A Collaborative Filtering based Recommender System for Movies

by

Peixin Liu

A design project report

presented to the University of Guelph

in fulfilment of the

project requirement for the degree of

MEng.

in

Engineering Systems and Computing

Guelph, Ontario, Canada, 2016

Abstract

Recommender systems are used in various fields to personalize customers' on-line experience in order to provide better service for users. No matter what on-line marketplace, every user has a distinct set of behaviours. However, user's preferences usually follow some patterns that we can model to find similarities based on their historical actions. Two of the most ubiquitous types of recommender systems are content-based filtering and collaborative filtering recommender systems. In this work, we focus on implementing a collaborative filtering recommender system framework for movies based on user's previous ratings of different movies. Similarity and decomposition methods are performed in order to quantify the user's similarity metric. As well, the root mean squired error (RMSE) is used to evaluate the success of prediction against test data. Results obtained indicate that the model-based filtering approach with matrix decomposition performs better than memory-based filtering approach.

Acknowledgements

First of all, I would like to thank my advisor Dr. Shawki. He gave me much help during my study in my Master program and gave me many good suggestions on my final project.

Second, much appreciation and thanks goes to the professor Dr. Graham Taylor and other professors that gave me lectures in last two years in the courses I took.

Last, much appreciation to my mother that gave me much support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Recommender system are a subclass of information filtering frameworks that are capable of predicting the user preference of books, movies, research articles and many others items in general. Today, automated recommender systems are widely used to address personalized recommendations for a wealth of different activities and items. Most notably, web services like on-line music or video content providers generate user-specific recommendations that satisfy their individual interests.

1.1 Motivation

Recommender systems are a useful approach to help users discover items they might not have found by themselves. Especially when facing overwhelming number of choices, there is need to filter, prioritize and efficiently deliver relevant information in order to alleviate the problem of information overload, which has created a potential problem to users. Recommender systems try to capture the patterns from huge number of records to help predict what user might like.

1.2 Thesis Statement

In this project, a recommender system framework is proposed and implemented for movies based on user's previous ratings as shown in Figure 1.1. We demonstrate that collaborative filtering can be successfully employed to generate user-specific recommendations of movies based on historical data. In addition, we implement our framework with two different collaborative filtering approaches (memory-based and model-based) and evaluate and compare both approaches. The memory-based collaborative filtering includes two approaches: user-user based filtering and item-item based filtering. User-user filtering calculates the similarity between two user vectors based on their rating values. Item-item filtering calculates the similarity between two item vectors based on their rating values. The model-based collaborative filtering approach is based on matrix factorization. We choose singular value decomposition (SVD) to implement our model based collaborative filtering. The evaluation measure we choose to assess these approaches is based on root mean squared error (RMSE). We demonstrate that SVD gives better prediction accuracy over content-based approaches.

1.3 Summary

The remainder of this thesis is organized in five chapters. Chapter 2 introduces necessary background on the topic of recommender systems. Literature review

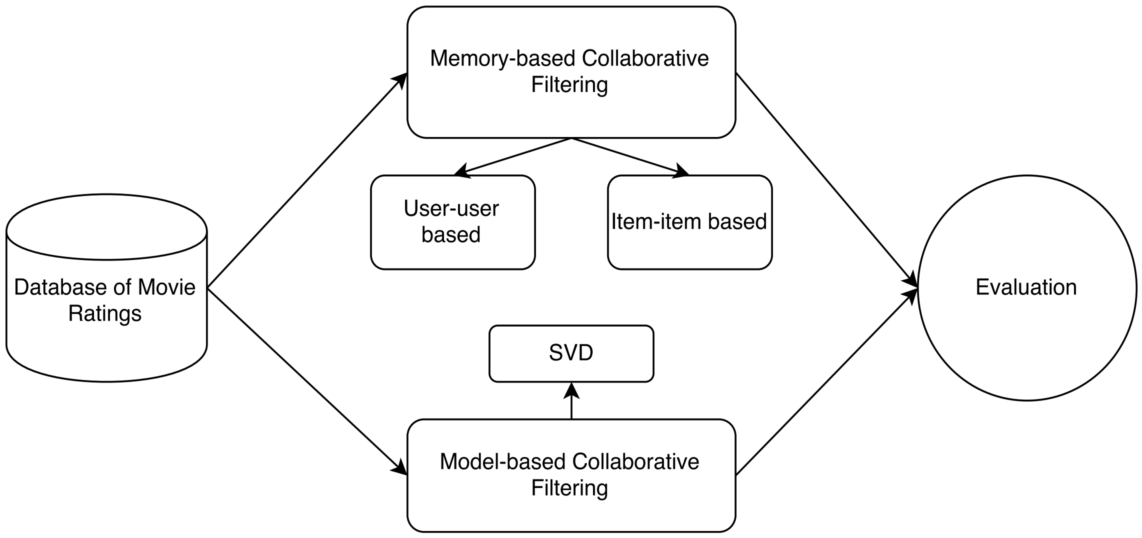
Memory-based Collaborative Filtering
User-user based
Item-item based
Database of Movie Ratings
Evaluation
SVD
Model-based Collaborative Filtering

Figure 1.1: Recommender System Framework

covering previous work is introduced in chapter 3. The main methodology and details of the implementation are introduced in chapter 4. Results based on the proposed framework are presented in chapter 5. The thesis is concluded in chapter 6 along with possible future work.

Chapter 2

Background

In this chapter, essential background material will be presented. This includes an overview of recommender systems, different classes of recommender systems and the evaluation method for these systems. The main objective of this chapter is to help the readers follow the concepts introduced in this thesis.

2.1 Introduction

A recommender system is a type of information system that is used for predicting or recommending items or activities for users throughout many fields of interest. Recommender systems are widely used in different fields including social networks, financial platforms, on-line shopping, business intelligence, insurance policy advising, on-line audio and video content provider, location-based services. Recommender systems are influencing how people find products, activities, information, and even new friends. Recommender systems attempt to learn from people's behaviour to establish patterns of user preferences based on their historical experiences and other people's similar choices.

2.2 Classification of Recommender System

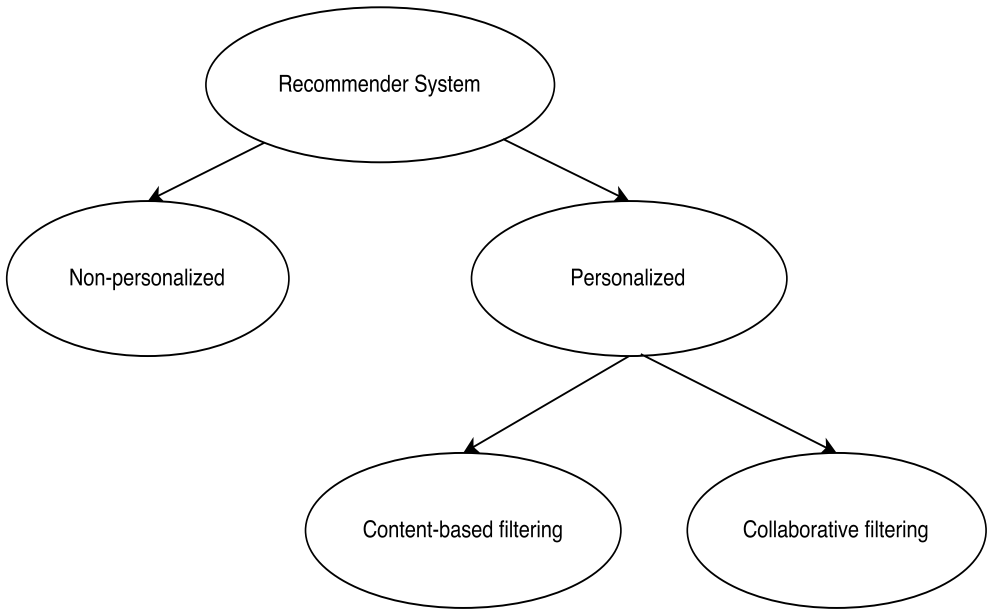

Figure 2.1: Classification of Recommender System

Typically there are different types of recommender systems as shown in Figure 2.1. One category is non-personalized recommender systems implemented by non-personalized summary statistics [1]. A second category is personalized recommender systems, which are implemented with either content-based filtering(CBF) or collaborative filtering(CF).

2.2.1 Non-personalized Summary Statistics

A non-personalized summary statistics recommender system is based on the statistics from public users activities or from external community data like trending popular items or best sellers. This data is usually created by public users previous ratings. To store or represent this data, we could easily use a matrix to show the ratings by different people as shown in Table 2.1. We can easily derive rating summary statistics from different users to different items. From this we can calculate the top rated items and recommend these popular items to users who haven't seen or experienced these items. This technique can solve the cold start problem [2], particularly when new users don't have enough historical data for the system to predict personalized recommendations. However, it is not suitable for recommending personalized results to established users who already have enough historical data.

	item1	item2	item3	...	itemN
user1	3	2	3	...	0
user2	0	5	4	...	2
user3	3	4	2	...	1
user4	1	2	4	...	5

Table 2.1: Non-Personalized Summary Statistics

2.2.2 Content-based Filtering

Content-based recommender systems identify items that users might prefer based on properties of the items or preferences of users. This method recommends items that are similar to the items someone earlier preferred. Therefore, we need to compare many items against items that have previously been rated by users to find the best matching results. In order to compare items, we need to establish some ways to extract common features among different items that can be compared. There are some methods mentioned below to illustrate how to discover features from data. Once features are identified, specific vectors for items and users are built with the same features to match users and predicted items to recommend.

2.2.2.1 Discovering Features from Item Properties

In content-based filtering, all the items are represented as records containing their features or characteristics. For some items in our daily life, it is easy to identify the important features, such as movies, music, books or cars. For example, for a certain movie, there are several important features including the director, cast, producer, genre and ratings that many patrons consider.

2.2.2.2 Discovering Features from Document

There are different kinds of documents that can be used in recommender system such as news articles, blogs and web pages. For example, we can recommend similar news articles to users based on content he or she read in the past. For these documents, we can measure features such as occurrence of the words where two documents with many common words would rank higher based on the number of common words or the frequency of their occurrence. This is commonly known as frequency inverse document frequency(TF.IDF) [3]. For all documents, we keep the set of words to search for exactly the same occurrence. Then for each set of high TF.IDF words, we represent it as a vector. The vector has binary element for each word in the set, where 1 means the word is common and 0 means the word is not common to the two documents. Comparing two vectors, there are only a limited number of words within two sets that will be used to compute the distance between vectors.

2.2.2.3 Representing Items and Users with Vectors

The main process for content-based filtering is to create an item vector containing feature values and a user vector containing preferences of users. In order to produce these vectors to represent items and users, we need to generate vector values from the original features. Usually, there are two kinds of values employed by these vectors: binary and positive integers. For binary elements, "1" indicates a feature is present and "0" means it is absent. For example, for the documents words, we can use binary elements to represent if it contains a certain word or not. For integer elements, we typically use it to represent a score or rating. For example, when recommending movies based on user's ratings of old movies, vectors containing rating values to calculate the similarity are needed.

2.2.2.4 Recommending Items to Users Based on Content

After building vectors for both users and items, we can use this information to find out recommendation results based on content. There are two primary methods to evaluate similarity.

Distance Calculation: Users' and Items' Vectors One approach is to use the user-item vectors to determine the user preference of an item by computing the distance between the user's vectors and item's vectors. The Cosine distance and Jaccard distance are commonly used to calculate distance between vectors. Specifically, for two vectors $A[A_1, A_2, A_3, \ldots, A_n]$ and $B[B_1, B_2, B_3, \ldots, B_n]$, the formula of Cosine distance is shown in Equation 2.1 which calculates the intersection between two vectors divided by the dot product of the two vectors. Similarly, the Jaccard distance is shown in Equation 2.2 which is calculated by the ratio of intersection and union of two sets.

$$\cos\theta = \frac{\sum_{i=1}^{n}(A_i \times B_i)}{\sqrt{\sum_{i=1}^{n}(A_i)^2} \times \sqrt{\sum_{i=1}^{n}(B_i)^2}} = \frac{A \times B}{\|A\| \times \|B\|} \tag{2.1}$$

$$J(A, B) = \frac{\|A \cap B\|}{\|A \cup B\|} \tag{2.2}$$

2.2.2.5 Content Based Filtering: Pros and Cons

The main strengths of content-based filtering is that they are easy to implement, and are suitable for document type files and easy to understand. In the software development industry, content-based recommender systems can be easily integrated with other systems such as search engines, indexing systems, and dictionary-based query systems. In addition, new unrated items can be recommended to users based on their feature values which can help to solve the cold-start problem. However, there are some drawbacks to content-based filtering. For example, if the content to be analyzed does not contain enough information to discriminate the items precisely, the recommendation will lack precision and accuracy. Also, user's rated items can be almost identical to each other based on the same properties and therefore it is difficult to recommend something new to each other.

2.2.3 Collaborative-based Filtering

Collaborative filtering is yet another popular approach that can be used to predict or recommend particular content to users based on similarity of preference among different users or items. Instead of using the features of items, the general process of collaborative filtering is to recommend new items to the users based on different user's ratings of items. There are different implementations of collaborative based filtering approaches that will be discussed next.

2.2.3.1 Memory-based Collaborative Filtering

In this method, instead of contriving a profile vector of users, they are represented them by their rows in a utility matrix (as shown in Table 2.2). The main approach of this method is to calculate the difference between rows to measure the similarity between users and recommend new items. For example, in Table 2.2, we find user1 and user3 are very similar in their ratings and therefore we should recommend item3 to user1 since no rating is registered by other users and user3 ranked item3 positive rating (assume a rating of 5 is the best score). Also, we can transpose this utility matrix to find similarity among different items. The methods we usually use to calculate the similarity between different rows is jaccard distance and cosine distance, which were introduced earlier by Equation 2.1 and 2.2 respectively.

	Item1	Item2	Item3	Item4
User1	5	3	N/A	0
User2	2	4	1	0
User3	4	2.5	4	1
User4	4	5	N/A	N/A

Table 2.2: Utility Matrix Data

2.2.3.2 Model-based Collaborative Filtering

This approach is more complicated than the memory-based filtering since a model needs to be built prior to applying it for recommendations on real data. In order to build a model, we have to find the pattern from the dataset and save this pattern model for the future use. Model-based filtering is mainly performed by matrix factorization as an unsupervised learning method for latent variable decomposition and dimensionality reduction. The main step and task of this approach is to obtain the potential preference of users and the potential features of items from the dataset. The system then seeks to recommend results or predict ratings based on the matrix factorization. The important advantage of this approach is that it can avoid the matrix sparsity problem by reducing the matrix to a lower-rank matrix, especially when the matrix data is highly dimensional. Singular value decomposition is one common approach to address matrix factorization for recommender systems as will be explained next.

Singular Value Decomposition Singular value decomposition(SVD) is a common approach for matrix factorization or decomposition (see Figure 2.2).

The general equation of SVD is shown by Equation 2.3, where A is the given

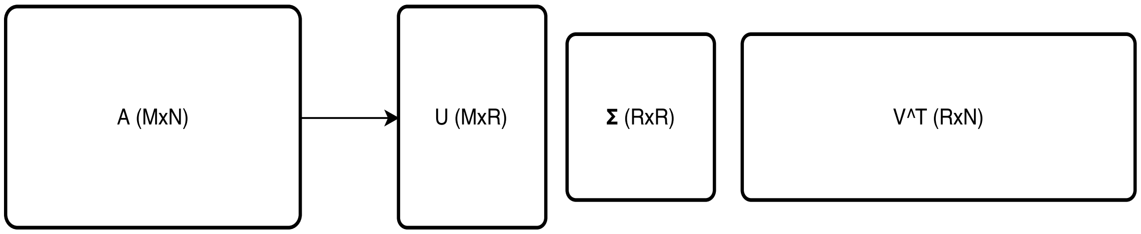

Figure 2.2: Singular Value Decomposition(SVD)

$M \times N$ matrix, U is an orthogonal $M \times R$ matrix, $\sum$ is a diagonal $R \times R$ matrix with non-negative real numbers on the diagonal positions, V^T is an orthogonal $R \times N$ matrix. Matrix A can be factorized into U, $\sum$ and V^T.

$$A = U \sum V^T \tag{2.3}$$

Matrix U represents the hidden features corresponding to users and matrix V^T stands for the hidden features to the items. After this factorization is completed, results can be recommended and predicted by calculating the dot product of U, $\sum$ and V^T.

2.2.3.3 Collaborative Filtering: Pros and Cons

Collaborative filtering is the process of establishing patterns among users or items. Collaborative filtering requires users to express opinions on items. Hosts collect opinions and recommend items based on the similarities of user's opinions. The principle advantage of collaborative filtering is that it does not require content information of items or additional user information. In addition, collaborative filtering can suggest serendipitous items by observing like-minded user's behaviour. However, a disadvantage of collaborative filtering is the cold-start problem for new or infrequent users. Without available rating data, or with only a little data, collaborative filtering can not recommend items to users. Further more, collaborative filtering is not content-aware even though some side information corresponding to users or items is available.

2.3 Evaluation Criteria

The evaluation criteria of any recommender system is an important task to assess the performance of algorithms employed. The commonly used evaluation methods are Mean Squared Error (MSE) and Root Mean Squared error (RMSE) which are illustrated by Equation 2.4 and 2.5 respectively. Typically, we split the dataset into two parts: training data and test data. After our framework establishes the recommendation results, we can compare these results with the test data to see how accurate the predictions are.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(A_i - B_i)^2 \tag{2.4}$$

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(A_i - B_i)^2}{n}} \tag{2.5}$$

2.4 Summary

In this chapter, essential background on recommender systems was presented along with applications. The different means of implementing recommender system was introduced. The personalized recommender system can be implemented using either a content-based filtering or collaborative filtering. The next chapter will introduce a literature review on recommender systems.

Chapter 3

Literature Survey

In this Chapter, we review literature and related works on the topic of recommender system. First, we introduce some data pre-processing approaches and validation methods. Next, we review previous research on content-based filtering recommender system and related work on collaborative filtering recommender system.

3.1 Preparation and Data Validation

G Shani et al. (2011) [4] introduced three evaluation metrics in their experiments: offline experiments, user studies and on-line evaluation. The main advantage of offline experiments is cost saving of resources. Instead of requiring many participants to contribute to the experiments directly, collecting data and performing simulation before inviting users to filter out useless candidates. With this pre-processing, unnecessary attempts can be dramatically reduced in the experiment. In [5], Erhard Rahm et al. (2000) performed a study on the data quality problem. They discussed some common approaches to handle data cleaning in two categories: single-source and multi-source problems. The single-source problems includes: illegal values, record violation, duplicated records, missing values and so on. The multi-source problems are more complicated. When several individual sources are combined together, each source will be required to be well-formatted. In order to address this problem, several methods such as definition of transformation work-flow and mapping rules are used.

3.2 Content-based Filtering

Content-based filtering recommender system has been widely used in behavior profiling. Many applications have already been used in the media content recommendation area. In [6], James Davidson et al. (2010) represented some features of their recommender system for YouTube video recommendation. They attempted to find the union of the history data among users and generate candidate recommendations based on the relations between users and their corresponding history data. In [7], Jayshri M et al. (2014) attempted to implement a recommender system to connect healthy food information to the food-order recommendations. Specifically, they built a real-time web service for the customers who visit restaurants. However, their system launched some aspects such as functions to collect and analyze more features and users data. When the location of restaurants were recorded this provide more accurate recommendations for customers based on locations. Also, they performed their evaluation approach by live A/B testing the two incoming data sources. In [8], Hung-Chen et al. (2001) extracted features from audio files to build a music recommender system. Traditional techniques and statistical methods were applied to figure out the target results based on end user's history and preferences among various songs. The authors used 6-dimensional classifiers to cluster and categorize music objects. Next, content-based and collaborative-based approaches were applied to group the music objects and user objects based on their relationships. The results show that these methods are effective for recommendation based on musical feature extraction.

In addition, the infrastructure of these content-based filtering recommender system has been developed. Riccardo et al. (2011) [9] proposed recommender system for the IPTV users. The advantages of their recommendation system was in the architecture, which used layers to separate independent functions, allowing easy deployment and expansion. Their recommender system was divided into three parts: data collection, recommendation generator and recommendation service. For the data collection part, their source data is primarily from Electronic Program Guide (EPG), content of video stream and behaviours of end users. All data was collected into the RDBMS and separated into two categories: item-content matrix (ICM) and user-rating matrix (URM). Their recommendation generation process is based on two approaches: batch process and real-time process. Another advantage of their system in batch processing which has the ability to generate models based on incremental rating data from end users after peak times. This approach is widely used by many Internet companies since the large size of incremental data is very time consuming.

In news media content services, content-based recommendation has been applied to push personalized news content to users based on their previous reading history. Lei Li et al. (2011) [10] improved the original news-article recommender system with a two-stage personalized recommendation approach. From their study, they discovered that there were potential connections among different article objects. In particular, the similarity and content pattern between different news articles was used for increasing accuracy of recommendations. In [11], Diandra et al. (2013) built a recommender system for news content to the end user based on their click behaviour on websites. With this collected data, their system could predict the type of articles users might be interested in. They applied a three-level approach, considering different aspects in their system to calculate the recommendation results, which improved upon a traditional one-way recommendation.

Content-based recommender system also can be used for recommending scientific software to end users. Naren et al. (1997) [12] built a PYTHIA system that implemented interfaces with GAMS mathematical software ontology. Their approach to implement the recommender system was based on an automatic generator. The main driver of this system is the knowledge engine sub-system that is used for generating new rules based on prior knowledge gained. The rules created were stored in a database for usage of recommendation. The main advantage of their recommendation system is that it directly connects software ontology to the end user to the software they desire.

Location based recommendation service is frequently used in many applications. Huiji Gao et al. (2013) [13] proposed a novel approach that depends on temporary information of user's locations based activities to improve their location-based recommender system. Specifically, the authors tried to make the analysis based on the users check-in data hourly which dramatically improved the accuracy of results for each user. Results obtained indicate that this approach is more effective for location-based recommender system. In [14], Erhard Rahm et al. (2000) designed a recommender system based on a large dataset which contains more than 2 years of GPS history data of many unique users. With such a large dataset, constructing recommendations of new places for users enables achieving accurate results. Authors attempted to connect the GPS data with the user's activities by regional area, and predict the possible destinations to users based on their previous locations. This technique can be used in travel recommendations and other similar scenarios. Thienne Johnson et al. (2014) [7] attempted to implement a recommender system to connect healthy food information to the food-order recommendations. Specifically, they built a real-time web service for the customers who visit restaurants. However, their system lacked some aspects such as functions to collect and analyze more features and users data. When the location of restaurants were recorded, this provided more accurate recommendations for customers based on locations. In [15], Damianos Gavalas and Michael Kenteris (2011) attempted to further improve the existing recommender system for tourist guides. Their proposed system contains some vital information such as location, time and weather. They used a wireless sensor network and points of interest from end users to increase accuracy of the location-based recommendation results. In [16], Betim Berjani and Thorsten Strufe (2011) implemented a recommender system focusing on the point object in the location-based on-line social network services. Their system dramatically offset the drawbacks of existing recommender systems had by providing end users recommendation results based on their recommendation scheme.

Some other algorithms also can be integrated with content-based filtering to improve the performance of recommendations. Michael J and Daniel (2007) [17] highlighted several algorithms that were employed in their proposed content-based recommender system. The most notable techniques were decision trees and rule induction. Decision Trees are used to partition the collection of objects into different blocks depending on certain features. This method seems to be more suitable for processing datasets with a small number of features, since a large number of attributes may demand large computational costs. The rule induction algorithm employed was ripper, which is similar to the decision tree approach. These two methods are suitable for some scenarios of content-based recommendation system. Moon-Hee et al. (2007) [18] introduced a Bayesian network recommender system based on users physical locations and corresponding information. Specifically, they applied a parameter learning technique based on the collected dataset and expectation maximization (EM) algorithm. With these approaches, they generated an output containing the updated mixture parameters and a matrix with cluster membership probabilities. One advantage of their work is its ability to solve the problem of display-size limitation on mobile devices. In order to overcome this barrier, they implemented a map-based interface containing the results from recommendation system which was easy for end users to use on their mobile devices. In [19], Yi and Jonathan (2007) used Bayesian hierarchical models to offset the cold-start problem. They created recommendation results from other user's data for those who lack enough historical data. Also, in order to speed up the process, they applied a fast learning approach called modified EM, which dramatically improves the speed of analysis. Some machine learning approaches have been applied to implement recommender systems in recent years. In [20], Santosh (2015) discussed different machine learning algorithms that are suitable for building recommender system. Notably, the FISM algorithm can be used to generate Top-N items for users. In addition, they made a temporary data set and compact during the process which can dramatically increase the efficiency and accuracy.

3.3 Collaborative Filtering

Collaborative filtering recommender system usually implemented by factorization of rating matrix. In [21], Badrul et al. (2002) applied the Singular Value Decomposition Algorithm (SVD) to solve some current barriers such as enhancing accuracy and output speed. This method can handle dynamic databases or incremental datasets, making it suitable for the instant recommendation requirement for end users. Also, they used the folding-in method to improve the overall efficiency and accuracy. This method can be effectively applied to recommender systems, however it required considerable computing resources. Ruslan and Andriy (2011) [22] introduced the Probabilistic Matrix Factorization model (PMF) and two other derivative models used in recommending movies to users based on their user history data on Netflix. These models are suitable for large datasets. The constrained PMF model is a particularly good approach to make stronger predictions for users with minimal historical data.

Location based recommender system also can be implemented with collaborative filtering approach. In [23], Vincent et al. (2012) built a personalized recommendation system based on users locations and activities. They compared two traditional methods (collaborative location-activity filtering algorithm and personalized collaborative location-activity filtering algorithm) against a new method called ranking-based personalized collaborative location and activity filtering (RPCLAF) which is an extension of two former methods. Also, they applied different kinds of definitions to represent the location such as: GPS trajectory, stay point and stay region. With the new method and definitions of data, their system offset some missing entries in each users location-activity matrix. Kenneth et al. (2011) [24] applied a collaborative location recommendation framework (CLR) into their GPS location based recommender system. With this new approach, they can overcome the barriers that persist in the traditional approaches, namely the huge resource consumption in matrix clustering and the difficulty in generating identification of relevant locations from large set of matrix data. This approach is a divide and conquer approach breaking the large problem into smaller problems by clustering the similar objects together before calculation and prediction. Also, this method supports incremental datasets, which is suitable for on-line services.

In addition, some new methods have been employed to improve the collaborative filtering approach. In [25], Nan and Qiudan (2011) applied three new strategies to an existing approach to recommend personalized results to end users based on tag and time frame data. With the three new strategies (time weight, tag weight, time-tag weight), their system generated modified ratings according to the frequency of tag and time information. Based on the ratings, the system calculated the users similarity and patterns. The proposed method effectively generated personalized results and showed better performance over prior work. In [26], Abbinandan et al. (2007) represented details about their recommender system used for aggregating news for end users. One such method was the Min-Hash algorithm that could estimate similarity between two objects. In order to speed up the calculation process, pruning was used when performing recursion or iteration. Also they examined the locality-sensitive hashing (LSH) algorithm which is suitable to solve close-neighbour search data analysis. This method uses a multi-hashing approach to identify the similarity between an object and its neighbours. In addition, they implemented Map-Reduce to deploy these methods into clusters which is now commonly used in many areas of research in order to handle large dataset.

3.4 Summary

In this Chapter, relevant literature for two types of recommender system was discussed. Both the content-based filtering approach and the collaborative filtering approach possess advantages and drawbacks. Many variations and incremental improvements have been discussed. In Chapter 4, our proposed collaborative filtering recommender system based on movie ratings will be introduced.

Chapter 4

Methodology

In this Chapter we present the overall methodology used to design and implement a recommender system for movies. We use two collaborative filtering based approaches (Memory-based collaborative filtering and Model-based collaborative filtering) to predict rating values.

4.1 Overview

The main objective of the work proposed in this project is designing a movie recommender system and using the open dataset from MovieLens [27] for evaluation. The framework consists of two main components (i) Memory-based collaborative filtering approach and (ii) Model-based collaborative filtering. The framework was designed using the Python programming language. The framework run in a OSX 10.11 operating system and was evaluated on a laptop with Intel Core i5 processor.

4.2 Memory-based Filtering

A memory-based filtering approach can be implemented in two different ways: user-user filtering and item-item filtering. User-user filtering can produce results that model a user's recommendations based on the patterns of similar users. Item-item filtering recommends items selected or purchased by other users who have at least one item in common. In the following sections, we describe in detail the two approaches of memory-based filtering recommender system. The steps of the methodology are shown in Figure 4.1.

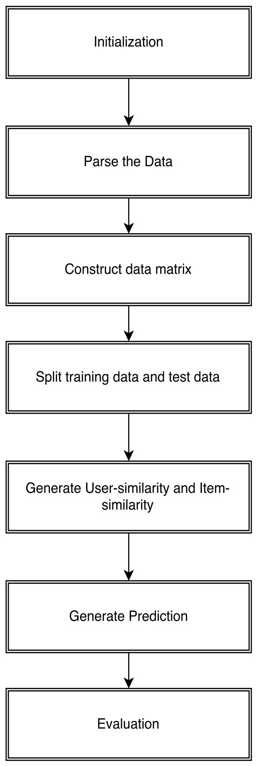

Figure 4.1: Overall Framework of Memory-based Recommendation System

4.2.1 Initialization

In the initialization step, the framework imports the necessary Python libraries. In this project, NumPy, Pandas and Scikit-learn were the main libraries used for calculation and data processing. NumPy is the fundamental package for scientific computing with Python. Pandas is another library performing easy-to-use data structure and data analysis. Scikit-learn is a Python library tool used for data-mining and analysis. Next, the framework loads the data for further analysis. Here the program loads the movie rating data from MovieLens. The data contains the rating values from different users on different movies. Table 4.1 shows a sample of the rating data from MovieLens. The first column shows the user ID, the second column introduces the movie ID, the third column highlights the rating and final column gives a timestamp.

User ID	Movie ID	Rating	Timestamp
1	1	5.0	1204927694
1	2	4.0	1204927438
1	3	3.0	1204927435
1	4	1.0	1204927444
2	1	4.0	1436165433
2	2	3.0	1436165496
2	3	3.0	1436165500
2	4	2.0	1436165499
3	1	5.0	920587155
3	2	4.0	920586920
3	3	3.0	920586945
3	4	1.0	920586155

Table 4.1: The Rating Data from MovieLens

	movie1	movie2	movie3	movie4
user1	5	4	3	1
user2	4	3	3	2
user3	5	4	3	1

Table 4.2: Re-arranged Rating (RR-Matrix)

4.2.2 The Training Matrix

Once the data is parsed and read from the dataset, the framework create a rating matrix that can be used for further processing. As shown in Table 4.1, the original data is neither grouped by users nor movies, and is unsuitable for analysis at this point. Therefore, the framework performs pre-processing to re-arrange the information as shown in Table 4.2 (RR-Matrix).

4.2.3 Similarity Matrix

In this step, the framework generates two similarity matrices based on RR-Matrix: a user-user collaborative filtering and an item-item collaborative filtering. The user-user filtering selects a single user and determines similar users based on their similarity ratings. The item-item filtering chooses a single item and determines similar items based on their similarity ratings. In order to calculate similarities between items or between users, we use the Cosine similarity (see Equation 2.1) or Jaccard similarity (see Equation 2.2) to calculate the distance between two user vectors or two item vectors. For example, if two user rating vectors are [5, 4, 3, 1] and [4, 3, 3, 2], the Cosine similarity between them is calculated by Equation 4.1 and the Jaccard similarity between them is calculated by Equation 4.2.

	user1	user2	user3
user1	1	0.97	1
user2	0.97	1	0.97
user3	1	0.97	1

Table 4.3: User-user Cosine Similarity Matrix

	movie1	movie2	movie3	movie4
movie1	1	0.99	0.99	0.90
movie2	0.99	1	0.99	0.89
movie3	0.99	0.99	1	0.94
movie4	0.9	0.89	0.94	1

Table 4.4: Item-item Cosine Similarity Matrix

$$\cos\theta = \frac{5\times 4 + 4\times 3 + 3\times 3 + 1\times 2}{\sqrt{5^2+4^2+3^2+1^2}\times\sqrt{4^2+3^2+3^2+2^2}} = 0.97 \tag{4.1}$$

$$J(\text{user}_1, \text{user}_2) = \frac{1}{4} = 0.25 \tag{4.2}$$

Accordingly, we generate two similarity matrices: a user-user similarity matrix based on the RR-Matrix and an item-item similarity matrix based on the transposed RR-Matrix. In this project, we use the pairwise-distances function from the sklearn library to calculate the similarity. The two Cosine similarity matrices are shown in Table 4.3 and 4.4 and the two Jaccard similarity matrices are shown in Table 4.5 and 4.6. For example, you can find the similarities between user 1 and another two users in the first line of Table 4.3 and 4.5.

The approach to build these similarity matrices is shown in Figure 4.2. The process is initiated by inputting RR-Matrix, similarity type and mode type. Depending

	user1	user2	user3
user1	1	0.25	1
user2	0.25	1	0.25
user3	1	0.25	1

Table 4.5: User-user Jaccard Similarity Matrix

	movie1	movie2	movie3	movie4
movie1	1	0	0	0
movie2	0	1	0.33	0
movie3	0	0.33	1	0
movie4	0	0	0	1

Table 4.6: Item-item Jaccard Similarity Matrix

on the mode type (item-item based or user-user based), we calculate the transpose of RR-Matrix and copy it back to RR-Matrix for item-item based approach. Then, we check the similarity type to decide which similarity method (Cosine distance or Jaccard distance) to be used. After that we can generate similarity matrices. The full algorithm of this approach is shown in appendix B.

4.2.4 Predicting Results

After generating the two similarity matrices, the framework produces prediction results based on the two similarity matrices. In order to produce the required rating based on the user-user similarity matrix, the formula shown in Equation 4.3 is used. As for the item-item similarity matrix, the required rating is calculated by the formula shown in Equation 4.4. For the user-user based filtering approach, a different formula that uses the relative differences in ratings is used instead of absolute

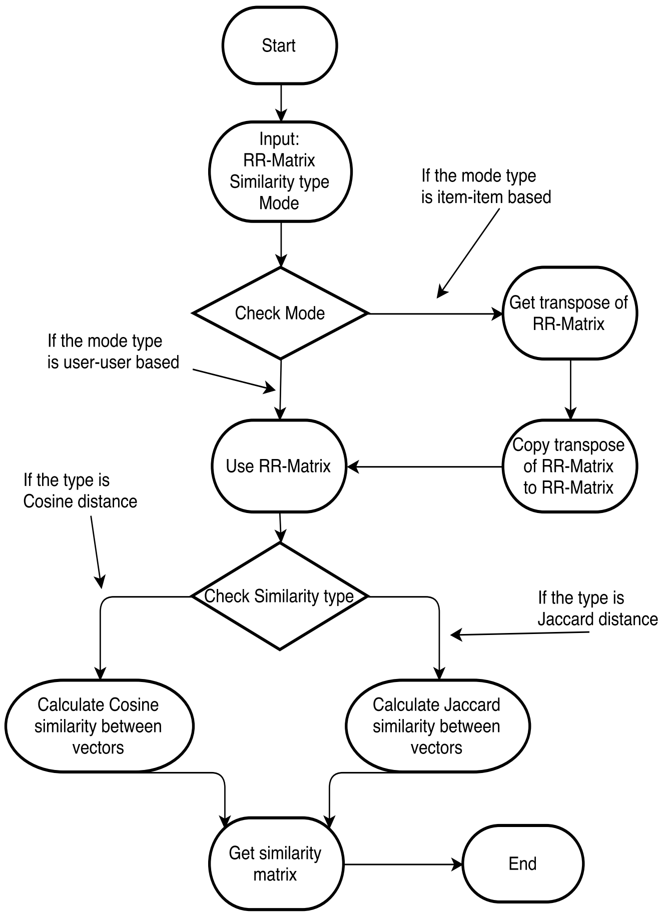

Figure 4.2: Approach to Build Similarity Matrix

rating values from users, (since different people have different rating systems).

$$\text{Rating(user-user)}_{\text{user}_m,\text{movie}_n} = \bar{\text{Rating}}_m + \frac{\sum_{user_a} \text{Sim}(\text{user}_m, \text{user}_a) \cdot (X_{a,n} - \bar{\text{Rating}}_a)}{\sum_{\text{user}_a} \|\text{Sim}(\text{user}_m, \text{user}_a)\|} \tag{4.3}$$

$$\text{Rating(item-item)}_{\text{user}_m,\text{movie}_n} = \frac{\sum_{\text{item}_a} \text{Sim}(\text{item}_n, \text{item}_a) \cdot (\text{Rating}_{m,a})}{\sum_{\text{item}_a} \|\text{Sim}(\text{item}_n, \text{item}_a)\|} \tag{4.4}$$

For example, user A gives 3 stars to his most liked movie and 2 stars to all the other movies. Now, suppose user B gives his most liked movies 5 stars and all other movies 4 stars. These two users should be similar despite the fact that their rating systems are very different. The prediction results are based on the user-user matrix and the item-item matrix. For example, the user-user based and item-item based predictions of rating from user 1 on movie 1 is calculated by Equation 4.5 and 4.6 respectively.

$$\text{Rating}_{\text{user}_1,\text{movie}_1} = \frac{5+4+3+1}{4} + \frac{1*(5-3.25)+0.97*(4-3)+1*(5-3.25)}{\sqrt{1^2}+\sqrt{0.97^2}+\sqrt{1^2}} = 4.75 \tag{4.5}$$

$$\text{Rating}_{\text{user}_1,\text{movie}_1} = \frac{1*5+0.99*4+0.99*3+0.90*1}{\sqrt{2.22^2}+\sqrt{3.69^2}+\sqrt{5.06^2}+\sqrt{9.54^2}} = 3.3 \tag{4.6}$$

The prediction matrices based on Cosine similarity are shown in Table 4.7 and 4.8 respectively and the prediction matrices based on Jaccard similarity are shown in Table 4.9 and 4.10 respectively. Among these tables, you can see the prediction rating from each user on each movie.

The approach to build these prediction matrix is shown in Figure 4.3. The

	movie1	movie2	movie3	movie4
user1	4.75	3.75	3.08	1.41
user2	4.5	3.5	2.83	1.17
user3	4.75	3.75	3.08	1.41

Table 4.7: User-user Cosine Similarity based Prediction

	movie1	movie2	movie3	movie
user1	3.31	3.31	3.28	3.19
user2	3.02	3.02	3.01	2.97
user3	3.31	3.31	3.28	3.19

Table 4.8: Item-item Cosine Similarity based Prediction

	movie1	movie2	movie3	movie4
user1	4.92	3.92	3.03	1.14
user2	4.25	3.25	2.92	1.58
user3	4.92	3.92	3.03	1.14

Table 4.9: User-user Jaccard Similarity based Prediction

	movie1	movie2	movie3	movie4
user1	5	3.75	3.25	1
user2	4	3	3	2
user3	5	3.75	3.25	1

Table 4.10: Item-item Jaccard Similarity based Prediction

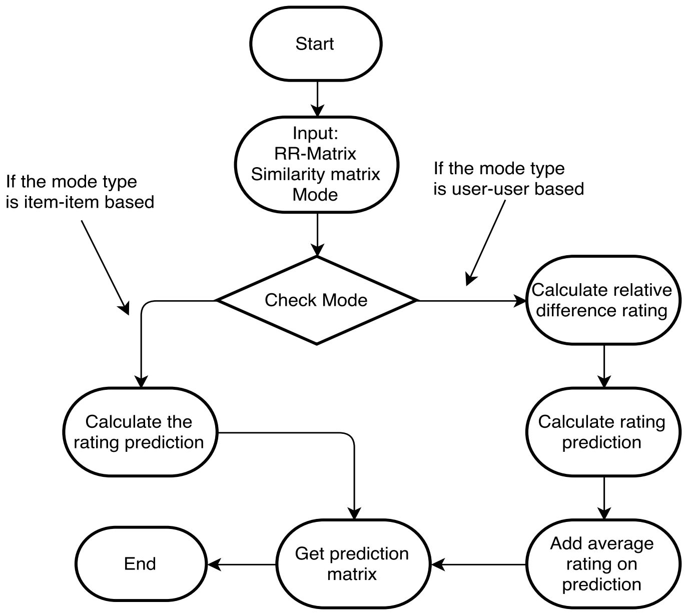
Start
Input:
RR-Matrix
Similarity matrix
Mode
If the mode type is item-item based
If the mode type is user-user based
Check Mode
Calculate relative difference rating
Calculate the rating prediction
Calculate rating prediction
End
Get prediction matrix
Add average rating on prediction

Figure 4.3: Approach to Build Prediction Matrix

process is initiated by inputting RR-Matrix, similarity matrix and mode type. Depending on the mode type (item-item based or user-user based), it followed by different processes. For item-item based approach, we can directly calculate the rating predictions. For user-user based approach, we have to calculate the relative rating difference for each rating of RR-Matrix followed by calculating rating predictions. Next, we need to add each users' average rating value back to each corresponding prediction rating value. The full algorithm of this approach is shown in appendix B.

4.3 Model-based Filtering

Model-based filtering techniques are typically implemented by matrix factorization or decomposition, where the framework attempts to find latent features of items and latent preferences of users from the RR-Matrix (Figure 4.3). The technique then predicts the unknown ratings based on the extracted information. The matrix factorization method can solve sparsity problems more efficiently than memory-based collaborative filtering since it re-arranges RR-Matrix into a low-rank structure. The overall methodology for model-based filtering is shown in Figure 4.4, which is similar to the memory-based filtering. The only difference is that, we use singular value decomposition (SVD) to implement matrix factorization based on the RR-Matrix instead of building similarity matrices.

4.3.1 Singular Value Decomposition

Figure 2.2 and Equation 2.3 introduced in chapter 2 revealed that for any $M \times N$ re-arranged rating matrix (RR-Matrix), it can be factorized into three matrices(U, $\sum$ and V^T), where

- U is an $M \times R$ matrix containing all the singular vectors that represent the latent preferences of users
- V^T is an $R \times N$ matrix containing all the singular vectors that represent the latent features of movies
- $\sum$ is an $R \times R$ diagonal matrix with only non-negative real numbers on the diagonal positions

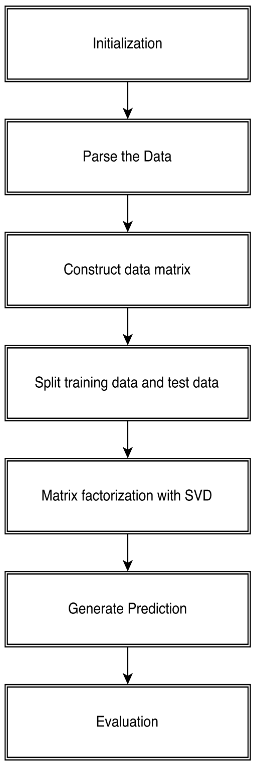

Figure 4.4: Overall Framework of Model-based Recommendation System

We can simply complete the SVD factorization with function svds (RR-Matrix, k) from Scipy library by assigning the parameter k a value less than the minimum value between M and N. For larger values of k, the system generates more latent information and requires more memory. So a trade off for larger or smaller values of k should be considered when performing SVD calculation.

Example of SVD Factorization For example, we start our factorization with a matrix A (as shown by Equation 2.4) and attempt to derive U, $\sum$ and V^T based on the A matrix shown by Equation 4.7. We can regard matrix A as a rating matrix which shows ratings from two users on three movies.

$$A = \begin{bmatrix} 3 & 1 & 1 \\ -1 & 3 & 1 \end{bmatrix} \tag{4.7}$$

The first step is to derive matrix U. We start with calculating of AA^T by first transposing matrix A as shown by Equation 4.8.

$$A^T = \begin{bmatrix} 3 & -1 \\ 1 & 3 \\ 1 & 1 \end{bmatrix} \tag{4.8}$$

The product matrix AA^T is shown by Equation 4.9.

$$AA^T = \begin{bmatrix} 3 & 1 & 1 \\ -1 & 3 & 1 \end{bmatrix} \begin{bmatrix} 3 & -1 \\ 1 & 3 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 11 & 1 \\ 1 & 11 \end{bmatrix} \tag{4.9}$$

Next, the eigenvalues and corresponding eigenvectors of AA^T are calculated as shown by Equation 4.10 and Equation 4.11.

$$A\vec{v} = \lambda\vec{v} \tag{4.10}$$

$$\begin{bmatrix} 11 & 1 \\ 1 & 11 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{4.11}$$

Based on Equation 4.11, two Equations 4.12 and 4.13 are produced respectively.

$$(11 - \lambda)x_1 + x_2 = 0 \tag{4.12}$$

$$x_1 + (11 - \lambda)x_2 = 0 \tag{4.13}$$

Next, we combine two Equations 4.12 and 4.13 into one Equation 4.14.

$$\begin{bmatrix} 11-\lambda & 1 \\ 1 & 11-\lambda \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \lambda \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{4.14}$$

Next, we create a matrix based on Equation 4.14 to solve λ by setting the determinant t of the coefficient matrix to 0 as shown by Equation 4.15.

$$\begin{bmatrix} (11-\lambda) & 1 \\ 1 & (11-\lambda) \end{bmatrix} = 0 \tag{4.15}$$

Now, we can solve for the two eigenvalues $\lambda = 12$ and $\lambda = 10$. Substituting these values back into Equations 4.12 and 4.13 we get Equation 4.16 and Equation 4.17. Combine them together we get the eigenvalue matrix as shown by Equation 4.18.

$$x_1 = x_2 \tag{4.16}$$

$$x_1 = -x_2 \tag{4.17}$$

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{4.18}$$

Next, we use the Gram-Schmidt orthonormalization process [28] to transform Equation 4.18 into an orthogonal matrix. Specifically, we normalize each column vector respectively to get normalized vector $\vec{v_1}$ (as shown by Equation 4.19) and vector $\vec{v_2}$ (as shown by Equation 4.20).

$$\vec{v_1} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} \tag{4.19}$$

$$\vec{v_2} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} \end{bmatrix} \tag{4.20}$$

Matrix U is obtained with these two normalized vectors as shown by Equation 4.21.

$$U = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix} \tag{4.21}$$

Next, matrix V (as shown by Equation 4.22) can be calculated similar to U. However, we use $A^T A$ instead of AA^T.

$$V = \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{30}} \\ \frac{2}{\sqrt{6}} & \frac{-1}{\sqrt{5}} & \frac{2}{\sqrt{30}} \\ \frac{1}{\sqrt{6}} & 0 & \frac{-5}{\sqrt{30}} \end{bmatrix} \tag{4.22}$$

The transpose of V is obtained as shown by Equation 4.23.

$$V^T = \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} & \frac{1}{\sqrt{6}} \\ \frac{2}{\sqrt{5}} & \frac{-1}{\sqrt{5}} & 0 \\ \frac{1}{\sqrt{30}} & \frac{2}{\sqrt{30}} & \frac{-5}{\sqrt{30}} \end{bmatrix} \tag{4.23}$$

Finally, the diagonal matrix $\sum$ needs to be calculated. In order to find out $\sum$, we create a new 2 x 3 matrix (Since U is 2 x 2 matrix and V^T is 3 x 3 matrix) and populate the diagonal positions with square roots of eigenvalues we got from Equation 4.15 ($\lambda = 12$ or $\lambda = 10$). The $\sum$ is shown by Equation 4.24.

$$\sum = \begin{bmatrix} \sqrt{12} & 0 & 0 \\ 0 & \sqrt{10} & 0 \end{bmatrix} \tag{4.24}$$

Equation 4.24 shows the final factorization of matrix A in terms of U, $\sum$ and V^T.

$$A = U \sum V^T = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} \sqrt{12} & 0 & 0 \\ 0 & \sqrt{10} & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} & \frac{1}{\sqrt{6}} \\ \frac{2}{\sqrt{5}} & \frac{-1}{\sqrt{5}} & 0 \\ \frac{1}{\sqrt{30}} & \frac{2}{\sqrt{30}} & \frac{-5}{\sqrt{30}} \end{bmatrix} \tag{4.25}$$

4.3.2 Predicting Results

Based on Matrix A shown previously in Equation 4.7, we can get three matrices U, $\sum$ and V^T through SVD as shown in Table 4.11, 4.12 and 4.13 respectively by performing SVD factorization on the matrix A. In our implementation, we use the SVD function library scipy, which can perform SVD factorization and generate U, $\sum$ and V^T. In order to use this function, we need to choose a parameter k (k is minimum value of M and N where A is M x N matrix) which indicates that how much information we want the factorization results include. In this example, we give k value 2 which is the largest value it can be assigned. The next step will be prediction based on the three matrices factorized from the SVD calculation. The framework will simply perform dot product of the matrices: U, $\sum$ and V^T to produce the prediction results as shown in Table 4.14.

0.7	0.7
0.7	-0.7

Table 4.11: Matrix U Generated with SVD

3.46	0	0
0	3.16	0

Table 4.12: Matrix $\sum$ Generated with SVD

0.41	0.82	0.41
0.9	-0.45	0
0.18	0.37	-0.91

Table 4.13: Matrix V^T Generated with SVD

	movie1	movie2	movie3
user1	3	1	1
user2	-1	3	1

Table 4.14: Prediction Matrix Generated by U, $\sum$ and V^T

The approach to perform SVD factorization is shown in Figure 4.5. The process is initiated by inputting RR-Matrix. Depending on obtaining U or V^T, we calculate AA^T and A^AT respectively. Next, we use the results from last step to find all the eigenvectors for U and V^T followed by generating two orthogonal matrices respectively. Then we can generate U and V^T by normalize the two matrices from last step. Based on U and V^T, we can find the corresponding eigenvalues to build the $\sum$ matrix. The full algorithm to perform SVD is shown in Appendix B.

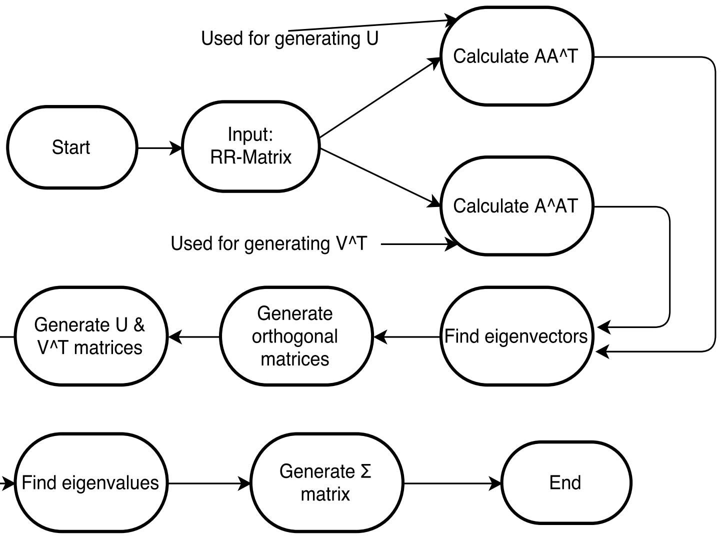

Figure 4.5: Approach to Perform Singular Value Decomposition (SVD)

4.4 Summary

In this chapter the proposed methodology based on two collaborative filtering approaches was presented. First a memory based filtering system was introduced followed by a model based filtering system. Both techniques have advantages and disadvantages. In the next chapter, results obtained based on these collaborative filtering approaches will be presented along with analysis of the results.

Chapter 5

Results

This chapter presents results based on the proposed framework introduced in chapter 4. The experimental setup will be introduced along with the data and benchmarks used. This is followed by observations on the results we got and corresponding analysis.

5.1 Experimental Setup

The proposed framework executes on a Mac OSX 10.11 operating system environment on a laptop with Core-i5 CPU. The Jupyter-notebook-4.2 is used as an editor based on Python-2.7. Several open source libraries including Numpy-1.11, Pandas-0.18, Scikit-learn-0.17 and Scipy-0.17 are used in this work.

5.2 Description of Data

The benchmark dataset used in this project is downloaded from MovieLens [27], which is a research site run by GroupLens Research at the University of Minnesota. MovieLens-100k dataset contains 100,000 ratings from 1000 users on 1700 different movies that were released in year 1998. Zheng et al. already applied CF-NADE method on the MovieLens-10M dataset and achived Root Mean Square Error of 0.772 which beats all the previous state-of-the-art methods by the year 2016 [29]. Table 5.1 shows the structure of this dataset which includes three columns of property.

userID	movieID	rating	timestamp
196	242	3	881250949
186	302	3	891717742
22	377	1	878887116
244	51	2	880606923
166	346	1	886397596
298	474	4	884182806
115	265	2	881171488
253	465	5	891628467
305	451	3	886324817
6	86	3	883603013
62	257	2	879372434
286	1014	5	879781125
200	222	5	876042340
210	40	3	891035994
224	29	3	888104457
303	785	3	879485318
...	...	...	...

Table 5.1: MovieLens-100k Dataset

5.3 Observations and Analysis

In this section, we attempt to use two collaborative filtering approaches (memory-based and model based) that were described in chapter 4 to predict the ratings for users based on the MovieLens-100k dataset followed by an evaluation and analysis of results. In our implementation, we use cross validation model from library "sk-learn" to splits the data into training and testing datasets with standard splits (80%/20%) of MovieLens dataset [27]. The training data is used for generating prediction results and the test data is used for evaluation.

After prediction matrices have been generated, we compares the difference in each user's rating values between the test data matrix and the prediction matrix generated by each approach. The Root Mean Squared Error (RMSE) is used to calculate the prediction errors. The reason we use RMSE as our evaluation method is that RMSE can compare each position of value between two datasets with same structure, which is convenient for us to perform the evaluation. The evaluation results for these approaches are shown in Table 5.2.

Approaches	RMSE
User-user Cosine similarity	2.910
Item-item Cosine similarity	3.129
User-user Jaccard similarity	2.904
Item-item Jaccard similarity	3.047
SVD	2.651

Table 5.2: Performance Measures

Based on the performance measures, the RMSE for user-user similarity based filtering and item-item similarity based filtering are almost the same for both cosine and jaccard similarity measures. This means these two similarity methods applied to the rating matrix result in very small differences in measuring similarity. However, the user-user similarity based filtering approach performs slightly better than item-item similarity based filtering. One possible reason is that we use relative difference ratings instead of absolute values when predicting results in the user-user based filtering approach. The model based collaborative filtering with SVD preforms best among these approaches as seem clearly from Table 5.2. This indicates that the SVD approach can produce matrices containing latent information from users or items from known ratings and these matrices can be used to predict unknown ratings well by computing their cartesian dot products.

Chapter 6

Conclusion and Future Work

In this project, a recommender system framework to predict ratings of movies for users based on their historical ratings was designed and implemented. Re-arranging the original rating matrix data and grouping data by different users. Next, two different approaches (memory based and model based collaborative filtering) were applied on the rating matrix to predict the rating results. In the memory based collaborative filtering approach, the user-user memory based similarity matrix was generated by the re-arranged rating matrix and the item-item memory based similarity matrix is generated by the transposed re-arranged rating matrix. Next, the two different similarity matrices were used to predict ratings for unseen movies. When performing user-user based prediction, we should note that different users have different rating systems. One user's lowest historical rating may be 2 while another's may be 7. Hence, we should use the relative difference between ratings instead of the absolute rating values to calculate the similarity measure. In the model based collaborative filtering approach, we use singular value decomposition(SVD) to achieve a matrix factorization to predict the user's rating results. The SVD factorizes the re-arranged rating matrix into three matrices that contain latent preferences of users and latent features of movies. We can use these singular values and singular vectors to predict the user's rating values by computing the dot product of these three matrices. The results suggest that the model based collaborative filtering with SVD approach performs better than memory based filtering when predicting user's rating values for movies.

Some possible future works on recommender systems can be summarized as follows:

1. Although the singular value decomposition method can outperforms the content-based filtering methods in predictive error, it still suffers from some problems such as overfitting with insufficient known entries, or large computational effort for larger datasets. In order to address these potential barriers, alternating least square(ALS) or stochastic gradient descent(SGD) could be applied to minimize computational load and improve predictive accuracy.

2. The recommender system framework built in this project can be applied on different datasets such as books, music and on-line shopping services.

3. A content based filtering approach could be a candidate for implementation and comparison with the current proposed collaborative filtering based approach.

Bibliography

[1] J Ben Schafer, Dan Frankowski, Jon Herlocker and Shilad Sen, "Collaborative filtering recommender systems", in *The adaptive web*, pp. 291–324, Springer, 2007.

[2] Xuan Nhat Lam, Thuc Vu, Trong Duc Le and Anh Duc Duong, "Addressing cold-start problem in recommendation systems", in *Proceedings of the 2nd international conference on Ubiquitous information management and communication*, pp. 208–211, 2008.

[3] Stephen Robertson, "Understanding inverse document frequency: on theoretical arguments for IDF", *Journal of documentation*, vol. 60, n. 5, pp. 503–520, 2004.

[4] G Shani and A Gunawardana, "Recommender Systems Handbook", 2011.

[5] Erhard Rahm and Hong Hai Do, "Data cleaning: Problems and current approaches", *IEEE Data Eng. Bull.*, vol. 23, n. 4, pp. 3–13, 2000.

[6] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston

et al., "The YouTube video recommendation system", in *Proceedings of the fourth ACM conference on Recommender systems*, pp. 293–296, 2010.

[7] Thienne Johnson, Jorge Vergara, Chelsea Doll, Madison Kramer, Gayathri Sundararaman, Harsha Rajendran, Alon Efrat and Melanie Hingle, "A Mobile Food Recommendation System Based on The Traffic Light Diet", *arXiv preprint arXiv:1409.0296*, 2014.

[8] Hung-Chen Chen and Arbee LP Chen, "A music recommendation system based on music data grouping and user interests", in *Proceedings of the tenth international conference on Information and knowledge management*, pp. 231–238, 2001.

[9] Riccardo Bambini, Paolo Cremonesi and Roberto Turrin, "A recommender system for an IPTV service provider: a real large-scale production environment", in *Recommender systems handbook*, pp. 299–331, Springer, 2011.

[10] Lei Li, Dingding Wang, Tao Li, Daniel Knox and Balaji Padmanabhan, "Scene: a scalable two-stage personalized news recommendation system", in *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pp. 125–134, 2011.

[11] Diandra Mayang Desyaputri, Alva Erwin, Maulahikmah Galinium and Didi Nugrahadi, "News recommendation in Indonesian language based on user click behavior", in *Information Technology and Electrical Engineering (ICITEE), 2013 International Conference on*, pp. 164–169, 2013.

[12] Naren Ramakrishnan, Elias N Houstis and John R Rice, "Recommender sys-

tems for problem solving environments", in *Technical Report WS-98-08 (Working Notes of the AAAI-98 Workshop on Recommender Systems)*, pp. 91–95, 1997.

[13] Huiji Gao, Jiliang Tang, Xia Hu and Huan Liu, "Exploring temporal effects for location recommendation on location-based social networks", in *Proceedings of the 7th ACM conference on Recommender systems*, pp. 93–100, 2013.

[14] Vincent W Zheng, Yu Zheng, Xing Xie and Qiang Yang, "Collaborative location and activity recommendations with gps history data", in *Proceedings of the 19th international conference on World wide web*, pp. 1029–1038, 2010.

[15] Damianos Gavalas and Michael Kenteris, "A web-based pervasive recommendation system for mobile tourist guides", *Personal and Ubiquitous Computing*, vol. 15, n. 7, pp. 759–770, 2011.

[16] Betim Berjani and Thorsten Strufe, "A recommendation system for spots in location-based online social networks", in *Proceedings of the 4th Workshop on Social Network Systems*, page 4, 2011.

[17] Michael J Pazzani and Daniel Billsus, "Content-based recommendation systems", in *The adaptive web*, pp. 325–341, Springer, 2007.

[18] Moon-Hee Park, Jin-Hyuk Hong and Sung-Bae Cho, "Location-based recommendation system using bayesian users preference model in mobile devices", in *Ubiquitous Intelligence and Computing*, pp. 1130–1139, Springer, 2007.

[19] Yi Zhang and Jonathan Koren, "Efficient bayesian hierarchical user modeling for recommendation system", in *Proceedings of the 30th annual international*

ACM SIGIR conference on Research and development in information retrieval, pp. 47–54, 2007.

[20] Santosh Kabbur, *Machine Learning Methods for Recommender Systems*, PhD thesis, University OF Minnesota, 2015.

[21] Badrul Sarwar, George Karypis, Joseph Konstan and John Riedl, "Incremental singular value decomposition algorithms for highly scalable recommender systems", in *Fifth International Conference on Computer and Information Science*, pp. 27–28, 2002.

[22] Ruslan Salakhutdinov and Andriy Mnih, "Probabilistic matrix factorization", 2011.

[23] Vincent W Zheng, Yu Zheng, Xing Xie and Qiang Yang, "Towards mobile intelligence: Learning from GPS history data for collaborative recommendation", *Artificial Intelligence*, vol. 184, pp. 17–37, 2012.

[24] Kenneth Wai-Ting Leung, Dik Lun Lee and Wang-Chien Lee, "CLR: a collaborative location recommendation framework based on co-clustering", in *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pp. 305–314, 2011.

[25] Nan Zheng and Qiudan Li, "A recommender system based on tag and time information for social tagging systems", *Expert Systems with Applications*, vol. 38, n. 4, pp. 4575–4587, 2011.

[26] Abhinandan S Das, Mayur Datar, Ashutosh Garg and Shyam Rajaram, "Google news personalization: scalable online collaborative filtering", in *Pro-*

ceedings of the 16th international conference on World Wide Web, pp. 271–280, 2007.

[27] "MovieLens", http://grouplens.org/datasets/movielens/100k/, Released: 1998.

[28] P Sam Johnson, "Gram-Schmidt Orthogonalization Process", 2014.

[29] Yin Zheng, Bangsheng Tang, Wenkui Ding and Hanning Zhou, "A Neural Autoregressive Approach to Collaborative Filtering", *arXiv preprint arXiv:1605.09477*, 2016.

Appendix A

Glossary

CBF	: Content-based Filtering
CF	: Collaborative Filtering
IF.IDF	: Frequency Inverse Document Frequency
RMSE	: Root Mean Squared Error
SVD	: Singular Value Decomposition

Appendix B

Algorithm

```
/* The Algorithm to Build Similarity Matrix*/
Time Complexity : $N^2 * M$ (where RR-Matrix is a N x M matrix)
Input: RR-Matrix, similarity-method-type, mode
Output: similarity (matrix)
1.   IF mode == ITEM-ITEM
2.       RR-Matrix = Transpose of RR-Matrix
3.   ENDIF
4.   Declare similarity[RR-Matrix.length][RR-Matrix.length]
5.   IF similarity-method-type == COSINE
6.       FOR i from 0 to RR-Matrix.length - 1
7.           Declare numerator, denominator
8.           FOR j from 0 to RR-Matrix.length - 1
9.               FOR k from 0 to RR-Matrix[0].length - 1
10.                  numerator += RR-Matrix[i][k] * RR-Matrix[j][k]
11.              ENDFOR
12.              Declare sum1, sum2
13.              FOR k from 0 to RR-Matrix[0].length - 1
14.                  sum1 += square of RR-Matrix[i][k]
15.                  sum2 += square of RR-Matrix[j][k]
16.              ENDFOR
```

Figure B.1: The Algorithm to Build Similarity Matrix

```
/* The Algorithm to Build Similarity Matrix*/
17.             denominator = (square root of sum1) + (square root of sum2)
18.             similarity[i][j] = numerator / denominator
19.         ENDFOR
20.     ENDFOR
21. ENDIF
22. ELSEIF similarity-method-type == JACCARD
23.     FOR i from 0 to RR-Matrix.length - 1
24.         Declare numerator, denominator
25.         Declare hashset
26.         FOR j from 0 to RR-Matrix.length - 1
27.             FOR k from 0 to RR-Matrix[0].length - 1
28.                 hashset.add(RR-Matrix[i][k])
29.                 hashset.add(RR-Matrix[j][k])
30.                 IF RR-Matrix[i][k] == RR-Matrix[j][k]
31.                     numerator++
32.                 ENDIF
33.             ENDFOR
34.         ENDFOR
35.         denominator = hashset.size
36.         similarity[i][j] = (denominator - numerator) / denominator
37.     ENDFOR
38. ENDIF
```

Figure B.2: The Algorithm to Build Similarity Matrix

```
/* The Algorithm to Build Prediction Matrix */
Time Complexity : $N^2 * K$ (N is length of RR-Matrix and K is length of Similarity-Matrix)
Input: Similarity-Matrix, RR-Matrix, mode
Output: prediction (matrix)
1.    IF mode == ITEM-ITEM
2.        RR-Matrix = Transpose of RR-Matrix
3.    ENDIF
4.    Declare prediction[RR-Matrix.length][RR-Matrix[0].length]
5.    IF mode == USER-USER
6.        Declare average[Similarity-Matrix.length]
7.        FOR i from 0 to Similarity-Matrix.length - 1
8.            Declare sum 9.          FOR k from 0 to Similarity-Matrix[0].length - 1
10.               sum += Similarity-Matrix[i][k]
11.           ENDFOR
12.           average[i] = sum divide by Similarity-Matrix[0].length
13.       ENDFOR 14.       For i from 0 to RR-Matrix.length - 1
15.           FOR j from 0 to RR-Matrix[0].length - 1
16.               Declare numerator, denominator
17.               FOR k from 0 to Similarity-Matrix[0].length - 1
18.                   Declare difference = RR-Matrix[k][j] - average[k]
19.                   numerator += (Similarity-Matrix[i][k] * difference)
20.                   denominator += square root of (square of Similarity-Matrix[i][k])
21.               ENDFOR
22.               prediction[i][j] = numerator / denominator
23.           ENDFOR
24.       ENDFOR
25.   ENDIF
26.   ELSEIF mode == ITEM-ITEM
27.       For i from 0 to RR-Matrix.length - 1
28.           FOR j from 0 to RR-Matrix[0].length - 1
29.               Declare numerator, denominator
30.               FOR k from 0 to Similarity-Matrix[0].length - 1
31.                   numerator += (Similarity-Matrix[i][k] * RR-Matrix[k][j])
32.                   denominator += square root of (square of Similarity-Matrix[i][k])
33.               ENDFOR
34.               prediction[i][j] = numerator / denominator
35.           ENDFOR
36.       ENDFOR
37.       prediction = Transpose of prediction
38.   ENDIF
39.   RETURN prediction
```

Figure B.3: The Algorithm to Build Prediction Matrix

```
/* The Algorithm to perform Singular Value Decomposition (SVD) */
Time Complexity: $O(min(M * N^2, M^2 * N))$ (where RR-Matrix is a N x M matrix)
Input: RR-Matrix
Output: U, S (stands for $\sum$), V (stands for $V^T$)
1.   Declare U = copy of RR-Matrix
2.   Declare m = RR-Matrix.row-dimension
3.   Declare n = RR-Matrix.column-dimension
4.   Declare e = [0.0]*n, S = [0.0]*n, V = []
4.   FOR i from 0 to n
5.       e[i] = 0.0, s = 0.0
6.       FOR j from i to m
7.           s += U[j][i] * U[j][i]
8.       ENDFOR
9.       Declare f = U[i][i]
10.      Declare h = f * g - s
11.      FOR j from i + 1 to n
12.          FOR k from i to m
13.              s += U[k][i] * U[k][j]
14.          ENDFOR
15.          f = s / h
16.          FOR k from i to m
17.              U[k][j] = U[k][j] + f * U[k][i]
18.      ENDFOR
19.      FOR j from i + 1 to n
20.          s += U[i][j] * U[i][j]
21.      ENDFOR
22.      f = U[i][i+1]
23.      h = f * g - s
24.      U[i][i+1] = f - g
25.      FOR j from i + 1 to n
26.          e[j] = U[i][j] / h
27.      ENDFOR
28.      FOR j from i + 1 to m
29.          s = 0.0
```

Figure B.4: The Algorithm to perform Singular Value Decomposition (SVD)

```
/* The Algorithm to perform Singular Value Decomposition (SVD) */
            FOR k from i + 1 to n
                s += (U[j][k] * U[i][k])
            ENDFOR
            FOR k from i + 1 to m
                U[j][k] = U[j][k] + (s * e[k])
            ENDFOR
        ENDFOR
    ENDFOR
    FOR i from 0 to n - 1
        Declare h = g * U[i][i+1]
        FOR j from i + 1 to n
            V[j][i] = U[i][j] / h
        ENDFOR
        FOR j from i + 1 to n
            FOR k from i + 1 to n
                s += (U[i][k] * V[k][j])
            ENDFOR
            FOR k from i + 1 to n
                V[k][j] += (s * V[k][i])
            ENDFOR
        ENDFOR
        FOR j from i + 1 to n
            V[i][j] = 0.0
            V[j][i] = 0.0
        ENDFOR
        V[i][i] = 1.0
        g = e[i]
    ENDFOR
    FOR i from 0 to n - 1
        g = S[i]
        FOR j from i + 1 to n
```

Figure B.5: The Algorithm to perform Singular Value Decomposition (SVD)

```
/* The Algorithm to perform Singular Value Decomposition (SVD) */
            U[i][j] = 0.0
        ENDFOR
        IF g != 0.0
            h = U[i][i] * g
            FOR j from i to n
                s=0.0
                For k from i + 1 to m
                    s += (U[k][i] * U[k][j])
                ENDFOR
                FOR k from i to m
                    U[k][j] += (f * U[k][i])
                ENDFOR
            ENDFOR
        ENDIF
        ELSEIF
            FOR j from i to m
                U[j][i] = 0.0
            ENDFOR
        ENDIF
        U[i][i] += 1.0
    ENDFOR
    eps = 0.0
    FOR k from 0 to n - 1
        x = S[l]
        y = S[k-1]
        g = e[k-1]
        h = e[k]
        z = S[k]
        f = ((y-z) * (y + z ) + (g - h) * (g + h)) / (2.0 * h * y)
        IF f ¡ 0
```

Figure B.6: The Algorithm to perform Singular Value Decomposition (SVD)

```
/* The Algorithm to perform Singular Value Decomposition (SVD) */
91.             f = ((x - z) * (x + z) + h * (y / (f - g) - h)) / x
92.         ENDIF
93.         ELSEIF
94.             f = ((x - z) * (x + z) + h * (y / (f + g) - h)) / x
95.         ENDIF
96.         Declare c = 1.0
97.         s = 1.0
98.         FOR i from 0 to k + 1
99.             g = e[i]
100.            y = S[i]
101.            h = s * g
102.            g = c * g
103.            z = 1 + f / h * 2.0
104.            e[i-1] = z
105.            c = f / z
106.            s = h / z
107.            f = x * c + g * s
108.            g = -x * s + g * c
109.            h = y * s
110.            y = y * c
111.            FOR j from 0 to n
112.                x = v[j][i-1]
113.                z = v[j][i]
114.                v[j][i-1] = x * c + z * s
115.                v[j][i] = -x * s + z * c
116.            ENDFOR
117.            z = 1 + f / h * 2.0
118.            S[i-1] = z
119.            c = f / z
120.            s = h / z
121.            f = c * g + s * y
122.            x = -s * g + c * y
123.            FOR j from 0 to m
124.                y = U[j][i-1]
125.                z = U[j][i]
126.                U[j][i-1] = y * c + z * s
127.                U[j][i] = -y * s + z * c
128.            ENDFOR
129.            e[i + 1] = 0.0
130.            e[k] = f
131.            S[k] = x
132.        ENDFOR
133.    ENDFOR
134.    RETURN U, S, V
```

Figure B.7: The Algorithm to perform Singular Value Decomposition (SVD)