r}_{u,i} = \mu + b_i + b_u + q_i^T \left(p_u + |N_u|^{-\frac{1}{2}} \sum_{j \in N_u} y_j \right) \tag{3.9}$$

Now, a user u is modeled as $p_u + |N_u|^{-\frac{1}{2}} \sum_{j \in N_u} y_j$. We use a free user-factors vector, p_u, much like in (3.6), which is learnt from the given explicit ratings. This vector is complemented by the sum $|N_u|^{-\frac{1}{2}} \sum_{j \in N_u} y_j$, which represents the perspective of implicit feedback. Since the y_j's are centered around zero (by the regularization), the sum is normalized by $|N_u|^{-\frac{1}{2}}$, in order to stabilize its variance across the range of observed values of N_u.

As usual, we learn the values of involved parameters by minimizing the regularized squared error function associated with (3.10):

$$\begin{aligned} min_{q_*, p_*, \ b_*, y_*} \sum_{(u,\, i) \in K} \left(r_{u,i} - \mu + b_u + b_i + q_i^T \left(p_u + |N_u|^{-\frac{1}{2}} \sum_{j \in N_u} y_j \right) \right)^2 \\ + \lambda \left({b_u}^2 + {b_i}^2 + \|q_i\|^2 + \|p_u\|^2 + \sum_{j \in N_u} \|y_j\|^2 \right) \end{aligned} \tag{3.10}$$

Model parameters are determined by minimizing the associated regularized squared error function through stochastic gradient descent. We loop over all training set, computing:

$$b_u \leftarrow b_u + \gamma . (e_{u,i} - \lambda_1 . b_u)$$

$$b_i \leftarrow b_i + \gamma . (e_{u,i} - \lambda_1 . b_i)$$

$$q_i \leftarrow q_i + \gamma . \left(e_{u,i} . \left(p_u + |N_u|^{-\frac{1}{2}} \sum_{j \in N_u} y_j \right) - \lambda_2 . q_i \right)$$

$$p_u \leftarrow p_u + \gamma . (e_{u,i} . q_i - \lambda_2 . p_u)$$

$$\forall j \in N_u :$$

$$y_j \leftarrow y_j + \gamma . \left(e_{ui} . |N_u|^{-\frac{1}{2}} . q_i - \lambda_2 . y_j \right)$$

The SVD++ algorithm is summarized in the following pseudo-code:

Data: K

Result: Updated model

Initialize model;

repeat

  forall $(u, i, r) \in K$ **do**

    $e_{u,i} = r_{u,i} - \hat{r}_{u,i}$

    $\Delta b_u = e_{u,i} - \lambda_1 . b_u$

    $\Delta b_i = e_{u,i} - \lambda_1 . b_i$

    $\Delta p_u = e_{u,i} . q_i - \lambda_2 . p_u$

    $\Delta q_i \leftarrow e_{ui} . \left(p_u + |N_u|^{-\frac{1}{2}} \sum_{j \in \in N_u} y_j \right) - \lambda_2 . q_i$

    $b_u = b_u + \gamma . \Delta b_u$

    $b_i = b_i + \gamma . \Delta b_i$

    $p_u = p_u + \gamma . \Delta b_u$

    $q_i = q_i + \gamma . \Delta q_i$

    forall $j \in N_u$ **do**

      $y_j = y_j + \gamma . \left(e_{ui} . |N_u|^{-\frac{1}{2}} . q_i - \lambda_2 . y_j \right)$

    end forall

  end forall

until *convergence*;

Algorithm 1: SVD++

Several types of implicit feedback can be simultaneously introduced into the model by using extra sets of item factors. For example, if a user u has a certain kind of implicit preference to the items in ${N^1}_u$ (e.g., she rented them), and a different type of implicit feedback to the items in ${N^2}_u$ (e.g., she browsed them), we could use the model:

$$r_{u,i} = \mu + b_i + b_u + q_i^T \left(p_u + \left|{N^1}_u\right|^{-\frac{1}{2}} \sum_{j \in {N^1}_u} y_j + \left|{N^2}_u\right|^{-\frac{1}{2}} \sum_{j \in {N^2}_u} y_j \right) \quad (3.11)$$

The relative importance of each source of implicit feedback will be automatically learned by the algorithm by its setting of the respective values of model parameters.

3.2 Incremental SVD++

Most users' preferences are more or less consistent throughout a short period of time. In other words, if a user enjoys a certain type of music today, we assume that she will enjoy the same genre in the near future. This assumption is needed so that the collected past information can be utilized to produce recommendations for the future. If all users change their preferences, there is no way for a recommender system to make any meaningful recommendations. In the other hand, the consistency is assumed for a short period of time. In real life scenarios, users do change preferences throughout time. The change might due to the awareness of new things, change of attitude, etc. So we must keep the recommender system up to date.

Figure 3.3 shows the process of a recommender system which can solve the above problem. This recommender system are trained using both online and offline training algorithms.

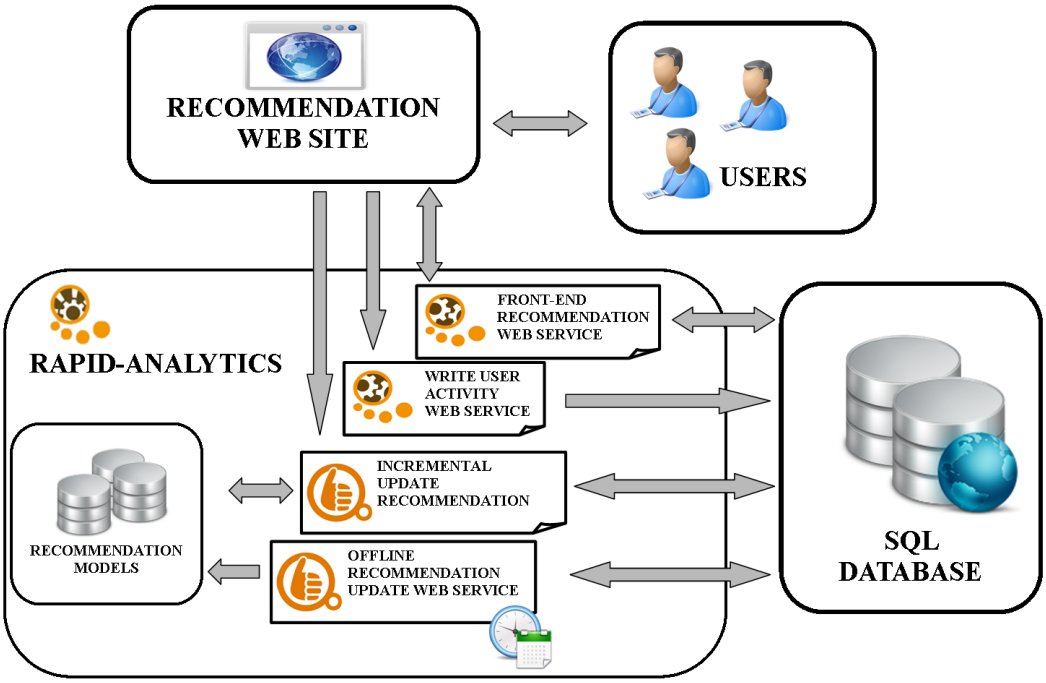

Figure 3.3
The process of a recommender system
Source: The Rapid-I Marketplace.

The main task of SVD++ model is creating the features of users and items by minimizing the associated regularized squared error function through stochastic gradient descent. So it must loop for all known rating to get the best fit model. This becomes a big problem when applying the model in a large-scale dataset.

We make an assumption that is the new ratings do not effect the hold system. Fox example, when user Joe rates a movie - Mummy. The features of Joe and Mummy change and need to be updated. But the features of another user Harry who has no relation to Joe or Mummy do not change. We propose Incremental SVD++, a new method based on the SVD++ technique. By cutting down the size of training data, our method significantly reduces the training time of classic SVD++, while maintaining its recommendation quality.

3.2.1 Notations

We describe again some notations. The set of all items that user u have rated is denoted by I_u and the set of all users who have rated item i is denoted by U_i. The set of all ratings user u have rated is denoted by $R_{u,*}$ and the set of all ratings for items i is denoted by $R_{*,i}$.

3.2.2 Algorithm

Given that $r_{x,y}$ is the new rating given by user x to item y. In our method, we divide the set of known ratings K into two subset: S_1 and S_2. S_1 includes all the ratings that have directly relation with x or y such as $R_{x,*}$ the known ratings given by x and $R_{*,y}$ the known ratings for y. S_2 includes the rest known ratings which have no directly relation with x or y.

For example, we have a ratings matrix:

	A	B	C	D
u_1	5	3		1
u_2	4	1	4	1
u_3	1			5
u_4	1	1		4
u_5			5	4

RATINGS MATRIX $R_{m\times n}$, $m = 5$, $n = 4$

User u_2 gives a new rating for C. So

$R_{*,u_2} = \{r_{u_2A}, r_{u_2B}, r_{u_2C}, r_{u_2D}\}$

$R_{C,*} = \{r_{u_2C}, r_{u_5C}\}$

The set of related ratings:

$S_1 = \{r_{u_2A}, r_{u_2B}, r_{u_2C}, r_{u_2D}, r_{u_5C}\}$

The set of unrelated ratings:

$S_2 = \{r_{u_1A}, r_{u_1B}, r_{u_1D}, r_{u_3A}, r_{u_3D}, r_{u_4A}, r_{u_4B}, r_{u_4D}, r_{u_5D}\}$

The idea of using matrix factorization is build a features model that determine how a user rates an item. This feartures model contents features vectors of

	A	B	C	D
u_1				
u_2	4	1	4	1
u_3				
u_4				
u_5			5	

Related ratings matrix

	A	B	C	D
u_1	5	3		1
u_2				
u_3	1			5
u_4	1	1		4
u_5				4

Unrelated ratings matrix

all users and items. And to learn this model, MF systems minimize the regularized squared error on the training set. In SVD and SVD++ approaches, training set is known ratings. Therefore, the model can explain all known ratings. With a new rating, the system updates the set of known rating and re-computes the model.

In Incremental SVD++ method, we try to reduce the training set.

We consider that:

$$min_{q_*,p_*,\ b_*,y_*} \sum_{(u,\,i)\in K} \left(r_{u,i} - \ \mu + \ b_u + \ b_i + \ q_i^T \ \left(p_u + |N_u|^{-\frac{1}{2}} \sum_{j\in N_u} y_j \right) \right)^2 \tag{3.12}$$
$$+ \lambda \left({b_u}^2 + \ {b_i}^2 + \ \|q_i\|^2 + \ \|p_u\|^2 + \sum_{j\in N_u} \|y_j\|^2 \right)$$

is equevelant to the sum of (3.13) and (3.14) where $K = S_1 \cup S_2,\ S_1 \cap S_2 = \emptyset$,

and the two components of the regularized squared error function are positive.

$$min_{q_*,p_*,\ b_*,y_*} \sum_{(u,\,i)\in S_1} \left(r_{u,i} - \ \mu + \ b_u + \ b_i + \ q_i^T \ \left(p_u + |N_u|^{-\frac{1}{2}} \sum_{j\in N_u} y_j \right) \right)^2 + \lambda \left({b_u}^2 + \ {b_i}^2 + \ \|q_i\|^2 + \ \|p_u\|^2 + \sum_{j\in N_u} \|y_j\|^2 \right) \tag{3.13}$$

$$min_{q_*,p_*,\ b_*,y_*} \sum_{(u,\,i)\in S_2} \left(r_{u,i} - \ \mu + \ b_u + \ b_i + \ q_i^T \ \left(p_u + |N_u|^{-\frac{1}{2}} \sum_{j\in N_u} y_j \right) \right)^2 + \lambda \left({b_u}^2 + \ {b_i}^2 + \ \|q_i\|^2 + \ \|p_u\|^2 + \sum_{j\in N_u} \|y_j\|^2 \right) \tag{3.14}$$

To minimize the regularized squared error on the training set, we minimize the regularized squared error on each subset. The idea of Incremental SVD++ is new ratings only affect the features of related users and items. We assume that new rating $r_{x,y}$ only affect the features of user x and item y. All the factors vectors and bias parameters of others users and others items unchanged.

Model parameters are determined by minimizing the associated regularized squared error function through conditional gradient descent on the related ratings S_1. Obviously, 3.14 is unchanged, so it is not concerned in the learning process.

O is denoted as the set of objects (e.g. x, y) which need to be updated. The Incremental SVD++ is summarized in the following pseudo-code:

Data: S_1, O, users bias, items bias, P, Q, Y

Result: Updated model

repeat
  forall $(u, i, r) \in S_1$ **do**
    $e_{u,i} = r_{u,i} - \hat{r}_{u,i}$
    $\Delta b_u = e_{u,i} - \lambda_1 . b_u$
    $\Delta b_i = e_{u,i} - \lambda_1 . b_i$
    $\Delta p_u = e_{u,i} . q_i - \lambda_2 . p_u$
    $\Delta q_i \leftarrow e_{ui} . \left(p_u + |N_u|^{-\frac{1}{2}} \sum_{j \in \in N_u} y_j \right) - \lambda_2 . q_i$
    `//this is an extra condition in Incremental SVD++`
    if $u \in Q$ **then**
      $b_u = b_u + \gamma . \Delta b_u$
      $p_u = p_u + \gamma . \Delta b_u$
    end if
    `//this is an extra condition in Incremental SVD++`
    if $i \in Q$ **then**
      $b_i = b_i + \gamma . \Delta b_i$
      $q_i = q_i + \gamma . \Delta q_i$
    end if
    forall $j \in N_u$ **do**
      `//this is an extra condition in Incremental SVD++`
      if $j \in Q$ **then**
        $y_j = y_j + \gamma . \left(e_{ui} . |N_u|^{-\frac{1}{2}} . q_i - \lambda_2 . y_j \right)$
      end if
    end forall
  end forall
until *regularized squared error* ≈ 0;

Algorithm 2: Incremental SVD++

Similar to SVD++, the rating is predicted by the following equation:

$$\hat{r}_{u,i} = \mu + b_i + b_u + q_i^T \left(p_u + |N_u|^{-\frac{1}{2}} \sum_{j \in N_u} y_j \right) \tag{3.15}$$

Note that each time the model is updated, the overall average rating changes. So in equation 3.15, μ is calculated at the last update of user u's factors.

Chapter 4

Experiments

In this section, we conduct experiments to compare the performance of our Incremental SVD++ algorithm with classic SVD++ algorithm.

4.1 Experiment setup

4.1.1 Data

We choose MovieLens[1] dataset to study empirical performance of our algorithms. The basic statistics of the dataset.

	MovieLens
No. ratings	100 000
No. ratings of training set	90 000
No. ratings of test set	10 000
No. users	5949
No. items	3295
Rating range	$[1, 5]$
Rating type	Integer

Table 4.1
Statistics of MovieLens dataset

The data set is randomly from the MovieLens 1M dataset. All the ratings

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

are ordered by the time stamp.

4.1.2 Parameters

The meta-parameters were the same in all experiments and set as follows: number of factors $k = 100$. $\gamma = \lambda = 0.01$.

4.1.3 Environment

All experiments were run on a PC with Intel Core i3-2310M @ 2.10GHz and 4 GB RAM, running Windows 7.

4.2 Evaluation measures

We adopt Root Mean Square Error(RMSE) to evaluate two algorithms i.e. SVD++ and Incremental SVD++. RMSE evaluates the root of average square error between true rating and predicted rating. Denote the test set by T , the definition of RMSE is given below:

$$RMSE = \sqrt{\frac{\sum_{(u,i,r)\in T} (\widehat{r_{u,i}} - r)^2}{|T|}} \tag{4.1}$$

4.3 Experiment Result

4.3.1 Comparison between SVD++ and ISVD++

The aim of these experiment is to compare the accuracy and training time of SVD++ and Incremental SDV++ in dynamic scenario where ratings happen continuously. Experimental scenario:

1. The model is pre-buit with a subset of 20000 ratings.

2. Each n new ratings, the model is updated by SVD++ and Incremental SVD++ (ISVD++).

3.Test set includes 10000 newest ratings.

The number of new ratings will be changed in three experiments:

- Experiment #1: $n = 500$
- Experiment #2: $n = 2000$
- Experiment #3: $n = 10000$

Experiment #1: $n = 500$

In this experiment, we compare training time and accuracy of two algorithm: SVD++ and Incremental SDV++. The model will be re-computed per 500 new ratings.

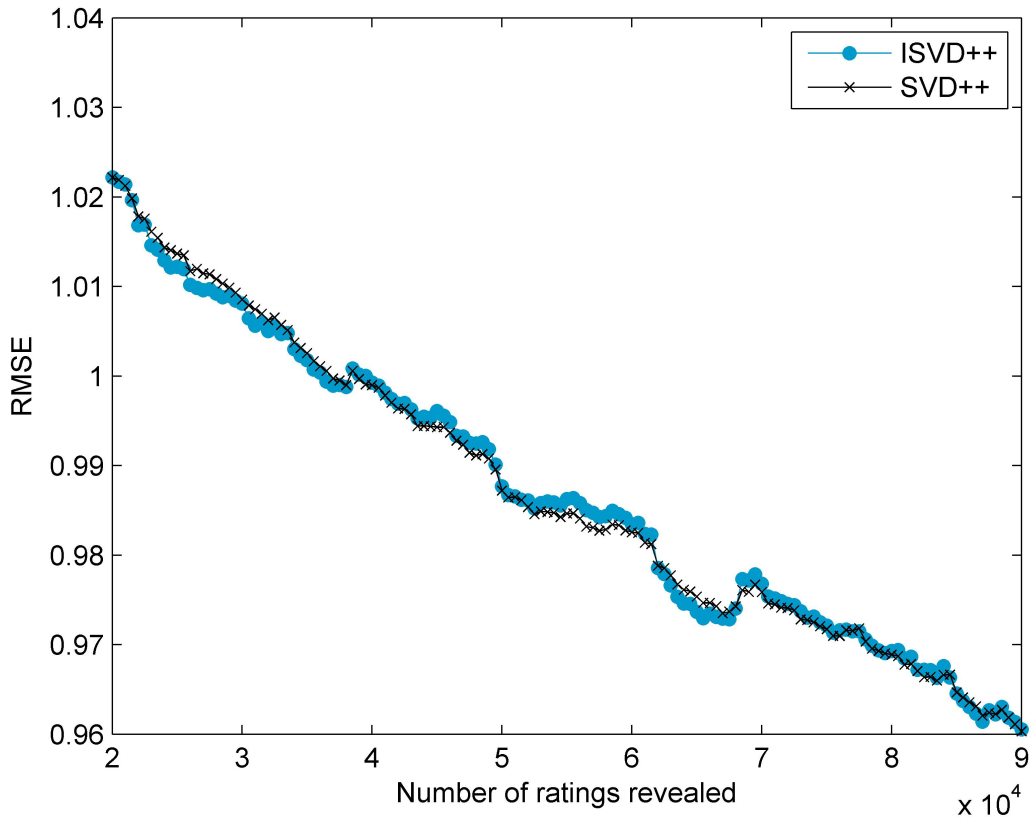

FIGURE 4.1
RECOMMENDATION ACCURACY (RMSE) COMPARISON BETWEEN THE PROPOSED INCREMENTAL SVD++ AND SVD++ ALGORITHM IN MOVIELENS

Figure 4.1 shows a comparison between SVD++ and ISVD++ based on their recommendation accuracies. Two methods constantly updates the model after each 500 new ratings. In general, RMSE value of both ISVD++ and SVD++ significantly decrease and are approximate.

At some point, the RMSE value increases. The quality of new ratings can be the explanation for these certainty increasing. Recommender systems update

model with new data in order to "catch up" the preferences of users. But new data does not always expresses the users' preferences. For example, Joe loves classical music. One day, he heard a rock song and loved it. But it do not mean that Joe loves rock music. In this situation, the updating new data will decrease the recommendation accuracy.

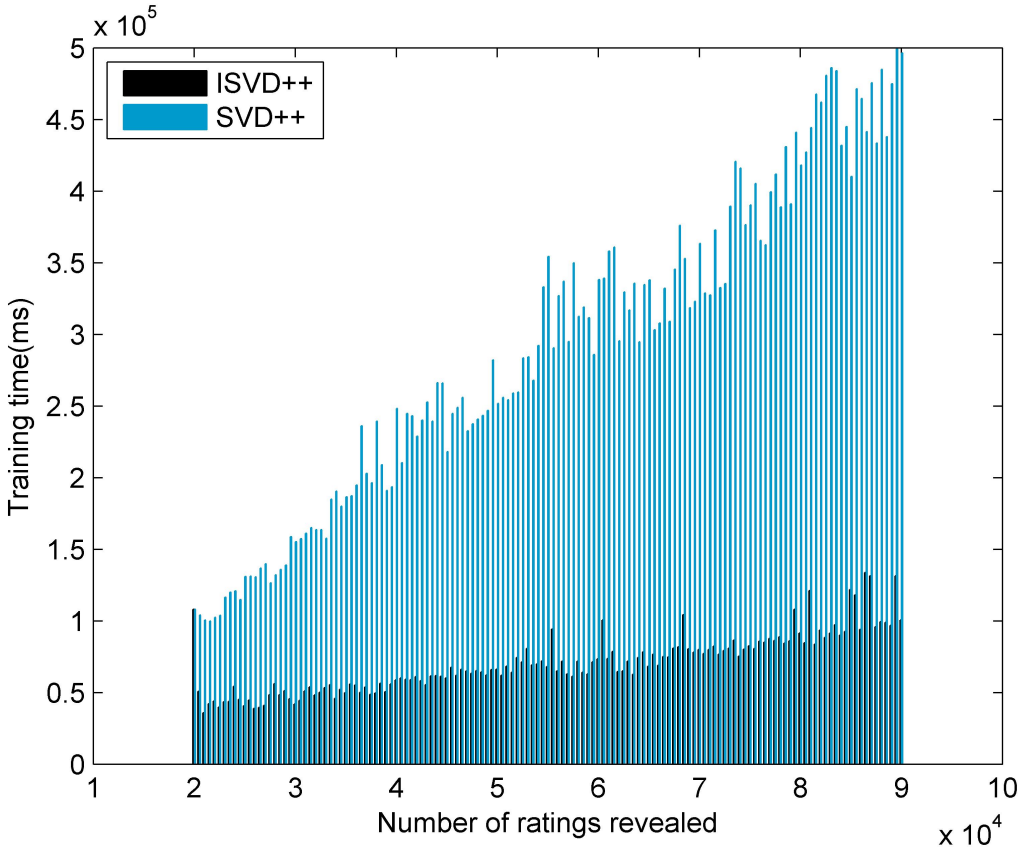

FIGURE 4.2
TRAINING TIME COMPARISON BETWEEN THE PROPOSED INCREMENTAL SVD++ AND SVD++ ALGORITHM IN MOVIELENS

Figure 4.2 presents the times required for SVD++ and ISVD++ to update the model after each 500 new ratings. It can be seen that once the model has been trained, the time ISVD++ requires for updating is significantly lower than that of SVD++. The training time of ISVD++ and SVD++ increase along with the increase in revealed ratings. While, there is a sharp rise in training time of SVD++, the training time of ISVD++ slightly rises.

Experiment #2: $n = 2000$

In this experiment, we compare training time and accuracy of two algorithm: SVD++ and Incremental SDV++. The model will be re-computed per 2000 new ratings.

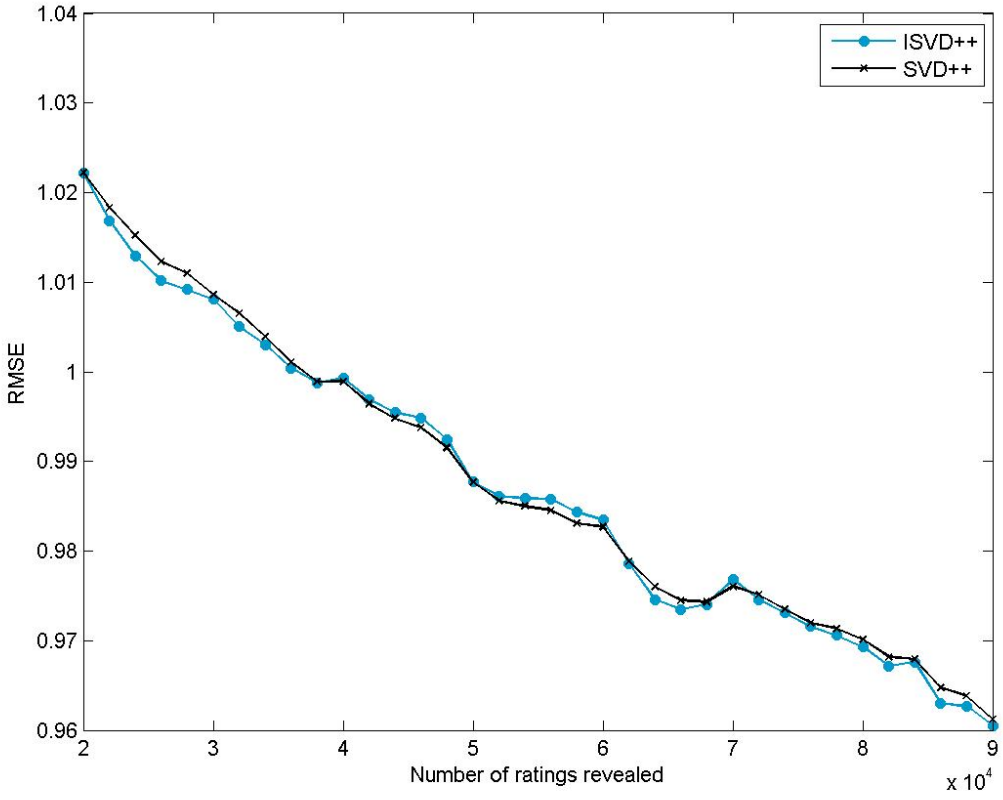

FIGURE 4.3
RECOMMENDATION ACCURACY (RMSE) COMPARISON BETWEEN THE PROPOSED INCREMENTAL SVD++ AND SVD++ ALGORITHM IN MOVIELENS

Figure 4.3 shows a comparison between SVD++ and ISVD++ based on their recommendation accuracies. Two methods constantly updates the model after each 2000 new ratings.In general, RMSE value of both ISVD++ and SVD++ significantly decrease and are approximate.

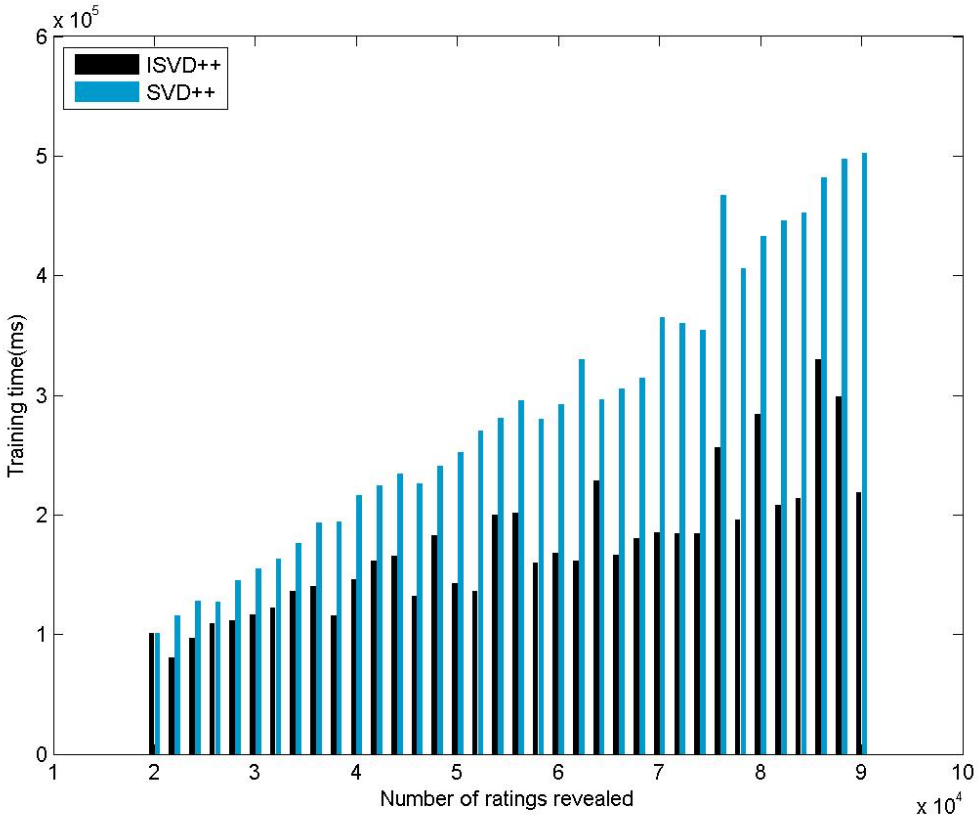

FIGURE 4.4
TRAINING TIME COMPARISON BETWEEN THE PROPOSED INCREMENTAL SVD++ AND SVD++ ALGORITHM IN MOVIELENS

Figure 4.4 compares the times required for SVD++ and ISVD++ to update the model after each 2000 new ratings. Similar to experiment #1, in this experiment ISVD++ requires less time to update model than SVD++. However, the distance between training times of two algorithms decreases because of the increase of number of new ratings in each update.

Experiment #3: $n = 10000$

In this experiment, we compare training time and accuracy of two algorithm: SVD++ and Incremental SDV++. The model will be re-computed per 10000 new ratings.

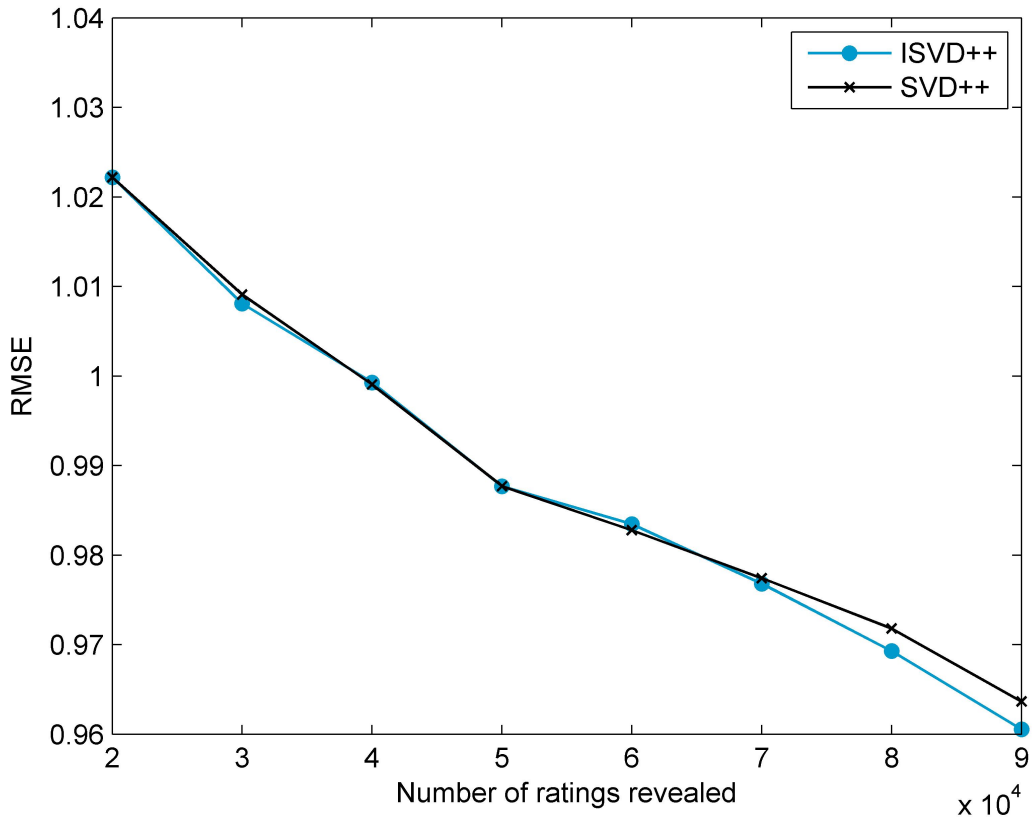

FIGURE 4.5
RECOMMENDATION ACCURACY (RMSE) COMPARISON BETWEEN THE PROPOSED INCREMENTAL SVD++ AND SVD++ ALGORITHM IN MOVIELENS

Figure 4.5 shows a comparison between SVD++ and ISVD++ based on their recommendation accuracies. Two methods constantly updates the model after each 10000 new ratings.Similar to experiment #1 and experiment #2, RMSE value of both ISVD++ and SVD++ in this experiment sharply decrease and are approximate.

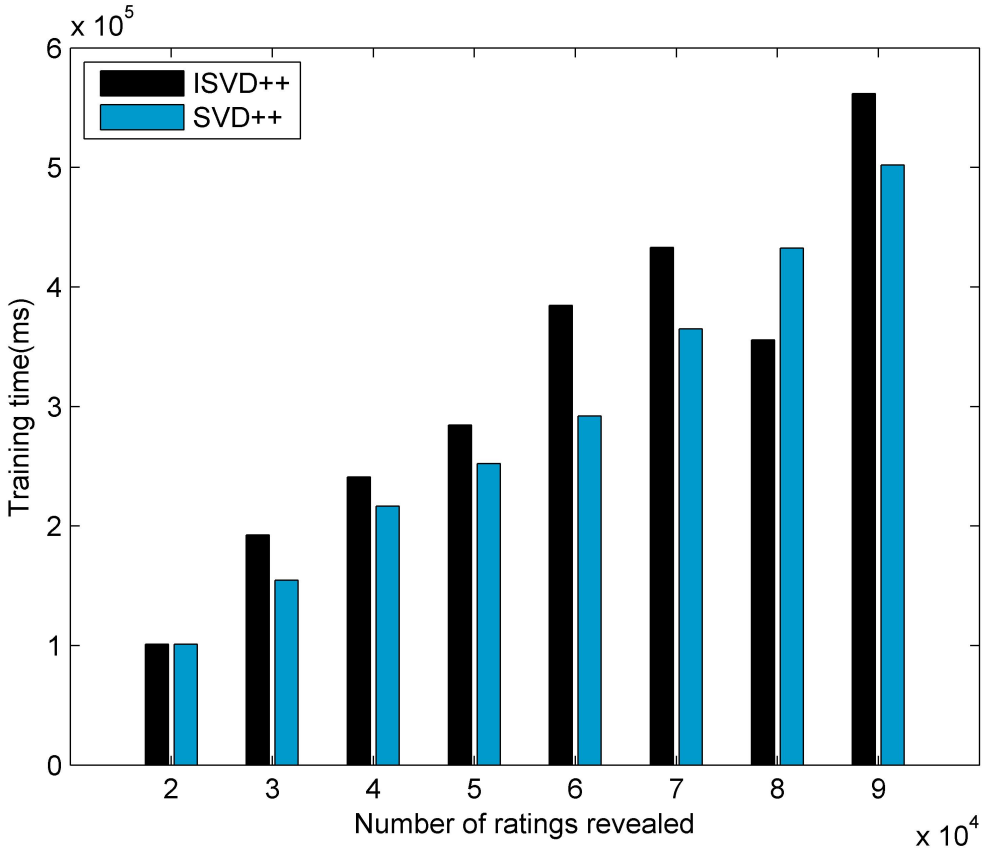

FIGURE 4.6
TRAINING TIME COMPARISON BETWEEN THE PROPOSED INCREMENTAL SVD++ AND SVD++ ALGORITHM IN MOVIELENS

Figure 4.6 compares the times required for SVD++ and ISVD++ to update the model after each 2000 new ratings. The training time of two methods sharply increase. Different than experiment #1 and experiment #2, in this experiment ISVD++ requires more time to update model than SVD++.

Experiment #4:

In this experiment, we evaluate training time of Incremental SDV++ in a bigger dataset. The model will be pre-built with a dataset of 500000 ratings and re-computed per 500 new ratings.

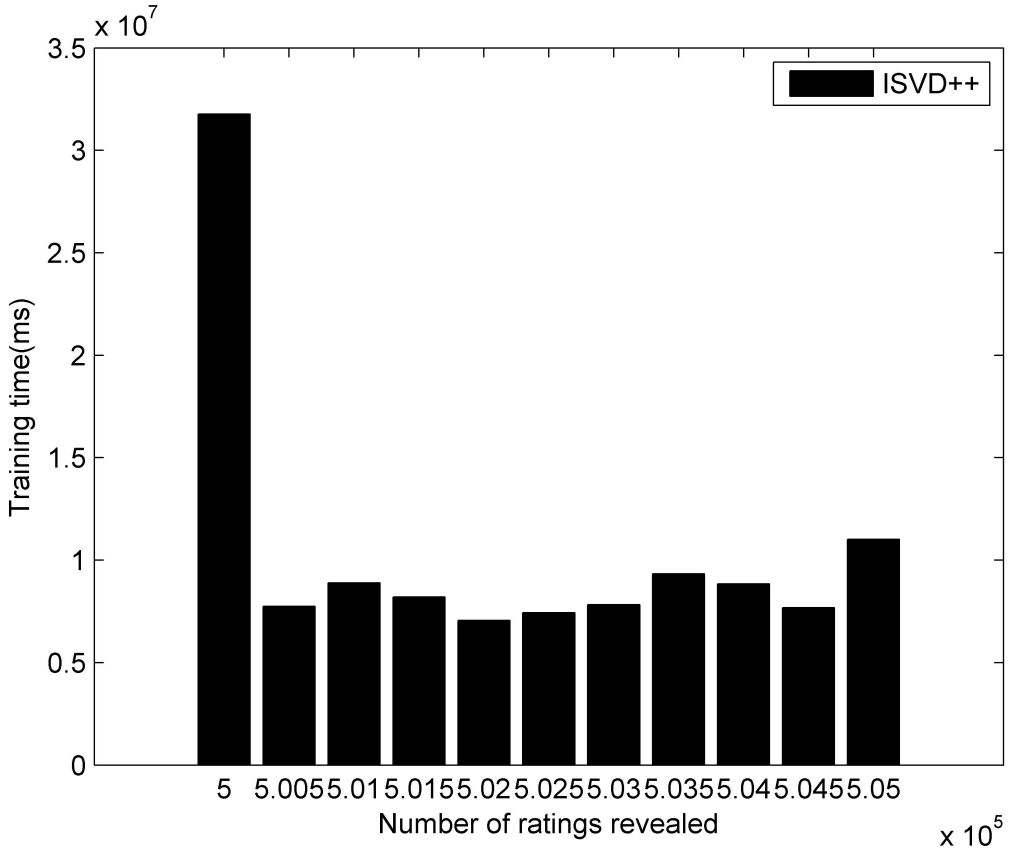

FIGURE 4.7
UPDATE TIME OF INCREMENTAL SVD++

Figure 4.7 shows the times required for ISVD++ to update the model after each 500 new ratings. Comparing this result with the result of experiment #1, the time required for each update is bigger. So the update time if ISVD++ does not only depend on the number of new ratings. On the other hand, it can be seen that the training time of SVD++ is very big (nearly 9 hours for training 500000 ratings).

The result of these experiments shows that Incremental SVD++ produces the same recommendation accuracy as SVD++. In the purpose to apply into real application, Incremental SVD++ has an advantage in the time required for updating compared with SVD++. However, this advantage is affected by many factors. For real application, we must consider the impact of these factors

to build a more effective system. In next section, we evaluate the impact of factors: number of observed ratings, number of new ratings, and number of observed ratings.

4.3.2 Factors affecting updating time

Number of observed ratings and new ratings

In this experiment, we evaluate the impact of number of observed ratings and number of new ratings on the update time of ISVD++. Experimental scenario:

1. The model is pre-buit with a dataset of n ratings. The values of n are 20000, 50000, 10000.

2. With each value of n, the model is updated with different number of new ratings.

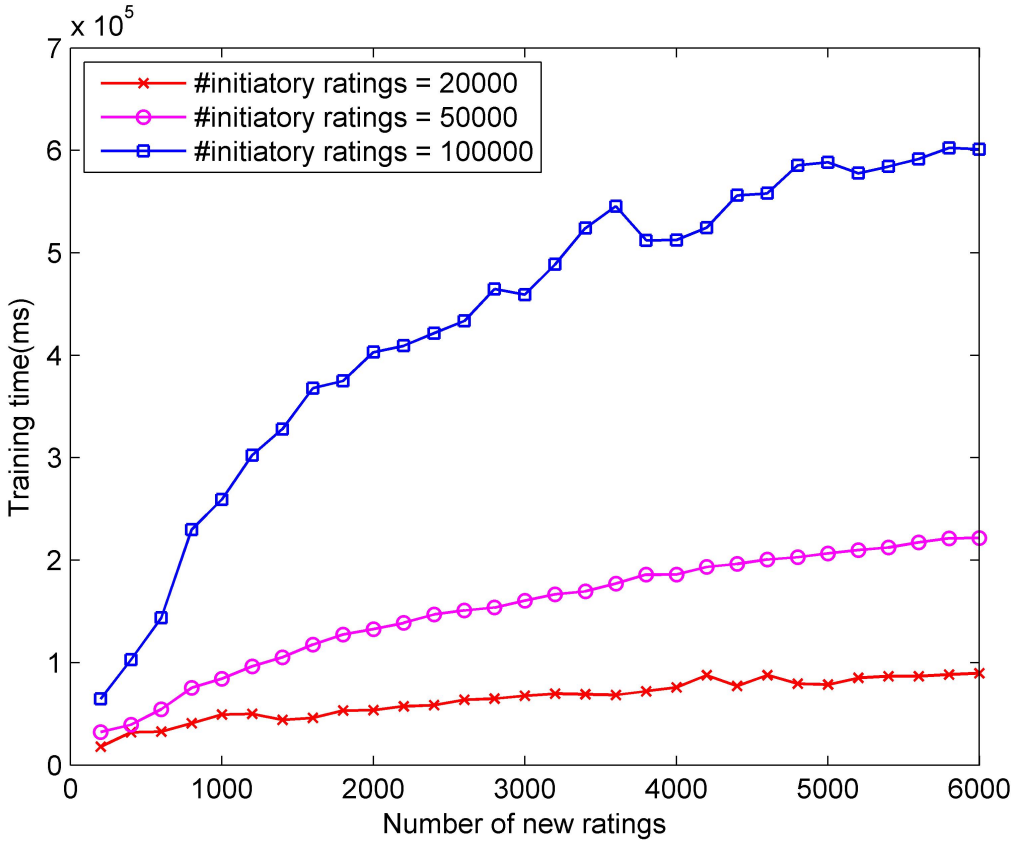

FIGURE 4.8
UPDATE TIMES OF INCREMENTAL SVD++

Figure 4.8 shows the times ISVD++ required for updating model in 3 size of pre-computed data. With the increasing of number of new ratings, the update time is increasing. On the other hand, with the same number of new ratings, the bigger dataset requires more time to update model.

Number of update objects

In this experiment, we evaluate the impact of number of update objects on the update time of ISVD++. Experimental scenario:

1. The model is pre-buit with a dataset of 100000 ratings.

2. The number of new ratings is fixed at 5000

3. The number of update object is changed

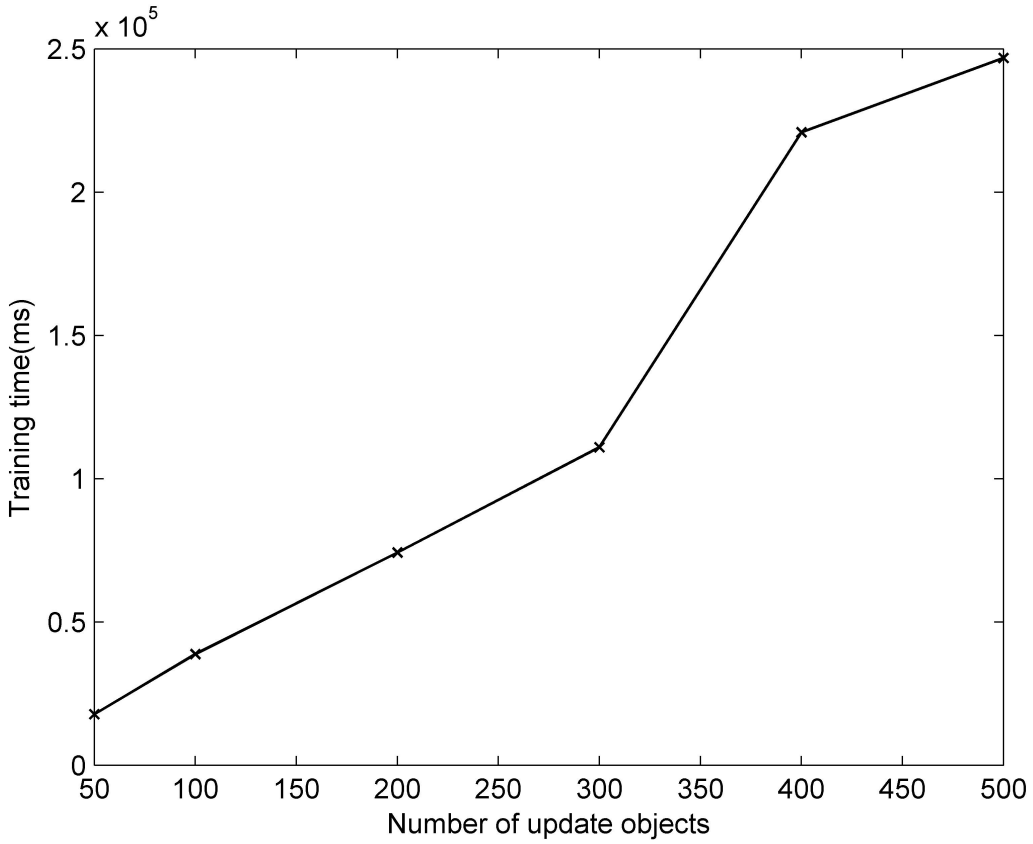

FIGURE 4.9
UPDATE TIMES OF INCREMENTAL SVD++

With fixed number of new ratings and size of pre-computed data, the update time increases when we increase the number of update objects.

In conclusion, the update time of ISVD++ depends on number of observed ratings, number of new ratings, and number of observed ratings. In section 3.2, we mentioned that the update time of ISVD++ is cutting down by updating in related set instead of all observed data. Number of observed ratings and number of observed ratings affect the size of related set. And the number of new ratings has an impact on the size of update objects.

Chapter 5

Conclusion

5.1 Conclusion

Throughout our observation and experiments, Incremental SVD++ proved to be more adaptive to production than its predecessor SVD++. The update time of Incremental SVD++ was much lower than SVD++ which means user will receive faster up-to-date recommended items. In production system, when need a frequently updated recommender system, the training need to be done on-the-fly. Incremental SVD++ would solve the scalability problem and bring about a more effective system.

Although the speed gap from the two algorithms were large, there is virtually no accuracy trade-off as the accuracy of Incremental SVD++ was usually the same as SVD++.

On the other hand, the algorithm still have some major drawbacks that should be kept in mind when using. One of them is to keep track of the update data's size. In some of our experiments we found out that if the size was too large the updating time will be long and if the size was too small, there is virtually no impact to the overall result.

5.2 Future work

Incremental SVD++ has a major weakness that it can not run as an offline algorithms. If we use ISVD++ to pre-compute the model, the systems will face the same situation as using ISVD++ to update model with a huge updating dataset. So in order to get advantage of live update from incremental SVD++ and recommendations for new data we propose a system where both SVD++ and Incremental SVD++ must be implemented.

Despite the fact that Incremental SVD++ was researched for scaled commercial systems, the project was not production ready. Every time we want to predict a rating of a user to an items we need to reload the whole trained model into memory.

In this project we computed and stored the trained model into our data structure matrices - sparse matrix and dense matrix - which the size of the matrices were defined before hand by processing the input data such as the number of items, the number of users and the number of implicit feedback features for each user. These matrices are isolated within the input data, new users and items cannot be integrated into the model. So each time testing with new user or new item, we must initialize their factors at the time to pre-compute model.

Our model data is first stored in matrices and then writes to binary files. Next time we want to update our model or suggest items to someone, we have to load the model from files to our data structure and store it in the memory. The problem for scale is that, when the files size exceed the memory size, the memory will be out of ram and crash the application.

On the other hand, our Incremental SVD++ algorithm was fast because it only rebuild a portion of the trained model to predict items more up to date. But with the currently save-load model mechanism, time and resource was wasted

to load and save other unnecessary data during the process.

To be able to scale and using Incremental SVD++ more effectively, we suggest re-structuring data and storing SVD++ trained model into a database management system such as relational database like MySQL or Microsoft SQL Server. By using these relational database management system (RDBMS) we can take advantage of the built-in scalability options and responsive support from its' huge communities. We may also reduce a lot of load time since only relevant data was load.

Propose database structure:

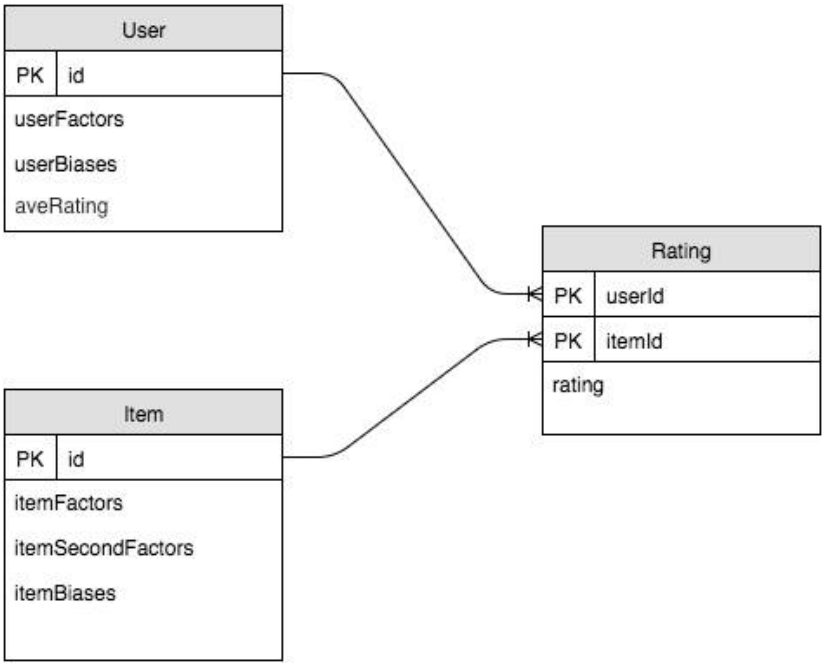

FIGURE 5.1
DATABASE STRUCTURE

Last but not least, we will welcome anyone caring about this thesis to give ideas or join us and develop Incremental SVD++. Please contact us at: cdanh@apcs.vn (Dung-Anh CAO) or hpanh@apcs.vn (Phuong-Anh HOANG) for more information.

Bibliography

[1] P. Lops, M. De Gemmis, and G. Semeraro, "Content-based recommender systems: State of the art and trends," in *Recommender systems handbook*, pp. 73–105, Springer, 2011.

[2] W. Hill, L. Stead, M. Rosenstein, and G. Furnas, "Recommending and evaluating choices in a virtual community of use," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 194–201, ACM Press/Addison-Wesley Publishing Co., 1995.

[3] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "Grouplens: an open architecture for collaborative filtering of netnews," in *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pp. 175–186, ACM, 1994.

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

[5] M. Chevalier, *Collaborative and Social Information Retrieval and Access: Techniques for Improved User Modeling: Techniques for Improved User Modeling*. IGI Global, 2009.

[6] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *IEEE transactions on knowledge and data engineering*, vol. 17, no. 6, pp. 734–749, 2005.

[7] M. Balabanović and Y. Shoham, "Fab: content-based, collaborative recommendation," *Communications of the ACM*, vol. 40, no. 3, pp. 66–72, 1997.

[8] 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*, pp. 426–434, ACM, 2008.

[9] T. Tran and R. Cohen, "Hybrid recommender systems for electronic commerce," in *Proc. Knowledge-Based Electronic Markets, Papers from the AAAI Workshop, Technical Report WS-00-04, AAAI Press*, 2000.

[10] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Application of dimensionality reduction in recommender system-a case study," tech. rep., DTIC Document, 2000.

[11] S. Funk, "Netflix update: Try this at home," 2006.

[12] G. Linden, B. Smith, and J. York, "Amazon. com recommendations: Item-to-item collaborative filtering," *IEEE Internet computing*, vol. 7, no. 1, pp. 76–80, 2003.

[13] P. Resnick and H. R. Varian, "Recommender systems," *Communications of the ACM*, vol. 40, no. 3, pp. 56–58, 1997.

[14] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the 10th international conference on World Wide Web*, pp. 285–295, ACM, 2001.

[15] R. Salakhutdinov, A. Mnih, and G. Hinton, "Restricted boltzmann machines for collaborative filtering," in *Proceedings of the 24th international conference on Machine learning*, pp. 791–798, ACM, 2007.

[16] Y. Koren, R. Bell, C. Volinsky, *et al.*, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[17] R. Salakhutdinov and A. Mnih, "Bayesian probabilistic matrix factorization using markov chain monte carlo," in *Proceedings of the 25th international conference on Machine learning*, pp. 880–887, ACM, 2008.

[18] Y. Zhang, B. Cao, and D.-Y. Yeung, "Multi-domain collaborative filtering," *arXiv preprint arXiv:1203.3535*, 2012.

[19] R. Jin, L. Si, and C. Zhai, "Preference-based graphic models for collaborative filtering," in *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*, pp. 329–336, Morgan Kaufmann Publishers Inc., 2002.

[20] S. Rendle and L. Schmidt-Thieme, "Pairwise interaction tensor factorization for personalized tag recommendation," in *Proceedings of the third ACM international conference on Web search and data mining*, pp. 81–90, ACM, 2010.

[21] H. Ma, D. Zhou, C. Liu, M. R. Lyu, and I. King, "Recommender systems with social regularization," in *Proceedings of the fourth ACM international conference on Web search and data mining*, pp. 287–296, ACM, 2011.

[22] R. Burke, "Hybrid web recommender systems," in *The adaptive web*, pp. 377–408, Springer, 2007.

[23] R. Burke, "Hybrid recommender systems: Survey and experiments," *User modeling and user-adapted interaction*, vol. 12, no. 4, pp. 331–370, 2002.

[24] D. W. Oard, J. Kim, *et al.*, "Implicit feedback for recommender systems," in *Proceedings of the AAAI workshop on recommender systems*, pp. 81–83, 1998.

Index