

Using K-means Clustering and Similarity Measure  
to Deal with Missing Rating in Collaborative  
Filtering Recommendation Systems

**CHENRUI XIONG**

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES  
IN PARTIAL FULFILMENT OF THE REQUIREMENTS  
FOR THE DEGREE OF  
**MASTER OF ARTS**

GRADUATE PROGRAM IN  
INFORMATION SYSTEM AND TECHNOLOGY  
YORK UNIVERSITY  
TORONTO, ONTARIO

JUNE 2017

©CHENRUI XIONG, 2017

# Abstract

The Collaborative Filtering recommendation systems have been developed to address the information overload problem and personalize the content to the users for business and organizations. However, the Collaborative Filtering approach has its limitation of data sparsity and online scalability problems which result in low recommendation quality. In this thesis, a novel Collaborative Filtering approach is introduced using clustering and similarity technologies. The proposed method using K-means clustering to partition the entire dataset reduces the time complexity and improves the online scalability as well as the data density. Moreover, the similarity comparison method predicts and fills up the missing value in sparsity dataset to enhance the data density which boosts the recommendation quality. This thesis uses MovieLens dataset to investigate the proposed method, which yields amazing experimental outcome on a large sparsity data set that has a higher quality with lower time complexity than the traditional Collaborative Filtering approaches.

# Acknowledgements

Many people have helped me along the road to completing this Master Degree. First and foremost, I would take this opportunity to express my greatest appreciation to my supervisor Dr. Zijiang Yang, who has guided me to develop as a professional researcher and to strength my personality. Dr. Zijiang Yang has supervised me through the whole study of my Master program with her extensive knowledge and patient guidance. In my last year, she spent great efforts and time to help me during the thesis writing with the enthusiastic encouragement and constructive suggestions.

More, I would like to extend my gratitude to the entire faculty in the School of Information Technology for their support, and thank all staff in School who had offered support and assistance to my study. Special thanks go to Dr. Jimmy Huang, who kindly offered to review my proposal and gave me his valuable advices on how to improve my thesis.

Finally, I want to give my sincerest thanks to my parents, who have loved me unconditionally and provided much encouragement to me. Many thanks also go to my cousin Tyrone Xiong who offered insight and gave me another prospective on my research topic by working together for the recommendation system project. And I would like to thank my friend Ke Hu, Xing Tan and James Li for their kind support.

# Table of Contents

<b>Abstract</b> .....	ii
<b>Acknowledgements</b> .....	iii
<b>Table of Contents</b> .....	iv
<b>List of Tables</b> .....	ix
<b>List of Figures</b> .....	x
<b>Chapter 1 Introduction</b> .....	1
1.1 Motivation.....	1
1.2 Problem Definition.....	5
1.3 Significance.....	7
1.4 List of Contribution.....	8
1.5 Thesis Outline.....	9
<b>Chapter 2 Literature Review</b> .....	10
2.1 Memory-based Collaborative Filtering.....	11
2.2 Model-based Collaborative Filtering.....	17
2.3 Hybrid Collaborative Filtering.....	20
2.4 Similarity Measure.....	22

<b>Chapter 3 Techniques</b> .....	25
3.1 Clustering Methods.....	28
3.1.1 Search-based Clustering.....	29
3.1.2 Model-based Clustering.....	30
3.1.3 Center-based Clustering.....	32
3.1.3.1 K-means Clustering.....	32
3.1.3.2 Optimal Cluster and Initial Centroids.....	34
3.2 Similarity Measures.....	36
3.2.1 Pearson Correlation Coefficient.....	36
3.2.2 Cosine Similarity.....	37
3.2.3 Adjusted Cosine Similarity.....	39
3.2.4 Distance Measures.....	40
3.3 Neighborhood Selection.....	40
3.3.1 Pre-filtering of Neighbors.....	40
3.4 Prediction Computation.....	42
3.4.1 Weighted Sum Prediction.....	42
3.4.2 Average Prediction.....	42

<b>Chapter 4 Proposed Methodology</b> .....	44
4.1 K-means Clustering Algorithm.....	46
4.2 Similarity Calculation.....	48
4.3 The Off-line Process .....	49
4.4 Similarity Measure Selection .....	50
4.5 Prediction Computation.....	50
4.6 Neighbour Determination and Selection.....	51
4.7 Methodology Implementation .....	52
<b>Chapter 5 Results and Discussion</b> .....	54
5.1 Data Overview .....	55
5.1.1 Exporting the Rating data.....	57
5.2 Evaluation Metrics.....	60
5.2.1 MAE and RMSE.....	61
5.2.2 Precision, Recall and F1 Score.....	62
5.3 Computing Environments .....	63
5.4 Improving the Recommendation Quality.....	63
5.4.1 Missing Value Prediction.....	64

5.4.2 Neighbour Selection.....	67
5.5 Recommendation Online Scalability.....	69
5.5.1 K-Means Clusters Selection.....	69
5.6 Recommendation Time Efficiency.....	71
5.7 Overall Recommendation Accuracy.....	72
5.7.1 Recall and Precision.....	72
5.7.2 The F1 Score .....	76
<b>Chapter 6 Conclusion .....</b>	<b>79</b>
6.1 Conclusion.....	79
6.2 Future Work.....	81
<b>Bibliography.....</b>	<b>82</b>
<b>Appendix Coding Reference.....</b>	<b>91</b>
A. Clustering .....	91
B. Similarity Calculation.....	95
C. Evaluation.....	98
D. Graph Generation.....	102
E. Main Function.....	103

F. Non-clustering method.....118



# List of Tables

Table 1. The User- Item Rating Matrix.....	49
Table 2. The Rating Matrix after Missing Value Prediction.....	51
Table 3. u.item and u.genre.....	56
Table 4. u.user and u.occupation.....	56
Table 5. Rating table u.data.....	57

# List of Figures

Figure 1. Cisco Forecast Report.....	3
Figure 2. Item-to-Item approach.....	16
Figure 3. K-means Clustering.....	33
Figure 4. K-means Algorithm.....	34
Figure 5. Cluster Center Algorithm.....	35
Figure 6. Cosine Similarity.....	38
Figure 7. Recommendation Approach Flowchart.....	46
Figure 8. Item Clustering Algorithm.....	47
Figure 9. Figure 9 (a), (b) Rating Distribution. ....	58
Figure 10. Average Rating (a), (b) .....	59
Figure 11. Heat-map of rating Matrix .....	60
Figure 12. The MAE rate on varying Matrix Density .....	65
Figure 13. The RMSE rate on varying Matrix Density.....	65
Figure 14. The Density MAE Comparison.....	66
Figure 15. MAE on optimal number of Nearest Neighbours.....	68
Figure 16. RMSE on optimal number of Nearest Neighbours.....	68

Figure 17. MAE of K-means Clusters.....	70
Figure 18. RMSE of K-means Clustering.....	70
Figure 19. K-means Clustering Compared to Traditional method.....	71
Figure 20. The Recall rate of List Range 50 – 500.....	73
Figure 21. The Comparison of Proposed method to Traditional method.....	74
Figure 22. The Precision rate of List Range100-1000.....	74
Figure 23. The Precision Comparison on List 100.....	75
Figure 24. The F1 score.....	76
Figure 25. F1 score Comparison to the Traditional method.....	77

# Chapter 1. Introduction

---

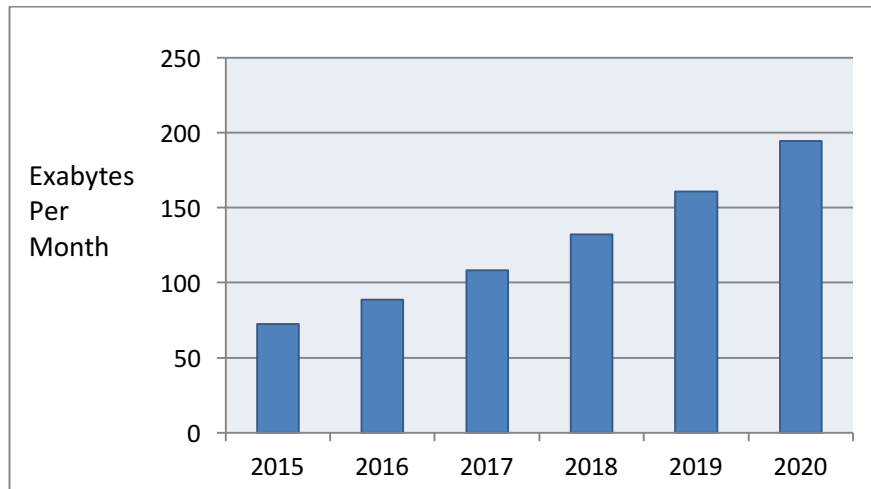
A recommendation system has become a vital and necessary component for many web applications nowadays, and there is an increasing demand for delivery time efficiency and high-quality prediction by the users. In real-life application, millions of users and Exabytes of data have resulted in data sparsity, online scalability and time-complexity problems to build high performance recommendation systems. This thesis is concentrated on solving these problems. The proposed methodology is applied in a Collaborative Filtering Recommendation system to address user-item matrix sparsity problems, which improves recommendation accuracy and speed. The proposed methodology can also be applied to other recommendation systems in which the dataset is sparse.

## 1.1 Motivation

A recent statistical report states that the total number of indexed World Wide Webpages has reached 4.25 billion (**World Wide Web Size 2016 [1]**) and the number of internet users is 3.6 billion (**Cisco Forecast Report, 2016 [2]**) in 2016. The Internet has become an integral part of the daily life for many people. People shop, consume, entertain and study online. By way of

illustration, it is said that users generate over 4 million posts on Facebook every minute, Instagram users click 100 million “likes” per hour (**G. Simos, 2015 [3]**), and 497 million products have been sold online by Amazon (**Amazon, 2015 [4]**).

Obviously, the volume of data is incredibly large and is continually growing at a high rate. In fact, the current Cisco Visual Networking Index (VNI) forecast projects that global data volume will nearly triple from 2015 to 2020 (**Cisco Forecast Report, 2016 [2]**). However, this myriad of information has brought an information overload problem. To discover the needed information from the massive amount of data available poses a significant challenge. Today, tools like search engines, portal websites and social media outlets are the primary ways in which people can obtain information faster and more easily. Nevertheless, locating the required object from hundreds of relevant items can always be time-consuming. Recommendation Systems have been developed in order to solve this problem. Recommendation System is a subclass of the information filtering system seeking to predict the ‘rating’ or ‘preference’ that a user would give to an item (such as movie, music etc.) or social media (e.g. people or organization) (**F. Ricci, L. Rokach and B. Shapira, 2010 [5]**). It aims to address the information overload problem and personalize the content or item to the user. However, the RS is not only designed to find what the user is looking for, it will also discover the user’s potential interests.



*Figure 1. Cisco Forecast Report 2015 [2]*

Recommendation systems have been considered an independent research discipline since the 1990s when the study team GroupLens at the University of Minnesota launched the GroupLens system in 1994 (P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, 1994 [6]). Based on how recommendations are generated, recommendation systems can usually be classified into three categories: Content-Based Filtering (CBF), Collaborative Filtering (CF), and Hybrid Filtering (HF). Each category has its own special features and purpose. The CBF method extracts features to build user profiles in order to find similar items in the user's history. While CBF always recommends similar items, one of its limitations lies in the fact that it lacks novelty. The CF method suggests new items or predicts the utility of a certain item for a particular user based on his/her historical likes and the opinions of other like-minded users. While both CBF and CF approaches have their own advantages, both fail to provide quality recommendations under specific conditions. The HF method integrates and optimizes the advantages of several recommendation methods while reducing the disadvantages of either single approach.

Currently, recommendation systems are becoming a standard component of commercial websites with the dual purpose of helping the users explore their interests, and also help

businesses drive more sales. This second goal is critical because research shows that 80% of sales come from 20% of the most popular products, an occurrence known as a ‘long tail phenomenon’ (C. Anderson, 2006 [7]). The existence of such a significant proportion of low-volume products indicates the massive increase in profitability available to the business if they can find a way to sell more of them. The advantage of CF method is that it recommends interests that the user may have not discovered yet based on his/her previous history and the similar styles of other users. The CF method can be further sub-divided into two types: the memory-based approach and the model-based approach. The memory-based approach calculates the user or item’s previous rating data, whereas the model-based approach uses data mining algorithms to find patterns based on the user’s behavior history data, which can be either explicit or implicit (P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, 1994 [6]). Explicit ratings include user reviews, opinions, and rating scales between 1 and 10. Implicit ratings are the interaction data between the user and website, usually including web clicks and stay time.

Data input into recommendation system is varied. In general, the data can be categorized into three categories: user profile such as name, address, and age; item profile such as description, price, and picture; and reviews such as “likes,” ratings, or written. These categories correspond to the row, column, and element in an  $m \times n$  matrix, which is used to determine the relationship between users and items. The  $m \times n$  matrix has “ $m$ ” number of users and “ $n$ ” number of items.  $R_{mn}$  means user “ $m$ ” has rated item “ $n$ .”

Amazon extensively uses a unique recommendation system to personalize recommendations for the user, since traditional Recommendation system cannot scale to Amazon’s massive user and product database. They utilize their own algorithm called Item-Based Collaborative Filtering, which calculates the similarity of items and builds a product-to-product matrix; whenever a user

purchases or views a product, Amazon recommends similar items from the product-to-product matrix. Additionally, on the front page the user will find lists of products titled “You viewed” and “Customers who viewed this also viewed.” According to scientist Greg Linden’s blog, 35% of Amazon’s sales come from its recommendation system (**B. Sarwar, G. Karypis, J. Konstan and J. Riedl, 2001[8]**) which indicates its significant success.

Netflix began as a DVD-by-mail rental service and grew to a global online movie and television provider. As the world’s largest online movie provider, Netflix has a wide range of selections to provide to the users by a recommendation system. They rely heavily on recommendation technologies to drive customer selections; approximately 60% of its subscribers select their movies through the recommendation list. In order to achieve a better recommendation result, Netflix held an open competition from 2006 to 2009: The Netflix Prize for the best CF algorithm to forecast user ratings based on user history. The prize worth USD 1 million was based on the improvement of Netflix’s Cinematch recommendation systems (**J. Bennett and S. Lanning, 2007 [9]**). In 2007, over 20,000 teams attended this event from over 150 countries and 2,000 teams submitted over 13,000 predication sets. (**F. Ricci, L. Rokach and B. Shapira, 2010 [5]**).

## **1.2 Problem Definition**

Producing high quality recommendations, performing many recommendations per second for millions of users and items, and achieving high coverage in the face of data sparsity are the key challenges for recommendation systems (**B. Sarwar, G. Karypis, J. Konstan and J. Riedl, 2001[8]**). Although a memory-based approach has been widely implemented in recommendation



systems, a number of problems still exist; typically, building up an efficient RS encounters problems of dataset sparsity, cold-start, time efficiency, and scalability.

Of these, the key problem of memory-based CF approach is data sparsity of user and item matrix. In many large scale e-businesses, both users and items are increasing fast. Hundreds and thousands of items and users may not have any history to begin with, with little in terms of correlation between them; i.e. most users belonging to the system have seldom used a new item and most of the tags and items have been used or tagged by only a small subset of users. Accordingly, the item and user matrix can be extremely sparse and the similarity and correlation between objects could be zero. The data sparsity causes traditional recommendation systems to be particularly sensitive to cold start problems (**V. Zanardi, [10]**). The cold start refers to user, item, and system cold start problems. Recommendation systems require history data to start with, but when a user registers with no data, the system basically knows nothing about the user. Therefore, the recommendation system cannot present any personalized recommendations to this user. Similarly, when a new item is added to the system with no rating or action data, it will not be recommended to anyone. To deal with this problem, a website usually asks user to fill up the personal information such as age, profession, and interests, or even allow users to log in through their social media accounts to gather information to be able to start recommendations.

In addition, the fact that the size of the dataset continues to grow rapidly causes other scalability and time efficiency problems. A scalable recommendation system is able to provide the real-time recommendation; however, the growing volume of users and items makes the calculation very time consuming and hampers scalability.

## 1.3 Significance

Many researchers in academia and large business have long been dedicated to the recommendation system field.

The Xerox Palo Alto Research Center first introduced the term “Collaborative Filtering” in 1992 which was developed to filter files (**D. Goldberg, D. Nichols, B.M. Oki, and D. Terry, 1992, [11]**). Resnick et al. (**P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, 1994 [6]**) at the University of Minnesota and MIT proposed automatic recommendation system GroupLens in 1994. This thesis is focused on the memory-based CF method. Although the memory-based CF method has been very popular, some problems still impede its application such as data sparsity in the user or item matrix. In other words, when users and items are accumulated in the system quickly, there are not enough reviews to generate recommendations. As a result, the unrated items cause the sparsity of user rating matrix, which affects the recommendation quality; thus, the missing ratings need to be dealt with.

For the purpose of resolving the data sparsity problem to produce better recommendations, Breese et al. (**J.S. Breese, D. Heckerman, and C. Kadie, 1998 [12]**) applied Bayesian network. Schafer et al. (**J.B. Schafer, J.A. Konstan, and J. Riedl, 2001 [13]**) used neural network to achieve better performance. E-business leader company Amazon proposed an “item-based Collaborative filtering” method in order to scale the very large customer-based retailer (**G. Linden, B. Smith, and J. York, 2003 [14]**).

After careful study, this thesis integrates the clustering algorithm and similarity measure to predict the missing ratings. First, the clustering method is used to group similar users and items into groups. Then, only the most relevant users or items in the same group will be used to make

rating predictions, which improves time efficiency significantly. Additionally, this thesis compares the similarity between users and items to determine whether user or item information should be used to predict the missing ratings. If the similarity between users is greater than that between items, the user information will be used to make the prediction and vice-versa. The proposed method is applied to a MovieLens dataset. Experimental results show that the proposed method can speed up the recommendation significantly and produce a better recommendation quality.

## 1.4 List of Contribution

The contributions of this thesis are provided below:

- **Time efficiency:** K-means clustering is applied to the dataset, which will speed up the later calculation. The dataset will be divided into smaller clusters and the similar data will be gathered into one cluster. The similarity calculation will be processed in each cluster instead of the whole dataset to search similar items or users, which could save a significant amount of time.
- **Improve online scalability:** An excellent and scalable recommendation system is able to provide the real-time recommendation to the users. Applying a clustering method could increase the system online scalability and decrease the processing time.
- **Improve prediction quality:** The pre-calculated similarity is used to compare and determine whether user-based or item-based criteria will be used for the rating forecast, which will improve the prediction quality.

## 1.5 Thesis Outline

This thesis is organized into the following chapters:

Chapter 2 - Discusses the Recommendation literature, different solutions to address the data sparsity, cold start, online scalability and time complexity problems, and how data mining technologies used to improve the recommendation systems. Chapter 3 - Defines relevant technologies applied to recommendation system, K-means clustering and several of similarity measurements. Chapter 4 - Proposes a rating prediction collaborative filtering approach. The clustering technique will be applied to the dataset first, and then the similarity for both user and item will be calculated and compared in each cluster. Finally, the rating for unrated item will be predicted based on the similarity result. Chapter 5 - Introduces the MovieLens data sample, presents the optimal clusters, best neighbour numbers and comparison of the missing value prediction method to the non-prediction method, discusses the observed advantages and limitations of the proposed methodology. Chapter 6 - Concludes the thesis and discusses possible future work.

# Chapter 2. Literature Review

---

The field of Recommendation Systems has been studied for over a decade, with several proposed techniques already being implemented into real world applications. Based on recent researches, recommendation systems can be categorized into two groups, the Collaborative Filtering recommendation system and the Content-based ones (A.S. Das, M. Datar, A. Garg, and S. Rajaram, 2007 [15]). The content-based approaches recommend similar objects based on users' past experience and extracted object's content profiles. However, content extraction from objects like movies, music, and videos is a very challenging and time-consuming task. Therefore, collaborative filtering, which is based on past ratings, does not require any content information from neither the user nor the item itself. One of the notable CF advantages is that they can consider profiles from other users for producing the requested personalized recommendation unlike the content-based methods that produce similar items only. In this section, a review of CF approaches will be conducted.

The term 'Collaborative Filtering' was firstly introduced in an email handling system called Tapestry in 1992 at the Xerox Palo Alto Research Center. The system was developed in order to receive and filter files and documents arriving in a continuous stream. The primary innovation

was that the filter queries had predictable semantics (**D. Goldberg, D. Nichols, B.M. Oki, and D. Terry, 1992, [11]**). However, Tapestry was a manual filtering system that required human intervention to produce the required recommendations. Two years later, (**Resnick et al. P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, 1994 [6]**) from the University of Minnesota and MIT introduced a Netnews system, namely the GroupLens, which was an automated recommendation system utilizing Nearest Neighbor Collaborative filtering method for helping users make selections based on other people's opinions. An additional innovation was the proposed rating system that allowed users to rate other's messages allows the query engine to compare and make suggestions according to these ratings. This concept is considered the foundation of modern Collaborative Filtering and is used until today.

## **2.1 Memory-based Collaborative Filtering**

Based on a popular previous research, there are two general classes for the CF approaches: Memory-based Collaborative Filtering and Model-based Collaborative Filtering systems (**B. Sarwar, G. Karypis, J. Konstan and J. Riedl, 2001[8]**). The Memory-based Collaborative Filtering can be defined as an approach that exploits the entire user-item data matrix to generate the predictions. Usually, these systems tend to seek for groups of similar items or users who bought or rated similar or even the same items, known as the neighbor-based methods, with the most popular algorithms being the Top-N and the kNN ones. Memory-based methods also known as user-based collaborative filtering are found mostly in the recommendation field. The memory-based CF method is widely used and successful, becoming popular due to its features that do not require additional user or item information since it produces the personalized

recommendation solely by the user's interests. However, it suffers also from several shortcomings, such as the cold-start, the data sparsity and scalability.

Based on previous research, the user-based methods are helpful in addressing the cold-start problem, considering the stereotype that a user may like an item also liked by someone's profile similar to his. To address this, it firstly applies a nearest neighbor to the user's profile, considering each user's profile as a vector, and then by cross-referencing the user's vector similarities against the other user's vectors. The  $k$  Nearest Neighbors ( $k$ NN) is the most popular and efficient algorithm used in the CF field. The  $k$ NN is an algorithm that determines the "closeness" between objects based on certain similarity measures. The calculation process usually involves the similarity computation between two users,  $a$  and  $b$ , based on their user item ratings. Similarly, the item based  $k$ NN version calculates the similarity between two items  $x$  and  $y$  based on the ratings. The user-based CF by using  $k$ NN algorithm uses the following three steps for the final recommendation (**S.K. Tiwari and S.K. Shrivastava, 2015 [16]**):

- (i) Selected similarity measure produces a set of similar user  $a$ . The  $k$  neighbors of user  $a$  are the  $k$  similar users to user  $b$ .
- (ii) Next, one of the following approaches is selected in order to obtain the prediction of item  $x$  on user  $a$ : the average, the weighted sum and the adjusted weighted aggregation (deviation-from-mean).
- (iii) Selection of the top- $n$  recommendations (items or users)

Algorithms like the  $k$ NN utilize the entire dataset of user or item preferences when generating the recommendations. These algorithms are easy to implement with a low training cost. However, the online performance tends to be slow as the size of the user and item sets grow

larger making the algorithms unsuitable for large datasets. Moreover, since the Top N relevance is based on a large number of users, the user calculation computational cost is significantly increased. Sinha and Swearingen (**R.R. Sinha, and K. Swearingen, 2001 [17]**) proposed a method to reduce the user calculation range by only targeting at the user's friend circle since people are likely to favor items liked by their friends influenced mostly by their social circle compared to the quality of recommendations made by the total users. During this study, 19 college students were invited for a survey, where each one should pick either three book or movie systems, as well as evaluate recommendations made from their friends. Comparing Amazon, Sleeper, RatingZone Quick Picks, Reel, MovieCirtic and Friends recommendation results showed a 20% better recommendation performance when friends are considered compared with conventional systems.

In early days, identifying the user's friends in the internet was challenging, however, current social network websites make such gathering information possible. In web-based social networks, the relationship status between the users can also define their similarity. Therefore, Al-Sharawneh and Williams (**J.A. Al-Sharawneh, and M. Williams, 2010 [18]**) proposed a method that relies primarily on people's trust in social network for making recommendations. In addition, advice from trusted experts leads to better recommendations improving prediction quality. The proposed approach generates recommendations based on leader's credibility in a "follow the leader" model of Top-N recommenders by incorporating social network information into a user-based CF approach. This is achieved by firstly extracting the website social network's trust information and then a clustering technology is applied to the user's relationship database. By testing this approach in three datasets, results showed that a leader based clustering method was highly effective, providing high accurate predictions.



Today, privacy issues on the internet are of everyone's main concern. Thus, collecting sensitive information from people's social network is challenging. However, user-based CF approach requires a quite complete historical dataset to meet its good recommendation accuracies. To help starting with the new users without historical data, Amatriain, Lathia, and Pujol (**X. Amatriain, N. Lathia, JM. Pujol, H. Kwak, and N. Oliver, 2009 [19]**) presented a novel recommendation method based on external expert's opinions. The authors found that a considerable error in explicit feedback based CF approach was prominent to the user's explicit feedback noise. Therefore, they used an independent dataset completed with expert's information instead of applying nearest neighbor algorithm on the user's rating data. The experts and their information are selected from independent expert sources with at least certain amount of ratings. Afterwards, a similarity matrix of the expert dataset is created between each user. Based on experimental results, the proposed method addressed some of the shortcomings in traditional Collaborative filtering approach such as the cold-start, privacy and data sparsity.

Konstan, Miller, Maltz, Herlocker, Gordon, and Riedl, (**J.A. Konstan, B.N. Miller, D. Maltz, J.L. Herlocker, L.R. Gordon, and J. Riedl, 1997 [20]**) from GroupLens indicated that users need certain history ratings to start the prediction. However, royal users may abandon the system when receiving only a fraction of the articles that they had read previously. Moreover, without any recommendations many users abandoned the system before providing a rating feedback. To address this data sparsity and cold-start issues, where no history data is available to start with, they proposed a method of providing average ratings for all the users, by a combination of implicit ratings and filter-bots, which reduced the effort. The lack of rating affected also the cold-start and sparsity difficulties, reducing the quality of recommendation results.

Since the user-based CF approach has attained a great success in recommendation system, it has also been widely used in many commercial websites. However, the tremendous growth of users in recent years introduces big challenges for the current recommendation systems. According to amazon's report, active users have reached 240 million in 2014 (C. Smith, 2014 [21]). Therefore, Sarwar, Karypis, Konstan and Riedl (B. Sarwar, G. Karypis, J. Konstan and J. Riedl, 2001[8]) has proposed a novel recommendation technology that collaborate with very large-scale user databases that can produce quickly high quality recommendations. Like the user-based approaches, the item-based approaches use a user-item matrix to identify item relationships and using these relationships to compute user predictions.

Item-based approaches predict the item rating based on similar items' rating information. These approaches are based on the assumption of content-based recommendation that a "user may like a similar item as a one liked in the past". For example, users who liked pizza may like pasta. Unlike content-based approaches that use item profiles to make prediction, these approach used item rating to make the prediction.

The first step of an item-based recommendation is the same as the user-based recommendation that is to find the nearest neighbor. Every single item and its rating are considered a vector compared to every other item for checking how closely they relate. The item-based approach has some advantages as listed below:

- Easy to calculate, especially when the number of users are greater than the number of items.

- Easy to explain to the users why the system made this recommendation. Moreover, users can simply change the recommendations by adding or deleting their history data on profile to affect the recommendation results.

The two traditional memory-based collaborative filtering approaches are the user-based and item-based methods by finding a group of similar users or items from a user-item matrix based on rating and purchase history. Then, they match similar items that were rated and purchased by each customer. For very large-scale datasets, similarity calculation is computationally expensive, and thus the scalable recommendation system should perform this calculation offline. To meet the short real-time response for users, developers at Amazon (**G. Linden, B. Smith, and J. York, 2003 [22]**) have proposed an item-based collaborative filtering method aiming at finding similar items, which was different from the traditional methods which targeted similar users. Rather than finding the similar customer with the same taste, the proposed item-item algorithm attempted to find similar items instead, according to user's rating and purchase history. The proposed algorithm utilizes an offline item table that contains similar items. Thus, the algorithm is fast especially for extremely large datasets.

---

**Algorithm** Item-to-Item approach [22]

---

For each item in product catalog,  $I_1$   
    For each customer  $C$  who purchased  $I_1$   
        For each item  $I_2$  purchased by customer  $C$ ,  
            Record that customer purchased  $I_1$  and  $I_2$ ,  
For each item  $I_2$   
    Computer the similarity between  $I_1$  and  $I_2$

---

*Figure 2. Item-to-Item Approach (G. Linden, B. Smith, and J. York, 2003 [22])*

This method was used immediately by the very large customer-based retailers and favored high-quality recommendations.

## 2.2 Model-based Collaborative Filtering

In 1998 one of the earliest works on model-based approaches was by Breese, Heckerman and Kadie from Microsoft (**J.S. Breese, D. Heckerman, and C. Kadie, 1998 [12]**), where they evaluated and compared several algorithms on collaborative filtering approaches, including correlation coefficients, vector-based similarity, clustering, and Bayesian methods. Moreover, they presented some comparison results addressing the collaborative filtering scalability and computational issues. This paper presents extensive results on test methods in several different situations. Results showed that correlation and Bayesian networks present less advantages compared over to other techniques when datasets have a low number of votes. Furthermore, they concluded that only Bayesian networks require small memory sizes, and allow faster predications compared with the correlation method of memory-based approaches. However, Bayesian method requires several hours for the learning phase and requires re-building the model in every update which is cost demanding.

Das, Datar, Gary and Rajaram from Google Inc. (**A.S. Das, M. Datar, A. Garg, and S. Rajaram, 2007 [15]**) have used a clustering method to recommend news to the users. As the number of users grew, Google News faced challenges for large and dynamic datasets, since millions of users and their underlying item sets were continually updating. These rapid updates made the system un-scalable and not efficient enough, which made the users no longer interested in the item unless the model was updated before few hours. For those reasons, Google proposed scalable collaborative filtering approaches which use three approaches: the MinHash clustering, the Probabilistic Latent Semantic Indexing, and the co-visitation counts. Additionally, the system

is able to provide instant user gratification feedback by verifying the user clicks. Experimental results on real world datasets showed the improved scalability and efficiency in live traffic.

In order to maintain the scalability as the information updates very frequently, the recommendation system must entail a low time complexity for the recommendation calculation process. Clustering is a popular data mining method and has been used for scalability improvement and processing time efficiency. However, the traditional clustering method creates disjoint clusters or enforces the entire dataset partitioning. In order to diminish these issues, Georgiou and Tsapatsoulis (**O. Georgiou, and N. Tsapatsoulis, 2010 [23]**) proposed a clustering method inspired from genetic algorithms. The proposed method creates dense clusters sharing common elements and converges rapidly even in very high dimensional spaces, allowing larger cluster to overlap. In order to apply the genetic algorithm into the cluster, the optimal measure of each string in the random population is firstly calculated. Secondly, genetic operators corresponding to mathematical models of crossover and mutation are applied to population. Finally, the new population replaces the old one. Based on results, their clustering method is faster and more accurate compared to the classic one with significantly better recommendations.

The expectation for reducing the overall prediction and recommendation time is based on the logical assumption that when training smaller datasets or clusters time will be less compared with the case when the whole dataset is used for training. In 2009, Braak, Abdullah, and Xu (**P.T. Braak, N. Abdullah and Y. Xu, 2009[24]**) clustered the user profiles to improve the model-based collaborative filtering recommendation performance. The basic idea is partitioning the training data into user-based profile clusters. Thus, the partitioned data will represent user segments that are more concisely targeting similar users with an increased prediction speed and without loss in accuracy. In simple terms, the partitions are defined as user interest groups, with

the author introducing the notion of grouping similar movies into different genres for determining user's interest. The interest of user is defined by both the ratings and the viewing habits. In data pre-processing steps, each movie has to be replaced by its genre and inherit its rating into each genre. Once the genre interest is determined, the profile is used for clustering. Results showed that the clustered training data had no impact on the accuracy of the recommendation system and it allowed significant improvements in the processing time.

A Bayesian method has been studied and applied by many researchers in personalized model-based collaborative filtering recommendations. Babas, Chalkiadakis and Tripolitakis (**K. Babas, G. Chalkiadakis, and E. Tripolitakis, 2013 [25]**) proposed a novel Bayesian approach, which uses a minimum set of ratings from various users to provide an overall recommendation. They used a Bayesian analysis that contained labels from items without relying on others' taste by not attempting to predict user ratings, providing the ability to define multivariate Gaussian dimensions that represent users. This method is well suited for sparse datasets, and can be used as a "bootstrapping" tool providing recommendations until more data is available. After testing this method on the MovieLens dataset, it performed better than methods utilizing other users' ratings.

Besides the clustering and Bayesian, other technologies have also been applied to collaborative filtering recommendation systems. Aggarwal, Wolf, Wu and Yu from IBM Watson Research Center (**C.C. Aggarwal, J. L. Wolf, K. L. Wu, and P.S. Yu 1999 [26]**), believe that collaborative filtering represents a total different data mining problem. More precisely, traditional data mining algorithms like clustering, and  $k$ NN do not seem to be appropriate in high dimensionality, closeness measure and incomplete specification problems. To overcome these problems, they introduced a new graphic-theoretic algorithm based on two novel concept

approaches, namely the Horting and the Predictability. In the Horting technique each node represents users and edges between nodes indicate the degree of similarity between them. The similarity is calculated by measuring the nearby nodes to indicate the nearest user. The difference from the nearest neighbor is that Horting technology may pass through other users who have not rated the selected items. Results showed that this technology is fast, accurate and scalable and it requires only a modest learning curve.

Then Zhou, Yang, and Zha (**K. Zhou, SH. Yang, and H.Y. Zha, 2011 [27]**) proposed a novel cold-start recommendation method called the Functional Matrix Factorizations (fMF) that addressed the initial interview construction within the context of user learning and item profiles. The idea was to ask new users opinion questions. Based on their feedback, the recommendation system could gradually refine user's profile. fMF constructs a decision tree with each node being an interview question, and the user at each node being partitioned into three disjoint subsets "like", "dislike" and "unknown" based on the interview answer. Then, the user profiles are built according to their answers. Experimental results on the current dataset demonstrated that the proposed FMF algorithm significantly outperformed the existing methods in case of cold-start recommendation.

## **2.3 Hybrid Collaborative Filtering**

Memory-based method is an early approach used in many commercial systems. However, the memory-based method still suffers from two fundamental problems, the data sparsity and scalability. Thus, to solve these problems many researchers have used hybrid methods that are combined with model-based method. Xue, Lin, Yang, Zeng, Yu, Chen (**G.R. Xue, C. Lin, Q. Yang, W. Xi, H.J. Zeng, Y. Yu, and Z. Chen, 2005 [28]**) proposed a novel hybrid approach

that uses a clustering smoothing technology that combines strengths from both approaches. In these approaches, the clustering technology is used to generate the basis and neighborhood from the training data. Clusters are used for grouping and smoothing the unrated user data. Clustering in smoothing permits the strength characteristic integration of both memory and model-based methods. By using the history rating information from groups of similar users, the unrated user's predicted group can be quickly performed. As a result, such approaches are applied on EachMovie and MovieLens datasets which exhibit higher accuracy and increased efficiency in their recommendations.

Yu, Xu, Tao, Ester, and Kriegel (**K. Yu, X. Xu, J. Tao, M. Ester, and H. Kriegel, 2002 [29]**) noted that the scalability is one of the biggest challenges that memory-based algorithms currently face. In this paper, they focus on typical user preference datasets containing many missing values. They proposed four novel instance reduction techniques called TURF1-TURF4 as a preprocess stage before the memory-based collaborative filtering approach. The concept is to make predictions to relevant instances instead of the entire database, in order to speed up the process. The proposed approach has four steps; the first algorithm operating in an incremental manner, starts with a few training instances, and then adds those instances with a novel profile into the training set. The second algorithm performs the filtering, and selects the instance with the most rational profile into the training set. The third algorithm combines with former algorithms to pick the instances with strong rational and novel profile. The final algorithm is considering the potential of storage reduction. Overall, they focus on clustering algorithms and methods that reduce the process time and improve the accuracy by limiting potential noise.

Many algorithms have been proposed for solving the data sparsity and scalability. Rashid, Lam, Karypis and Riedl (**Al.M. Rashid, S.K. Lam, G. Karypis, and J. Riedl, 2006 [30]**)



proposed a Hybrid Collaborative filtering approach that mainly targets in improving the recommendation quality. Their CLUSTKnn method is a simple and intuitive algorithm which is inexpensive and well suited for operating in large-scale datasets. Their approaches first compress the data using BISECTING K-means clustering technologies, and then select the largest cluster to be split in two sub-clusters, repeating this step until they get the best intra-cluster similarity. Finally, recommendations are generated quickly using the Nearest Neighbor method. Comparison results with number of approaches such as the Item-based  $k$ NN, user-based  $k$ NN, and SVD model-based Collaborative filtering approach showed that the hybrid method did not obtain the best results, but it is still intuitive and presents very good recommendation accuracies.

## 2.4 Similarity Measure

The concept of similarity is important in many scientific fields such as mathematics, statistics, and Computer Science. The similarity measure can define the relevance between two items, two users, two queries, two articles, and more. There are many similarity methods proposed in the literature. Usually, they are categorized based on their data types such as numerical data, categorical data, time series data, binary data, and mix-type data. Numerical data similarity methods usually use the Euclidean distance, the Manhattan distance, the Minkowski distance, and the average distance (G.J. Gan, C.Q. Ma, and J.H. Wu, 2007 [31]). On normal scale categorical data, the Chi-square statistic and the overlap measure are widely used. The best-known method dealing with Binary data is the Jaccard coefficient. In practice, the dataset may be composed of various types of data. In such case, the Gower similarity coefficient and the generalized Minkowski distance are used since they can deal with complex datasets. In this

thesis, the rating data is considered as ordinal data. Therefore, Cosine, Pearson, and Distance similarity measures will be discussed in details.

Both user and item-based methods are based on the hypothesis that users will like similar items or a user will like items liked by similar people. This hypothesis made the similarity estimation one of the most significant and critical steps while implementing recommendation systems. Generally, common methods of similarity calculation include the Pearson correlation, Cosine similarity, Jaccard similarity etc. However, traditional similarity methods are still limited in meeting today's requirements. Fiasconaro et al. (**A. Fiasconaro, M. Tumminello, V. Nicosia, V. Latora, and R.N. Mantegna, 2015 [32]**) indicated that similarity measures are crucial for the development of efficient personalized recommendation systems in many cases. The basic idea of a recommendation system is to predict user interests based on similarities of certain users or items. They combined recommendation scores derived from users and objects similarities computed by existing methodologies such as Jaccard, Network-based Inference, and Pearson correlation. In addition, **Y. Zou, A.J. An, and X.J. Huang [33]**, **Y. Liu, X.J. Huang, and A.J. An [34]**, **J. Wei, J. He, K. Chen, Y. Zhou, and Z. Tang [35]**, **A. Forestiero [36]**, to mention a few, also conducted research in this area.

The computation of similarity measures has based on the traditional distance and cosine vector such as cosine and Pearson. However, it has been proved that they are not effective in the recommendation problem field. Ahn (**H.J. Ahn, 2008 [37]**) presented a new heuristic similarity method that focuses on recommendation quality improvement under a cold-start condition, was named PIP (Proximity-Impact-Popularity) measure that utilizes specific domain interpretations for user rating on products, overcoming the drawbacks of traditional methods in cold-start conditions. Firstly, the Proximity factor evaluates the difference between two ratings checking if

the ratings are in agreement and applying a penalty to the ratings in cases of disagreement. Next, the impact factor considers the chance of a user's like or dislike upon the item. Last, the popularity factor is applied when an impact factor value is further from the average rating value that similarity could obtain. After extensive tests on MovieLens, Netflix, and Jester datasets, results showed that the PIP measure is very accurate and can achieve the overall best performance and lowest absolute mean error. The primary contribution of this method is that it requires no additional data and minimal additional implementation compared to the existing collaborative filtering approaches.

# Chapter 3. Techniques

---

The advent of information technology has brought easiness to everyday life in a number of ways; however, the vast amount of available digital data has introduced several challenges for its compilation and its access. Companies such as Amazon and Google are successful in solving these problems, but personalization and prioritization are areas that still need improvement. Recent studies have shown that a recommendation system possesses the capacity to address these complex areas. The differ Recommender systems has differ way to analyze these data sources to develop notions of affinity between user and item. The collaborative filtering systems analyze historical interactions alone, while content-based filtering systems are based on profile attributes (P. Melville, and V. Sindhwani, 2011 [38])

Collaborative filtering approaches overcome some of the drawbacks of content-based approaches. It can be further sub-divided into memory-based (neighborhood-based) and model-based approaches. Both the memory collaborative filtering approaches and model-based approach are using the user-item rating matrix as data sources. The memory-based approaches can be further divided into the user-based and item-based methods. The user-based approaches, evaluate the interest of a user  $u$  for an item  $i$  using the ratings for item by other users, called

*neighbors* that have similar rating patterns (P. Melville, and V. Sindhwani, 2011 [38]). Item-based approach, on the other hand, predicts the rating of user  $u$  for an item  $i$  based on the ratings of  $u$  for items similar to  $i$  (P. Melville, and V. Sindhwani, 2011 [38]). Differ to the memory-based approaches, the model-based approaches use these ratings to study the pattern of a predictive model. The general idea is to model the user-item rating matrix with factors representing latent characteristics of the users and items. The model-based methods usually include clustering, classification, Singular Value Decomposition and Support Vector Machines etc.

The main advantage of collaborative filtering recommendations is simply to implement, efficiency, and it provides a concise and justification for the computed predictions. However, current recommendation systems still suffer from data sparsity, cold start, online scalability, and information overload issues. This thesis focusses on the current recommendation system challenges such as data sparsity, cold start and scalability problems. This chapter will study the following two types of recommendation related technologies: clustering methods and missing value handling using similarity measures. Moreover, it will examine in depth these two techniques in order to explore their advantages and disadvantages.

Traditional clustering methods are used primarily in the Data Mining field. The clustering methods are characterized by their descriptive functions and mostly used in unsupervised learning (G.J. Gan, C.Q. Ma, and J.H. Wu, 2007 [31]). They are used to determine new sets of categories with the objects of the same set being highly similar. Thus, a big dataset is divided into smaller clusters and similar data are including in one cluster. This operation is widely used in recommendation system where the target users or items are clustered within similar sets. The clustering operation is then applied in each cluster instead of the whole dataset, in order to

further classify similar items or users. In such way, recommendation systems can be scalable and efficient to provide real-time recommendations to the user. In this thesis, the clustering method is used to pre-process the dataset, where the proposed method calculates the similarity of the target user or item within each cluster to determine the nearest neighbors which are relevant to the objects, reducing the computational burden.

Cases of missing value often occur when no data value is stored in a variable under observation. There are several types of missing value: Missing completely at random, missing at random, and missing not at random (**J.L. Schafer and J.W. Graham, 2002 [40]**). The issue of missing value is an active topic of research and requires data processing. Handling an incomplete data set can cause inaccurate inferences to the final result and may affect further operations. Nowadays, many technologies or methods have been proposed to solve the missing value problem, such as replacement, deletion, mean calculation, mod average and statistic models for predicting the missing value.

The key step of collaborative filtering approaches is to identify the most relevant user or item through the user-item rating matrix by utilizing the rating history. Thus, correct and accurate ratings are crucial to the collaborative filtering approaches. However, since the size of user and item data has grown exponentially, cases of many unrated objects can lead to a very sparse user-item rating matrix affecting the recommendation quality. The missing value is addressed in this thesis by the use of similarity methods since it is identified as a critical and necessary step in Collaborative filtering recommendation systems.

For this reason, an unrated item in user-item rating matrix is treated as a missing value that must be predicted using similarity methods for improving the final prediction quality. Moreover,

a similarity pre-calculation has been performed for both items and users, arranging the top similar groups for further improvement on the online scalability and recommendation speed. The pre-calculated similarity is used to compare and determine whether user-based or item-based criteria should be selected for the rating forecast. This selection improves the prediction quality upon the used similarity method.

### **3.1 Clustering Methods**

Clustering is one of the most commonly used techniques in collaborative filtering technology and is applied across diverse research fields including machine learning, bioinformatics, data mining and image analysis. The clustering technique, also known as clustering analysis, has been proved a very popular statistical tool widely used also in pattern recognition, knowledge discovery, and statistical data analysis (**A.K. Jain, 2010 [39]**). This technique creates partitions in a set of data to create similar data subsets. Once a cluster has been created, it is easy to calculate the other user's feedback in the subgroup and create the desired response to the individual user. One of the characteristics of a properly clustered group is its high intra-group similarity and its low similarity with other groups. Self-organizing maps (SOM) and K-means clustering are two of the most widely used techniques in clustering. The Self-organizing maps (SOM) perform an unsupervised learning while the K-means clustering classifies a set of  $n$  items into  $K$  clusters (**G.J. Gan, C.Q. Ma, and J.H. Wu, 2007 [31]**). These clustering techniques can be used to reduce the set of candidates in a collaborative algorithm by increasing the intra-group similarity and decreasing the inter-group similarity.

Typical clustering methods can be categorized to hierarchical clustering, center-based clustering, fuzzy clustering, search-based clustering, graph-based clustering, grid-based

clustering, model-based clustering, subspace clustering, and density-based clustering (**G.J. Gan, C.Q. Ma, and J.H. Wu, 2007 [31]**). The following section will discuss the three most popular clustering methods: Center-based Clustering, Search-based Clustering, and Model-based Clustering.

### **3.1.1 Search-based Clustering**

The Search-based Clustering includes several methods such as Tabu Search, Variable Neighborhood Search for Clustering, Al-Sultan's Method, J-means, Global k-means, and the well-known genetic algorithms.

Genetic algorithms were proposed by John Holland (**J.H. Holland, 1975 [41]**) in the 1970s. Genetic algorithms are adaptive heuristic search algorithms based on the evolutionary idea of natural selection and genetics. In general, the procedure of genetic algorithms includes the following steps: initialization, selection, crossover, and mutation (**D.E. Goldberg, 1989[42]**). They are commonly used to address optimization problems by an intelligent exploitation of the random search. The beauty of the genetic algorithms is its easiness of use and the limited mathematics knowledge necessary for their operation. The drawbacks of genetic algorithms lie in the fact that they cannot guarantee a global optimum, and cannot assure optimization response time.

Georgiou and Tsapatsoulis applied genetic based clustering method in model-based collaborative filtering approaches (**O. Georgiou and N. Tsapatsoulis, 2010 [23]**). The authors prove that there is no need to generate the disjoint clusters to all the data. By this method dense clusters appropriate for model-based collaborative filtering recommendation system are created and moreover, the created clusters are allowed to present overlaps to each other. Both solutions



are almost constant and independent of the number of clusters, which subsequently minimizes the time complexity.

### **3.1.2 Model-based Clustering**

Early works of model-based clustering methods were firstly introduced by Wolfe (**J.H. Wolfe, 1965[43]**) in the 1960s. Unlike the traditional clustering which determines the groups of objects, the model-based techniques attempt to identify the clustering patterns and optimize matching between the dataset and the model. Common model-based algorithms include Expectation-Maximization algorithm, Gaussian Clustering, model-based clustering, COOLCAT, and STUCCO.

In the recommendation research field, Cadez, Heckerman, and Meek et al. (**I. Cadez, D. Heckerman, C. Meek, P. Smyth, and S. White, 2003[44]**) proposed a mixture of Markov model-based clustering using the Expectation-Maximization (EM) algorithm to cluster users. The runtime of this algorithm scales linearly with the number of clusters and it is implemented easily to handle hundreds of user sessions in memory. Nilashi, Ibrahim, Ithnin, and Sarmin (**M. Nilashi, O.B. Ibrahim, N. Ithnin, and N.H. Sarmin, 2015 [45]**) proposed several new hybrid approaches that utilize Gaussian mixture models with EM algorithms for enhancing the prediction accuracy. EM algorithms are distance-based methods which can solve the clustering problem even when the data is incomplete. They assign a certain probability to each data instance for every cluster and begin with initial iterations in order to find the maximum like-hood estimation for the parameters (**M. Nilashi, O.B. Ibrahim, N. Ithnin, and N.H. Sarmin, 2015 [45]**). The algorithm is an iterative method for finding maximum likelihood or maximum a

posteriori (MAP) estimates of parameters where the model depends on unobserved latent variables (M. Verma, M. Srivastava, N. Chack, A.K. Diswar, and N. Gupta, 2012 [46]).

EM algorithm includes two steps: Expectation step and the Maximization step (G. McLachlan and T. Krishnan, 2007 [47]). This name was given by Dempster and Laird in 1977 on their paper Maximum likelihood from incomplete data via the EM algorithm. Where  $p$  is an unknown parameter of distribution, give a set  $y$  of observed data, let  $x$  be the vector of random variables. The idea behind the EM algorithm is that for the unknown  $x_1$  and  $x_2$ , knowledge of the underlying distribution  $f(x_1, x_2, x_3|p)$  can be used to determine for  $p$ . Then, let  $p^{[k]}$  indicate the estimate of  $p$  after the  $k$ th iteration,  $k=1, 2, 3 \dots n$ , the algorithm of EM consists of two steps as below (T.K. Moon, 1996 [48]):

The **Expectation step** computes the expected value of the  $x$  data using the current estimate of parameter  $p$  and  $y$  observed data.

$$x_1^{[k+1]} = E[x_1|y_1, p^{[k]}]$$

The **Maximization step** uses the data from the first step. The maximum-likelihood estimation is a means of estimating the parameters of a distribution based on observed data. Let  $\theta = [\theta_1, \theta_2, \theta_3, \dots, \theta_r]^T$  denote a set of values,  $\theta^{[k+1]}$  be that value of  $\theta$  which maximizes  $Q(\theta | \theta^{[k]})$  (T.K. Moon, 1996 [48]):

$$\theta^{[k+1]} = \arg \max_{\theta} Q(\theta | \theta^{[k]})$$

The EM algorithm consists of choosing an initial  $\theta^{[k]}$ , and then implements the E-step and the M-step successively until convergence.

### **3.1.3 Center-based Clustering**

In a center-based model, each cluster is represented by a central vector. This category includes methods such as k-means, x-means, k-harmonic means, mean shift, k-probabilities, k-prototypes, and k-modes algorithms, where we use the well-known k-means algorithm as an example (G.J. Gan, C.Q. Ma, and J.H. Wu, 2007 [31]).

K-means clustering is considered a dynamic clustering algorithm proposed by Macqueen in 1967 (J. MacQueen, 1967 [49]). The k-means algorithm was designed to process numerical data; each cluster has a defined center, called centroid or Mean. The basic steps require the selection of K points as the initial centroids and the assignment of data points to their closest centroids. In each recursive step, a re-calculation of the centroid of each cluster is initiated until the centroids remain stable. Being the most popular clustering algorithm, the K-means algorithm is efficient in large data sets and performs well on numerical data. However, there are some drawbacks such as its performance which depends on the initialization of the centers and the pre-selection of the K-value. Moreover, the K-means algorithm does not perform well on high-dimensional data.

#### **3.1.3.1 K-means Clustering**

K-means clustering is a popular machine learning algorithm that is extensively used on recommendation systems. With the volume of users, items, and profiles growing fast on commercial websites, the datasets are becoming less scalable for online recommendations and moreover, the sparse data lowers the recommendation quality. Therefore, many researchers have utilized and proved that K-means clustering algorithm is suitable for recommendation systems since it can adequately address the aforementioned problems.

The process of k-means algorithm can be divided into two stages: initialization stage and iteration stage (R. Salman, V. Kecman, Q. Li, R. Strack, and E. Test, 2011 [50]). In the first stage, the k-means algorithm assigns randomly the observations into k clusters. In the iteration step, the algorithm calculates each distance between observation and cluster centroid and assigns each observation into the nearest cluster.

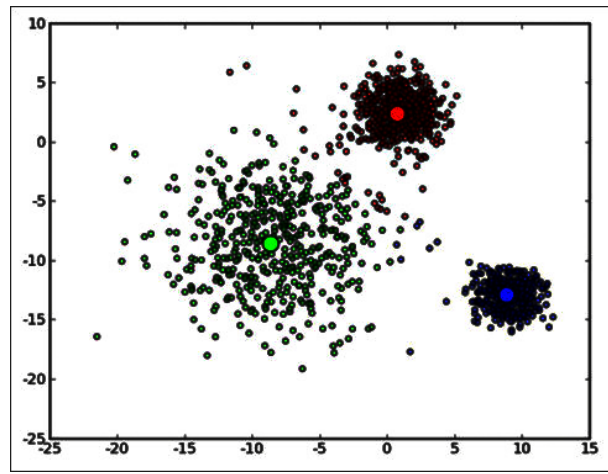


Figure 3. K-means Clustering (A.K. Jain, 2010 [39])

As we can see from the Figure 3, the clustering method aims to group similar objects into one cluster and effectively locates target objects to clusters similar to the object. Thus, the nearest neighbors in each cluster can be found easily since all objects in each cluster are similar to the target object, lowering the dimension of the data and saving processing time. The most used distance metrics are the Euclidean distance, Manhattan distance, and Minkowski distance. One of the most commonly used is the Euclidean distance, two multi-dimensional data points  $X = (x_1, x_2, x_3, \dots, x_n)$  and  $Y = (y_1, y_2, y_3, \dots, y_n)$  and which is described as follows(I.H. Witten, and E. Frank, 2005 [53]):

$$D(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \dots + (x_n - y_n)^2}$$

The k-means algorithm showed in Figure 4, starts by picking  $k$  (the number of clusters) and initializing clusters by selecting  $n$  points per cluster. Then, for each point, the algorithm places it in the cluster whose current centroid is nearest and computes the new mean for each cluster. Last step to loop the step two until points do not move between clusters and centroids. Although the k-means clustering algorithm is simple to implement, it has some drawbacks like it highly depends on the arbitrary selection of the initial centroids.

---

**Algorithm** K-Means clustering algorithm [39]

---

**Require:**  $D = \{d_1, d_2, d_3, \dots, d_n\}$  // Set of  $n$  data points.

**Set:**  $K$  number of clusters.

**Steps:**

1. Arbitrarily choose  $k$  data points from  $D$  as initial centroids;
2. **Repeat**
  - Assign each point  $d_i$  to the cluster which has the closest centroid;
  - Calculate the new mean for each cluster;

**End:** until convergence criteria is met.

---

*Figure 4. K-means Algorithm (A.K. Jain, 2010 [39])*

### 3.1.3.2 Optimal Cluster and Initial Centroids

Although the k-means algorithm has been used in several recommendation approaches, the algorithm is defined by three parameters:  $K$  number of clusters, initial clusters, and the distance metric. There does not exist an algorithm for selecting the optimal  $K$  value, however various heuristics can be used to provide a reasonable value. The initial clusters affect the final clusters due to the nature of the algorithm to converge upon the local minima (A.K. Jain, 2010 [39]). The usual approach to overcome this is to run the K-means, for a given  $K$ , with different initial

clusters which can be chosen randomly, dynamically, or based on the lower and upper bounds. Generally initial cluster center initialization algorithm are select randomly, afterwards I compare the results and choose the partition with the smallest squared error.

---

**Algorithm** Cluster Center Initialization Algorithm [51]

---

**Input:**  $D = \{d_1, d_2, \dots, d_n\}$  // set of  $n$  data, and items  $K$  // Number of clusters

**Output:** A set of  $K$  initial centroids.

**Steps:**

---

1. Set  $m = 1$ ;
2. Compute the distance between each data point and all other data points in the set  $D$ ;
- 3 Find the closest pair of data points from the set  $D$ . From a data point set  $A_m$  ( $1 \leq m \leq K$ ) which contains these two data points, delete these two data points from data set  $D$ ;
4. Find the data point in  $D$  that is closest to the data point set  $A_m$ , add it to  $A_m$  and delete it from  $D$ ;
5. Repeat step 4 until the number of data points in  $A_m$  reaches  $0.75 * (n/K)$ ;
6. If  $m < K$ , then  $m = m+1$ , find another pair of data points from  $D$  between which the distance is the shortest, form another data-point set  $A_m$  and delete them from  $D$ , Go to step 4;
7. For each data point set  $A_m$  ( $1 \leq m \leq K$ ) find the arithmetic mean of the vectors of data points in  $A_m$ , these means will be the initial centroids

---

**End**

---

*Figure5. Clustering Center Algorithm (K.A. Nazeer, and M.P. Sebastian, 2009 [51])*

Like most clustering algorithms, the K-means algorithm is efficient in large data sets and works efficiently on numerical data. Since a recommendation system has to process a massive dataset, the K-means cluster provides the ability for fast calculation of larger volume of variables, unlike hierarchal clustering. Moreover, it can produce tighter clusters than the ones of hieratical clustering and can work well in cases of the hyper-spherical shaped clusters, being also a

significant advantage over other clustering techniques. However, it is hard to compare the cluster quality provided by the K-means clustering. Another disadvantage is that different initial partitions may result in different final partitions since the choice of the initial clustering center has a direct effect on the clustering results. Finally, its performance depends on the center initialization does not work adequately on high-dimensional data.

## 3.2 Similarity Measures

In the field of recommendation systems, similarity methods are used to determine the relation between objects and are considered as the key step for determining the nearest neighbours for each object. The accuracy of the calculation impacts on the accuracy of the prediction since the recommendation results are generated from the nearest neighbours, proving that the similarity process is the core function of a recommendation system. Traditional similarity methods that have been used in the recommendation field include Pearson Correlation Coefficient, Distance Measures, and Cosine Similarity and adjusted Cosine Similarity.

### 3.2.1 Pearson Correlation Coefficient

The Pearson correlation is a measure of the linear degree between two variables X and Y with the coefficient lying between  $[-1, 1]$ . When the linear relationship between two variables increases, the correlation coefficient tends to reach 1 or -1. When there is a related increase in both variables, a positive correlation exists and the correlation coefficient is greater than zero. In the case of an increase and decrease to the two variables, respectively, a negative correlation exists and the correlation coefficient is less than zero.

Let  $R_{u,i}$  indicate the user  $u$  that has rated an item with rating  $i$ . The value  $\overline{R}_i$  denotes the

average rating of item  $i$ , and the user set  $U$  includes all the users who bought both items  $i$  and  $j$ . The similarity between item  $i$  and  $j$  can be calculated by the Pearson Correlation Similarity equation as follows (Q. Li, and B.M. Kim, 2003 [52]):

$$sim(i, j) = \frac{\sum_{u \in U} (R_{u,i} - \bar{R}_i)(R_{u,j} - \bar{R}_j)}{\sqrt{\sum_{u \in U} (R_{u,i} - \bar{R}_i)^2} \sqrt{\sum_{u \in U} (R_{u,j} - \bar{R}_j)^2}}$$

This equation is used to denote the item-based similarity calculation. In user-based approaches, the similarity calculation between user  $a$  and  $b$  can be expressed by the Pearson Correlation Similarity shown below (Q. Li, and B.M. Kim, 2003 [52]):

$$sim(a, b) = \frac{\sum_{i \in I} (R_{a,i} - \bar{R}_a)(R_{b,i} - \bar{R}_b)}{\sqrt{\sum_{i \in I} (R_{a,i} - \bar{R}_a)^2} \sqrt{\sum_{i \in I} (R_{b,i} - \bar{R}_b)^2}}$$

where  $i \in I$  denotes the rating sum from rated items by both users  $a$  and  $b$ ; and  $\bar{R}_a$  indicates the average rating of user  $a$ .

### 3.2.2 Cosine Similarity

The vector similarity method was firstly used in text mining and Natural Language Processing fields to calculate similarities between two documents, where every document can be represented by a vector based on the frequency of words (G.J. Gan, C.Q. Ma, and J.H. Wu, 2007 [31]). In recommendation, Cosine similarity can be used also for similarity computation. The item features can be considered as vectors in the user space and the similarity between two items is described as the cosine of the angle. It measures the inner product space of the cosine angle in order to calculate the difference between two or more vectors. A  $0^\circ$  angle means that the



two lines overlap in the same direction where a  $90^\circ$  angle denotes that their direction is completely dissimilar. A  $180^\circ$  angle means that they are in opposite direction.

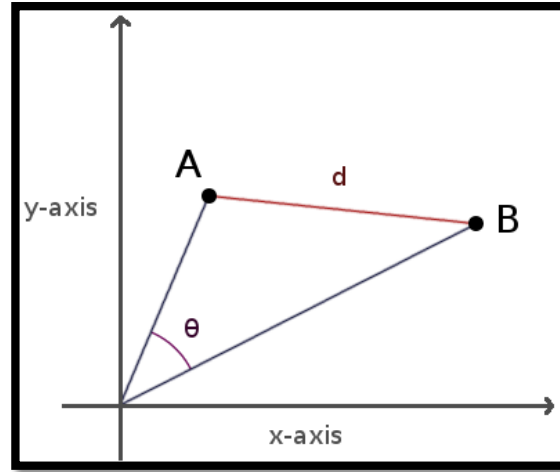


Figure 6. Cosine Similarity (I.H. Witten, and E. Frank, 2005 [53])

As illustrated above, the smaller the angle between two vectors, the higher the degree of similarity between them.

For  $I$  being the total item rating set from user, let  $I_{A,B}$  denote the total item rating set which has been rated by both users  $A$  and  $B$  with  $I_{A,B} = \{i \in I \mid R_{A,i} \neq \emptyset, R_{B,i} \neq \emptyset\}$ ,  $R_{A,i}$ ,  $R_{B,i}$  indicate all the ratings of user  $A$  and  $B$  for item  $i$ . The cosine similarity for user  $A$ ,  $B$  can be written as below (I.H. Witten, and E. Frank, 2005 [53]):

$$\text{Cosine similarity}(A, B) = \frac{\sum_{i \in I_{A,B}} R_{A,i} R_{B,i}}{\sqrt{\sum_{i \in I_{A,B}} R_{A,i}^2} \sqrt{\sum_{i \in I_{A,B}} R_{B,i}^2}}$$

The cosine similarity is a popular similarity measure as it considers the user ratings as vectors and is efficient to evaluate. However, one drawback lies in the fact that the difference in rating scale is not taken into consideration.

### 3.2.3 Adjusted Cosine Similarity

Since the traditional similarity measure does not consider the item rating variation by the different users, the recommendation accuracy might be deteriorated. The adjusted cosine similarity overcomes this by considering the rating scale from each user eliminating the rating difference from different users. Therefore, it can more accurately express the similarity between each item.

Let  $R_{A,i}$  be the rating of item  $i$  by user  $A$ .  $\bar{R}_A$  stands for the average rating of user  $A$  and the user set  $U$  includes the users who rated both items  $i$  and  $j$ . Then the adjusted cosine similarity of item  $i$  and  $j$  can be calculated as following (Q. Li, and B.M. Kim, 2003 [52]):

$$sim(i, j) = \frac{\sum_{u \in U} (R_{u,i} - \bar{R}_u)(R_{u,j} - \bar{R}_u)}{\sqrt{\sum_{u \in U} (R_{u,i} - \bar{R}_u)^2} \sqrt{\sum_{u \in U} (R_{u,j} - \bar{R}_u)^2}}$$

These advantages of the adjusted cosine similarity method overcome the traditional cosine similarity drawback by subtracting the average from each user which results in reduced rating preference.

### 3.2.4 Distance Measures

The basic concept in distance measures is to calculate the distance of two or more objects in order to understand their similarity, dissimilarity, and correlation. Typical methods in this category include Euclidean distance, Manhattan distance, Maximum distance, Minkowski distance, Mahalanobis distance, and average distance (G.J. Gan, C.Q. Ma, and J.H. Wu, 2007 [31]).

In the data mining and analysis fields, we can use the similarity measurement to determine the differences between objects, in order to estimate their similarity or category. The Minkowski distance is commonly used in such similarity measurements. The Minkowski distance is a metric in normed vector spaces which can be considered as a generalization of both Manhattan distance and Euclidean distance (A. Herrero, J. Sedano, B. Baruque, H. Quintián, and E. Corchado, 2015 [54]). For two datasets  $X$  and  $Y$ , with  $X = \{x_1, x_2, x_3 \dots x_n\}$  and  $Y = \{y_1, y_2, y_3 \dots y_n\}$ , the Minkowski similarity of  $XY$  can be defined as (I.H. Witten, and E. Frank, 2005 [53]):

$$sim(X, Y) = \sqrt[q]{\sum_{i=1}^n |x_i - y_i|^q}$$

where  $n$  indicates the  $X$  and  $Y$  dimensions, with  $x_i$ , and  $y_i$  the points of dimension  $i$ . For  $q=1$ , the equation calculates the Manhattan distance, and for  $q=2$  it calculates the Euclidean distance.

### 3.3 Neighborhood Selection

After the similarity computation between objects, how many of nearest-neighbors to select for recommendation can serious impact the quality of recommendation system. In recommendation system the neighbor selection normally include two steps: 1) an overall filtering step where only the most similar candidates are kept, and 2) a per prediction step which chooses the best candidates for this forecast (P. Melville, and V. Sindhvani, 2011[38]).

#### 3.3.1 Pre-filtering of neighbors

There are several popular ways in pre-filtering of similar neighbors like Top-N, Threshold

method, and Negative filtering (P. Melville, and V. Sindhwani, 2011[38]). In the large scale data environment that the large matrix contains millions of value, usually it is very expensive to load and store the data into memory for similarity process. More, it is extremely wasteful as the only propose of these values are used for predictions. Therefore, the above listed method could reduce the amount of similarity weight has to store in memory and it save the cost and time as well.

**Top- $N$  filtering:** During the recommendation generation that for either user or item only a list of the  $N$  nearest-neighbors and their respective similarity weight will be kept (P. Melville, and V. Sindhwani, 2011[38]). To increase the recommendation efficiency and accuracy, the  $N$  should be chosen carefully. Therefore, too large or too small  $N$  will result in large number of neighbors holding memory with slow process speed and less value in calculation reduces the recommendation coverage.

**Threshold filtering:** Instead of using a fixed number of neighbors, the threshold method keeps all the neighbors whose similarity weight has a magnitude greater than a given threshold  $V_{min}$  (P. Melville, and V. Sindhwani, 2011[38]). This method is more flexible than Top- $N$  method since the most significant neighbors are kept. However, the value of the threshold  $V_{min}$  is difficult to determine.

**Negative filtering:** The negative rating correlations are less reliable than Top- $N$  and Threshold methods. Intuitively, this is because strong positive correlation between two users is a good indicator of their belonging to a common group (P. Melville, and V. Sindhwani, 2011[38]).

## 3.4 Prediction Computation

By using the results from previous optimal neighbour selection step the next rating prediction method utilizes neighbor's similarity results for an unrated item's prediction. These prediction results can be used in the final step where items with higher scores will be recommended to the users. Usually, two methods are commonly used including the weighted sum and regression (B. Sarwar, G. Karypis, J. Konstan and J. Riedl [8]).

### 3.4.1 Weighted sum prediction

Where user  $u$ , item  $i$ , computer prediction  $P_{u,i}$  by calculate the sum of the ratings given by  $u$  on the items similar to  $i$ . Each rating weighted by the similarity  $S_{i,j}$  between  $i,j$ . The weighted sum rating prediction formula sees below (B. Sarwar, G. Karypis, J. Konstan and J. Riedl, 2001 [8]).:

$$p_{u,i} = \frac{\sum_{total\ simialr\ item\ D,} (S_{i,D} * R_{u,D})}{\sum_{total\ simailr\ item,D} S_{i,D}}$$

While user-based methods rely on the users with similar taste to predict rating, and item-based method look at the ratings given to the similar items. In general, in large commerce database that has fewer items than users, in situation of dataset MovieLens with thousands of items but more users to recommended, may benefit more from item-based neighborhood methods.

### 3.4.2 Average prediction

The average prediction method was the most used and simplest method to implement. There are several method commonly used for average prediction including: user rating average, item-

rating average and overall average.

Let user  $u$ , item  $i$ ,  $r_{ui}$  denote item  $i$  has real rating from user  $u$ , the user average rating  $\bar{r}_u$  calculated below (F. Ricci, L. Rokach and B. Shapira, 2010 [5]):

$$\bar{r}_u = \frac{\sum_{i \in N(u)} r_{ui}}{\sum_{i \in N(u)} 1}$$

# Chapter 4. Proposed methodology

---

During the past years, many recommendation systems have been proposed with notable results. While collaborative filtering recommendation approaches have been proved quite successful in practice, the methods along with their learning process present several limitations. More precisely, the memory-based collaborative filtering presents fundamental issues related to the previously mentioned online scalability and limited recommendation quality. This thesis has proposed an intuitive solution to address the problem is by partition the entire dataset and predict the missing rates in the user-item matrix respectively.

Given the attributes in matrix, the rating prediction performs the desired similarity computation based only on the user or item information. Therefore, in case of sparse ratings, the quality of the recommendation could be deteriorated since an adequate amount of user or item information is required. Ma, King and Lyu (**H. Ma, I. King and M.R. Lyu, 2007 [55]**) have proposed an effective missing data prediction method for recommendation system. Sarwar, Karypis, Konstan and Riedl (**B. Sarwar, G. Karypis, J. Konstan and J. Riedl, 2001 [8]**) introduced clustering techniques into recommendation. Li and Kim (**Q. Li, and B.M. Kim, 2003 [52]**) have proposed hybrid recommendation methods based on item-based clustering. These

results could be considered promising but not adequate enough for modern recommendation systems. In this chapter, a novel collaborative filtering approach is proposed that entails clustering along with missing value prediction. To be more precise, clustering is used for dataset pre-processing. Similarity, between users and items, is exploited for determining whether user or item information should be used in missing ratings. In case of higher user similarity, the user information will contribute to the prediction; otherwise, prediction will be based on the item information. The proposed recommendation method adequately addresses the data sparsity and cold-start issues for new users or items, and presents an improved time performance as well.

Collaborative filtering recommendation approaches with clustering and missing value prediction require the following five steps showed in Figure 7: Item clustering, similarity pre-calculation, approach selection, missing value prediction, and recommendation calculation.

- Apply item clustering, group the items into  $k$  clusters.
- User and item similarity calculation for each cluster with results stored in different tables.
- Table similarity comparison for determining which method (user/item) presents the highest similarity scores.
- Missing value prediction for each user-item rating matrix using the chosen method in Step 3.
- Recommendation list generating.



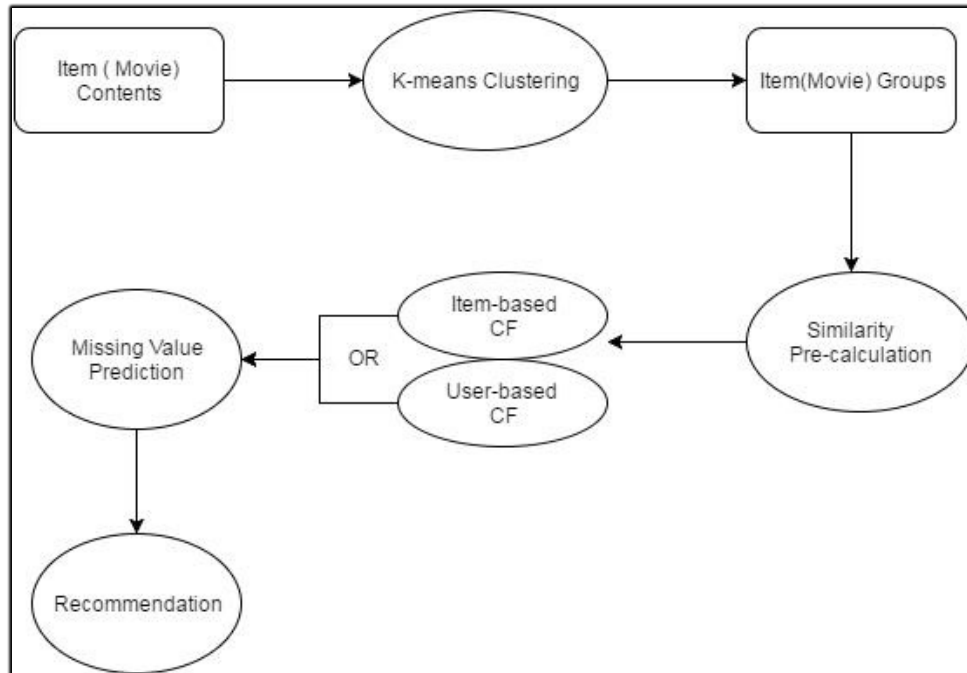


Figure 7. Recommendation Approach Flowchart

## 4.1 K-means Clustering Algorithm

Sarwar, Karypis, Konstan and Riedl (**B. Sarwar, G. Karypis, J. Konstan and J. Riedl, 2001 [8]**) introduced clustering techniques into recommendation systems in order to improve their time performance by scaling up the neighborhood formation in large-scale datasets for faster recommendation calculation. In this paper, the large item sets are clustered based on the item contents. Items in the same cluster usually share similar characters, thus the collaborative filtering approaches can improve the recommendation quality using the intra-similarity features of the same cluster. Moreover, the clustering method discards dissimilar items which could potentially affect the prediction accuracy. However, the recommendation quality is cluster dependent; therefore, accuracy might vary in different clusters.

Since the used experimental dataset includes more items than users, the clustering process is

item-based (movie) and performed offline. Utilizing the k-means algorithm, k number of clusters is initialized by selecting one point per cluster. Each point is then grouped in the cluster with the nearest centroid. After grouping all points, the centroids of the k clusters are updated reassigning all points to the new closest centroids. This process is repeated until all points remain in the same cluster for a number of steps.

---

**Algorithm** Item Clustering Algorithm

---

**Input:** *N records of Dataset D*

**Output:** *Initial Cluster set S*

---

```

(1) Randomly pick k items as the initial cluster
center.
(2) For (read each record  $p_i$  in dataset D)
    //Calculate the point density of record.
(3) Order the dataset D in descending order based
on the point density size.
(4) While ( Read unassigned records in dataset D)
    {
        //Read the current record;
        If ( this record is read twice
            //Put the record into the set S;
        Else {
            Calculate the distance of unassigned records
between each initial cluster center in the set S;
            //Store the minimum distance in the dist-min;
            If (dist-min < minimum distance between
records in set S)
                If (the number of record in set S<k)
                    //Added the current record into set S;
                Else if( the number of record in set S =k)
                    // Repeat the steps, until the center is not
changed;
        }
    }

```

---

*Figure 8. Item Clustering Algorithm*

After clustering, all the items have been assigned into  $k$  clusters  $C, D = \{C_1, C_2, C_3 \dots C_{k-1}, C_k\}$ . However, as mentioned in the previous part, it is difficult to determine the value  $k$  in advance and the choice of  $k$  affects the final recommendation results. In order to determine the optimal value  $k$ , a trial and error process will be used to determine the favorable cluster value.

## 4.2 Similarity calculation

The similarity measurement is a key step in a recommendation system since its value affects the accuracy of the rating prediction and the recommendation results. In traditional user-based or item-based collaborative filtering approach, the similarity calculation is based solely on past user ratings with the sparse user-item matrix, which directly leads to low prediction accuracies. Ma, King and Lyu (**H. Ma, I. King, and M.R. Lyu, 2007 [55]**) have proposed an effective missing data prediction method, which considers both the user and the item rating history in order to enhance the prediction accuracy in data sparsity cases. In this paper, due to the limitation of user-item ratings or items rated by limited users, false recommendation results are likely to be generated. The proposed framework suggests an effective missing value prediction method which analyzes the available rating information in Table 1 and determines the best missing data process for optimal results. Its fundamental novelty against similarity calculation of conventional user-based and item-based collaborative filtering approach is its row-wise user-based similarity calculation in the user-item rating matrix and its column-wise item-based similarity calculation (**B. Sarwar, G. Karypis, J. Konstan and J. Riedl, 2001 [8]**).

As discussed in the Chapter 3, three common similarity methods are used including the Pearson Correlation, the Cosine similarity and the adjusted Cosine similarity. In this paper, the adjusted cosine similarity of item  $i$  and  $j$  used to find out the similarity and the equation as

following:

$$similarity(i, j) = \frac{\sum_{u \in U} (R_{u,i} - \bar{R}_u)(R_{u,j} - \bar{R}_u)}{\sqrt{\sum_{u \in U} (R_{u,i} - \bar{R}_u)^2} \sqrt{\sum_{u \in U} (R_{u,j} - \bar{R}_u)^2}}$$

Let  $R_{u,i}$  and  $R_{u,j}$  denotes the rating of item  $i$  and  $j$  rated both by user  $u$ ,  $\bar{R}_u$  stands for the average rating of user  $u$  and the user set  $U$  includes all the users from dataset.

	$i_1$	$i_2$	$i_3$	$i_4$	$i_5$
$u_1$	3				
$u_2$				3	
$u_3$					
$u_4$		2			5
$u_5$	4				
$u_6$					
$u_7$					4
$u_8$	5		1		
$u_9$					
$u_n$				2	

Table 1. The User- Item Rating Matrix ( $m \times n$ )

### 4.3 The Off-line Process

The datasets in business environment are usually very large and the running cost is extremely high. Therefore, some off-line process is necessary. In this thesis, the off-line K-means clustering and similarity measures are pre-processed and the results are stored for future calculation in order to save time and cost. The off-line process can be run every day or every hour depending on the business requirement.

## 4.4 Similarity Measure Selection

In user-item rating matrix, each row represents one user and each column states one item. When the traditional algorithms produce the recommendation process, they only consider one approach (row or column) at a time. However, the dataset is sparse and the ratings are distributed randomly. Therefore, a process to compare the similarity between user and item is carried out. The proposed method calculates the similarity based on both item and user data. Then the higher similarity is used in the missing value prediction process.

## 4.5 Prediction Computation

The fourth step in rating prediction utilizes similarity results for an unrated item's prediction. These results will be used in the final step where items with higher scores will be recommended to the users. For this step, two methods are commonly used including the weighted sum and regression (B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, 2001 [8]). To reduce the proposed algorithm's complexity and increase its efficiency, the weighted sum method was chosen. Hence, the prediction is scaled by the sum of the similarity terms, ensuring that the prediction results lie within the predefined range. Table 2 shows that the user-item rating matrix has been filled up with predicted rating  $P_{m,n}$ .

1. As discussed previously, the similarity results between items will be sorted from top to low.
2. Set user  $u$ , item  $i$ , compute the prediction  $P_{u,i}$  on an item  $i$  for a user  $u$ : by calculating the sum of the ratings given by  $u$  on the items similar to  $i$ . Each rating weighted by the similarity  $S_{i,j}$  between  $i,j$ . The weighted sum rating prediction formula is provided below:

$$p_{u,i} = \frac{\sum_{total\ similar\ item\ D, (S_{i,D} * R_{u,D})}{\sum_{total\ similar\ item, D} S_{i,D}}$$

3. Based on the prediction results, the top score items will be delivered to the user.

	$i_1$	$i_2$	$i_3$	$i_4$	$i_m$
$u_1$	R3	P <sub>2,1</sub>	P <sub>3,1</sub>	P <sub>4,1</sub>	P <sub>m,1</sub>
$u_2$	P <sub>1,2</sub>	P <sub>2,2</sub>	P <sub>3,2</sub>	R3	P <sub>m,2</sub>
$u_3$	P <sub>1,3</sub>	P <sub>2,3</sub>	P <sub>3,3</sub>	P <sub>4,3</sub>	P <sub>m,3</sub>
$u_4$	P <sub>1,4</sub>	R2	P <sub>3,4</sub>	P <sub>4,4</sub>	R5
$u_5$	R4	P <sub>2,5</sub>	P <sub>3,5</sub>	P <sub>4,5</sub>	P <sub>m,5</sub>
$u_6$	P <sub>1,6</sub>	P <sub>2,6</sub>	P <sub>3,6</sub>	P <sub>4,6</sub>	P <sub>m,6</sub>
$u_7$	P <sub>1,7</sub>	P <sub>2,7</sub>	P <sub>3,7</sub>	P <sub>4,7</sub>	R4
$u_8$	R5	P <sub>2,8</sub>	R1	P <sub>4,8</sub>	P <sub>m,8</sub>
$u_9$	P <sub>1,9</sub>	P <sub>2,9</sub>	P <sub>3,9</sub>	P <sub>4,9</sub>	P <sub>m,9</sub>
$u_n$	P <sub>1,n</sub>	P <sub>2,n</sub>	P <sub>3,n</sub>	R2	P <sub>m,n</sub>

Table 2. The Rating Matrix after Missing value Prediction

## 4.6 Neighbours Determination and Selection

Finding the most appropriate objects / neighbours is not only challenging but also critical for the prediction step in recommendation systems. If the selected neighbours are not optimal, poor recommendation results might be produced. In this thesis, Top- $N$  filtering method has been used for neighbor selection and determination. During the recommendation process that for either user or item only a list of the Top- $N$  nearest-neighbors and their respective similarity weight will be kept (P. Melville, and V. Sindhwani, 2011[38]). More, a trial and error method is exploited to determining the best number of neighbours which contribute in the recommendation results.

## 4.7 Methodology Implementation

Python is a great object-oriented, interpreted, and interactive programming language. More, Python is a scripting language that combines remarkable power with very clear syntax. It has included the modules, classes, exceptions, high level dynamic type system and automatic memory management. There are available to many system calls and libraries, as well as to various GUI systems (**Python, [56]**). This open-source language provided a large library to support recommendation system, and data mining tasks.

R programming is widely used in academia and industry for statistical computation and graphics, developed in Bell Labs by John Chambers and his colleagues (**R. Gentleman, W. Huber, and V.J. Carey, 2011[57]**). The R framework is an integrated software suite that can be easily extended with 7000+ supported packages available on CRAN. The suite is oriented for data manipulation, graphics display, and statistical calculation providing a wide variety of data mining statistical tools like clustering, classification, time series analysis, linear and non-linear modeling. Effective data manipulation is possible using libraries explicitly developed for array and matrix calculations and graphic data analysis (**R. Gentleman, W. Huber, and V.J. Carey, 2011 [57]**).

The overall proposed methodology recommendation platform was mainly built in Python with its coding and development. Some additional R functions were coded to facilitate the analysis and computational steps described in this research.

*User-item-matrix.r*, this file utilizes the reshape2 and Matrix library for R. The function first loads the data file into the system, reformats it and converts its values to numeric type. The data is then saved to the user-item matrix.

*Evaluation.py*, this file calls the related function to calculate the MAE, RMSE, recall, precision and F1 measure to verify recommendation results in different situations and determine their variations.

*Estimate.py*, this file use to stores the data array, average data array, user mean, and item mean.

*Main.py*, this is a main file does the most functions for proposed recommendation approaches, which including similarity calculation, missing value prediction, set up k value, list length, neighbours size and matrix density.

*Nonclustering.py and Cofilter.py*, this files does the traditional method calculation, calls the calculated data array to performs the comparison between user and item's similarities for evaluating the best score to be used in the missing value prediction. Moreover, it generates the results used to compare with the proposed method.

*Pic.py*, this file calls all the calculate functions to print the results and draw the pictures.



# 5. Results and Discussion

---

This chapter gives in-depth analysis of the K-means clustering technology and Cosine similarity calculation designed to implement the proposed recommendation methodology. Several experiments are conducted aiming at measuring the quality of the proposed recommendation approach against the other start-of-the-art methods.

The results are presented through the following questions:

- How does the k-means clustering process affect the recommendation result?
- How does the number of neighbors affect the recommendation quality?
- Does the percentage of missing value predicted affect the final result?
- What is the time complexity difference between the proposed method and the traditional methods?
- What is the Recall, Precision, and F1 score of the final recommendation result?

In the experiment R functions were coded to facilitate the data analysis and Python functions

were developed for clustering, similarity computation, recommendation generation, and result evaluation. In order to obtain the accurately results, the five-fold cross validation method is applied in the evaluation process.

## 5.1 Data Overview

The MovieLens 100k dataset used for evaluation process were obtained from GroupLens Research Lab in the Department of Computer Science at the University of Minnesota (<https://grouplens.org>). The dataset has been released to the public for many years and is becoming a popular dataset used in the recommendation literature. It uses five-star rating scale containing ratings from one to five: 1. Poor, 2. Fair, 3. Good, 4. Very Good, 5. Excellent. The dataset has six main files including u.data, u.info, u.item, u.genre, u.user, and u.occupation.

The u.item and u.genre files are combined and shown in Table 3. There are 24 attributes in total representing the information about movies. Those attributes include movie id, movie title, release date, video release date, IMDb URL, and 19 genres of each movie such as unknown, Action, Adventure, Animation, Children's, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, Musical, Mystery, Romance, Sci-Fi, Thriller, War, and Western. One in the movie genre cell indicates that the movie belongs to that genre and 0 otherwise.

movie id	movie name	release date	video release date	IMDB URL	unknow	...	Action	Adventure	Animation
1	Toy Story(1995)	01-Jan-95	0	<a href="http://us.imdb.com/">http://us.imdb.com/</a>	0	...	0	0	1
2	GoldenEye(1995)	01-Jan-95	0	<a href="http://us.imdb.com/">http://us.imdb.com/</a>	0	...	1	1	0
3	Four Rooms(1995)	01-Jan-95	0	<a href="http://us.imdb.com/">http://us.imdb.com/</a>	0	...	0	0	0
4	Get Shorty(1995)	01-Jan-95	0	<a href="http://us.imdb.com/">http://us.imdb.com/</a>	0	...	1	0	0
5	Copycat(1995)	01-Jan-95	0	<a href="http://us.imdb.com/">http://us.imdb.com/</a>	0	...	0	0	0
6	Shanghai Triad(1995)	01-Jan-95	0	<a href="http://us.imdb.com/">http://us.imdb.com/</a>	0	...	0	0	0
7	Twelve Monkeys(1995)	01-Jan-95	0	<a href="http://us.imdb.com/">http://us.imdb.com/</a>	0	...	0	0	0
8	Babe(1995)	01-Jan-95	0	<a href="http://us.imdb.com/">http://us.imdb.com/</a>	0	...	0	0	0
9	Dead Man Walking(1995)	01-Jan-95	0	<a href="http://us.imdb.com/">http://us.imdb.com/</a>	0	...	0	0	0

Table 3. *u.item* and *u.genre*

The *u.user* and *u.occupation* files contain the demographic information about each user and their occupation information. The two files are combined and the sample data are provided below in Table 4, which includes user id, age, gender, occupation, and zip code.

user id	age	sex	occupation	zipcode
1	24	M	technician	85711
2	53	F	other	94043
3	23	M	writer	32067
4	24	M	technician	43537
5	33	F	other	15213
6	42	M	executive	98101
7	57	M	administrator	91344
8	36	M	administrator	5201

Table 4. *u.user* and *u.occupation*

The *u.data* file contains four attributes: user id, movie id, rating, and timestamp. The sample data are provided in Table 5. Each row shows one movie rating by one user. This is the most

important file used to compose the matrix in the later experiment analysis.

(100000, 4)				
	user_id	movie_id	rating	unix_timestamp
0	196	242	3	881250949
1	186	302	3	891717742
2	22	377	1	878887116
3	244	51	2	880606923
4	166	346	1	886397596

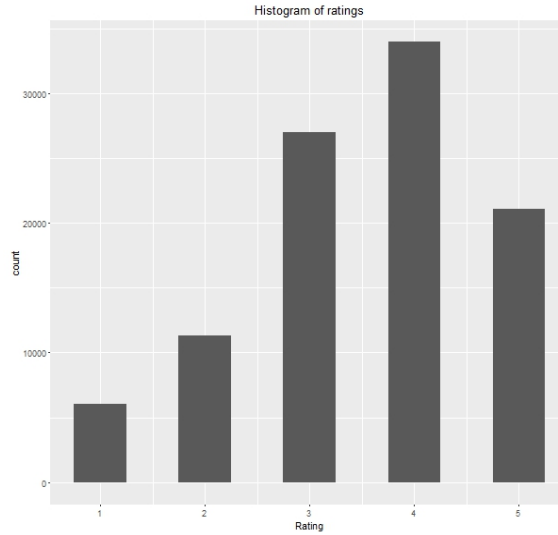
*Table 5. Rating table u.data*

The u.info file shows that the full u.data dataset has 100,000 ratings rated by 943 users on 1682 movies. Each user rated more than 20 movies in average in the dataset. The density of the user-item matrix is computed as below:

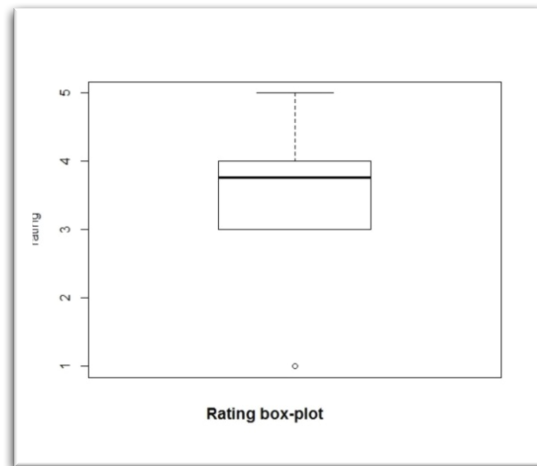
$$\text{Density: } \frac{100000}{1682 \times 943} = 6.3\%$$

### 5.1.1 Exporting the Rating data

According to the documentation, ratings marked as 0 indicate missing value. 100,000 ratings in total can be plotted using R language ggplot2 package. The following Figure 9 (a) and (b) show the distribution of the user ratings. Figure 9a shows the majority of ratings are above two, and Figure 9b shows the most common rating is between three and four, and the mean is 3.53 (O.R. Zaiane, 2002 [58]).



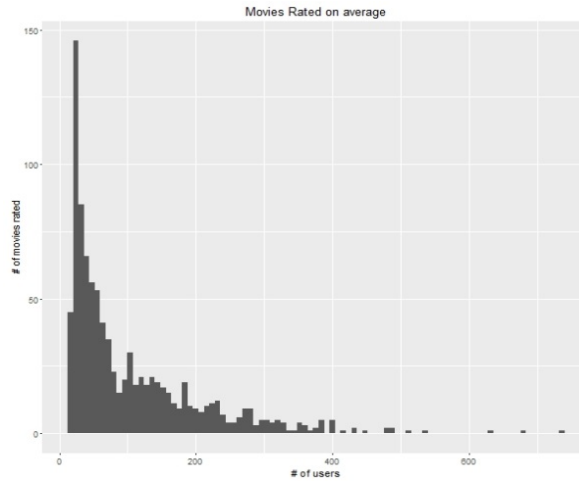
*(a) Rating Histogram*



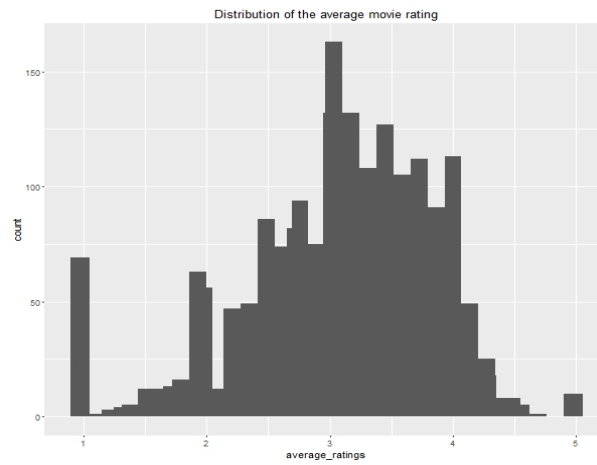
*(b) Rating box-plot*

*Figure 9, (a), (b) Rating Distribution*

Figure 10(a) clearly shows that most movie ratings are contributed by the first 200 users. Figure 10(b) implies that few movies were rated one star or five stars, and the common rating range is between three and four stars.



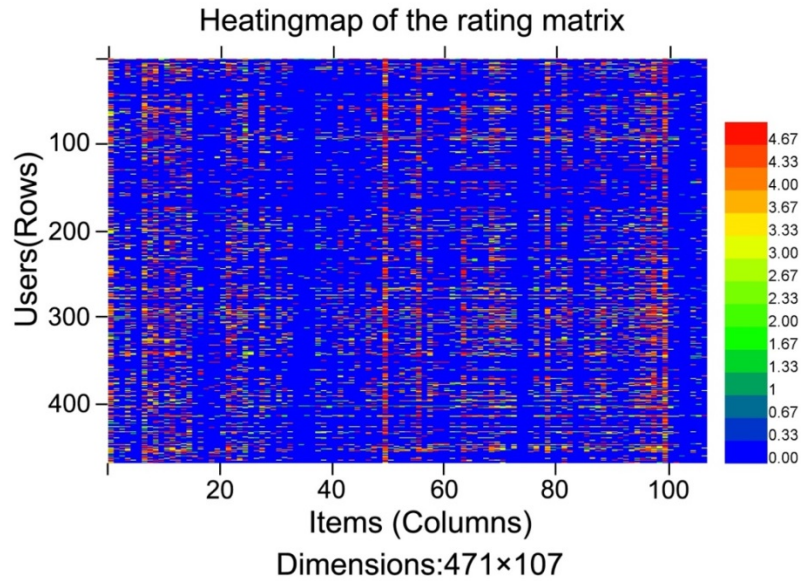
(a) Movie rated by users



(b) Average movie rating

Figure 10 (a), (b) Average Rating

The user-item rating matrix has 1664 rows and 943 columns. Each row of the matrix represents a user, each column corresponds to a movie, and each cell represents the rating.



*Figure 11. Heat-map of rating Matrix*

As the full dataset is too large to display and the dataset is too sparse. From Figure 11 we can see the data randomly split from full dataset. The heat-map has data dimension 471 x 107. The blue color indicates the missing value and the red color states the highest rating score. Generally speaking, the heating-map is sparse where the blue color is dominant. Moreover, the blue rows or columns represent that users or items receive low ratings or no rating, while the red lines represent the high-rated movies.

## 5.2 Evaluation Metrics

In the evaluation stage, the recommendation accuracy will be evaluated by five metrics including Mean Absolute Error (MAE), Root Mean Square Error (RMSE), Recall, Precision and F1 score based on 14 evaluation files which have been split into 80% training and 20% testing from u.data file. The MAE and RMSE are the most commonly used evaluation metrics in recommendation field. In literature, Chou, Yang, and Lin (**S.Y. Chou, Y.H. Yang, and Y.C. Lin,**

2015 [59]) used RMSE to evaluate music recommendation in a real-world setting. Zheng, Mobasher, and Bruke (Y. Zheng, B. Mobasher, and R. Burke, 2015 [60]) used RMSE to evaluate their novel Java-Based Context-aware Recommendation algorithms. Choi, Oh, Kim, and Ryu (I.Y. Choi, M.G. Oh, J.K. Kim, and Y.U. Ryu, 2016 [61]) used MAE to evaluate the collaborative filtering with facial expressions for online video recommendation. Moreover, Wu, Zhang, Lu (D. Wu, G. Zhang, and J. Lu, 2015 [62]) used MAE to evaluate the fuzzy preference tree-based recommender system for the personalized Business-to-Business E-services as well.

### 5.2.1 MAE and RMSE

MAE method calculates the average error between prediction values and the actual values in order to measure the recommendation quality where  $p_{u,i}$  indicates the predicted rating for user  $u$  on item  $i$ ,  $r_{u,i}$  denotes the real rating, and  $N$  is the total number of rating in the testing dataset. The MAE is defined as below (F. Ricci, L. Rokach, and B. Shapira, 2010 [5]):

$$MAE = \frac{\sum_{u,i} |p_{u,i} - r_{u,i}|}{N}$$

RMSE measures the square error between the predicted values and the actual known values. The predicted rating  $p_{u,i}$  is predicted by user  $u$  on item  $i$ ,  $r_{u,i}$  is the real rating and  $n$  is the total number of ratings. Smaller RMSE indicates better recommendation result. The RMSE is defined as below (F. Ricci, L. Rokach, and B. Shapira, 2010 [5]):

$$RMSE = \sqrt{\frac{1}{n} \sum_{u,i} (p_{u,i} - r_{u,i})^2}$$

Moreover, RMSE method has been used by Netflix Prize competition to evaluate the



recommendation result for those who can improve the result by 10% compared to the old CineMatch recommendation system. However, the RMSE evaluation method puts more emphasis on larger absolute error than MAE. The lower the RMSE is, the better the recommendation result is. Furthermore, some studies show that the rounding of the prediction results will reduce the MAE error if the rating scores are integers. In general, RMSE and MAE are easy to understand and simple to calculate.

### 5.2.2 Precision, Recall and F1 Score

According to the former Amazon scientist Greg Linden (**G. Linden, 2009 [63]**), he indicated that the purpose of movie recommendation was to find out which movie users was most likely to be interested in, rather than predict what rating score would be given to the movie. Therefore, the Top- $N$  recommendation method is more in line with the actual movie recommendation evaluation.

The Top- $N$  recommendation generally provides a personalized recommendation list to users which can be evaluated by precision and recall measurement. The precision and recall measures are used to measure the items classified as good (true positive) or bad. Precision is the probability whether the truly classified records are relevant. Recall is the probability whether the relevant records are truly classified. F1 is a measure of recommendation accuracy, which combines both precision and recall into a single measure. F1 score can be defined as a weighted average of both precision and recall. The Precision, Recall and F1 are defined below: (**O.R. Zaiane, 2002 [58]**).

$$precision = \frac{tp}{tp + fp} = \frac{\text{good movies recommended}}{\text{all recommendations}}$$

$$recall = \frac{tp}{tp + fn} = \frac{\text{good movies recommended}}{\text{all good movies}}$$

$$F1 \text{ score} = 2 * \frac{Precision * recall}{precision + recall}$$

False negative (*fn*) means all the responses where the actual response is positive, but what the model predicted is negative. True positive (*tp*) means all the responses where the actual response is negative and what the model predicted is positive. False positive (*fp*) denotes all the responses where the actual response is negative, however what the model predicted is positive (O.R. Zaiane, 2002 [58]).

The MSE and RMSE metrics are too sensitive and application specific, thus they cannot provide a good estimate of whether the recommendation results are liked by users. Therefore, the Recall metric is used to evaluate the probability of a good recommendation over the total number of recommendations, and the Precision metric is used for measuring the truly good recommendations over the good recommendations.

### 5.3 Computing Environment

In the experiment, a Dell Laptop equipped with Quad-core Intel Core i7-4700MQ processor, 16 GB memory, 750 GB Hard disc, and Nvidia Geforce 765M with 2GB memory Graphic card is used. The functions are coded by Python 2.7 and R language 3.3.3.

### 5.4 Improving the Recommendation Quality

Experiments are conducted in this thesis in order to demonstrate the performance of the proposed missing value and clustering method. In addition, the experiments use a five-fold cross-

validation scheme, where the MovieLens dataset  $D$  is randomly split into five subsets  $D_u$ ,  $u = 1, 2, \dots, 5$ .

### **5.4.1 Missing Value Prediction**

When the traditional algorithms produce the recommendation process, they predict all the missing values in the matrix. However, the 100k MovieLens dataset is very sparse and more than half of the users have no rating records. Therefore, it is not efficient and accurate to use conventional way. Then, I consider carrying out a research on finding the optimal density of matrix which only certain percentage of missing values is predicted. Furthermore, the proposed method is concerned with both item and user data in order to determine the higher similarity used for the missing value prediction process.

To perform the task, the proposed method determines whether the user-based or the item-based method should be used. Since two similarities were calculated from both user and item method, the higher similarity value will be used to enable the proposed missing value prediction. The prediction experiment, starting from 5% of missing value prediction and increasing every 5% each run until 100% of the missing values, has been predicted in the user-item rating matrix.

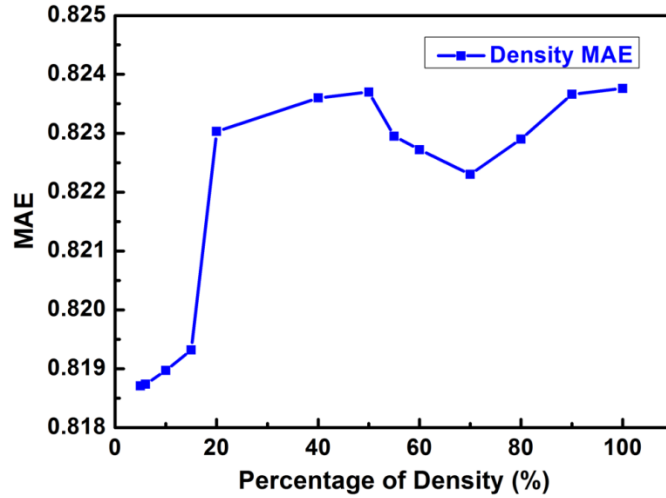


Figure 12. The MAE rate on varying matrix density

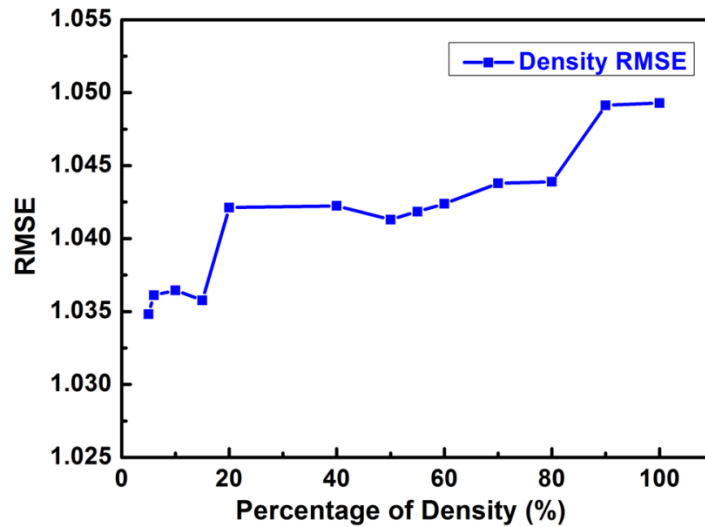


Figure 13. The RMSE rate on varying matrix density

As we can see from two graphs above, the general trend of MAE and RMSE are rising as the percentage of missing values predicted increases. In Figure 12, at the starting point with 5% missing value prediction, while the lowest point of MAE graph line is at 0.8191. When the MAE line steadily raises to 15%, it then boosts to 20% from 0.8191 to 0.8230. The rest of the MAE line reveals a general trend of up and down before it reaches the peak (100%). As indicated in

Figure 13, the starting point 5% is the lowest point at the RMSE graph line. The general trend of graph line is steady-going, while two section of line are increasing fast from 15% to 20% and from 80% to 90%. Generally speaking, two graphs show that both MAE and RMASE share the same optimal missing value percentage at 5%. That embraces with an obvious trend that as missing value percentage decreases, recommendation accuracy keeps improve.

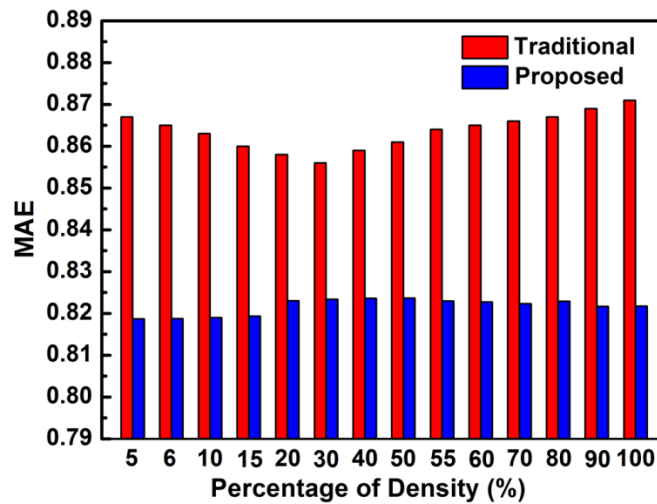


Figure 14. The Density MAE comparison

Figure 14 shows the comparison of the prediction qualities on the traditional non-clustered recommendation method and proposed method with various matrix density rates. It is obvious that the proposed method has better results in general for each density level. Moreover, compared with the best rate from traditional method, the optimal density level improves 4.5%.

As discussed previously, the hypothesis was to use the similarity method to fill the spare matrix for the purpose of boosting the recommendation quality. However, when the matrix density reaches a certain percentage, the accuracy curve initially rises and eventually drops due to the prediction error accumulation. Since the predicted values are not real values, the prediction formula can only show the similar values which are based on the user history data. As the density

increases, the error is accumulated as well. When the error reaches a threshold that affects the quality of the recommendation accuracy, the results are inferior. Accordingly, the optimal density has to be taken into consideration.

### **5.4.2 Neighbour Selection**

In order to improve recommendation accuracy, the optimal number of neighbors has been tested in this section. The experiments are conducted to demonstrate that the neighbor has a link with the recommendation performance. Finding the most appropriate neighbour is a challenging and critical task for overall recommendation performance.

To obtain the rate of MAE and RMSE on optimal number of neighbours, according to (**J.L. Herlocker and J.A. Konstan, 2002 [64]**), we select the range of k neighbours from candidate neighbors  $k \in \{5, 10, 20, 25, 30, \dots, 90\}$ . The trial and error method is applied to find the best neighbour size which will finally contribute to the recommendation results. The experimental test starts with 5 neighbours, and it increases 5 neighbours in each iteration. Meanwhile, the optimal density from pervious test will be applied in test.

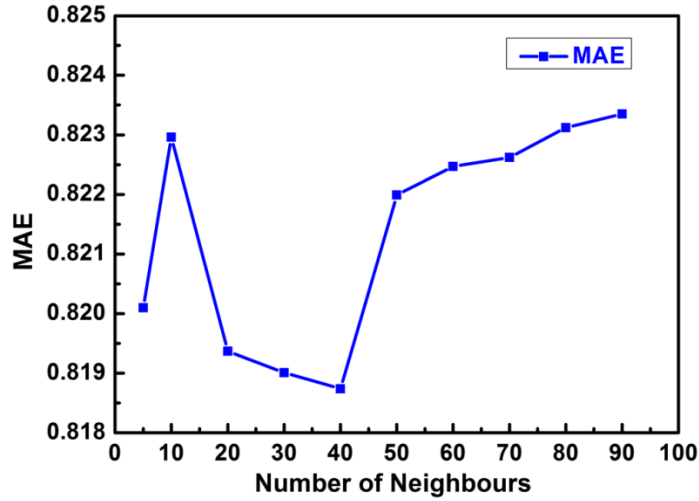


Figure 15. MAE rate on Optimal number of Nearest Neighbours

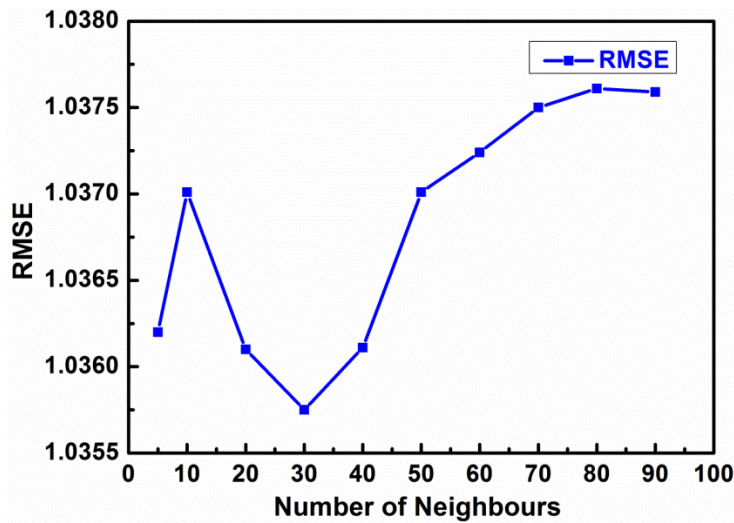


Figure 16. RMSE rate on Optimal number of Nearest Neighbours

From Figures 15 and 16, it can be read that MAE and RMSE are respectively 0.8201 and 1.0362 when the test started with 5 neighbors. Then, as the size increases to 10, both lines have a quick rise to 0.823 and 1.037. As the iteration process goes, when the number of neighbour steadily increases, while the curved line has a bottomed out at size 40 and has MAE rate 0.81847. In the meantime, the RMSE rate reaches the lowest point at size 30 and got rate 1.03575. When

the number of neighbours is equal to 50, both MAE and RMSE start to rise and keep stable until the highest point when the number of neighbours reaches 90.

The experimental results validate the hypothesis that the various sizes of neighbours affect the recommendation results. And more precisely, when the number of neighbours is between 30 and 40, the average MAE and RMSE lines reach their lowest points indicating the best results in neighbour selection process. With the neighborhood's increasing by size 40, MAE and RMSE graphs embrace with a general trend of steady increase.

## **5.5 Recommendation Online Scalability**

### **5.5.1 K-Means Clusters Selection**

This section examines how the k-means clustering method improves the online scalability. Since the MovieLens dataset includes more items than users, here I used item clustering method and performed approaches offline.

Since the dataset size includes only 943 users and 1682 items, the k value is tested in small range from 1 to 10. The optimal k is evaluated during the experiments by trial and error. The applied K-means method starts with  $k=1$ , and is increases one cluster in each iteration until ten clusters. Subsequently, the optimal neighbour  $n = 40$  from past experiments is used for neighbour selection. Finally, the recommendation results are compared to traditional non-clustered recommendation method.



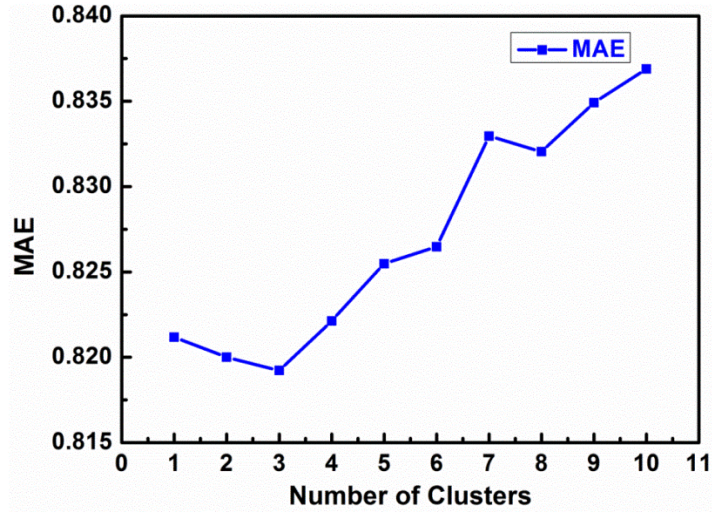


Figure 17. MAE of K-means clusters

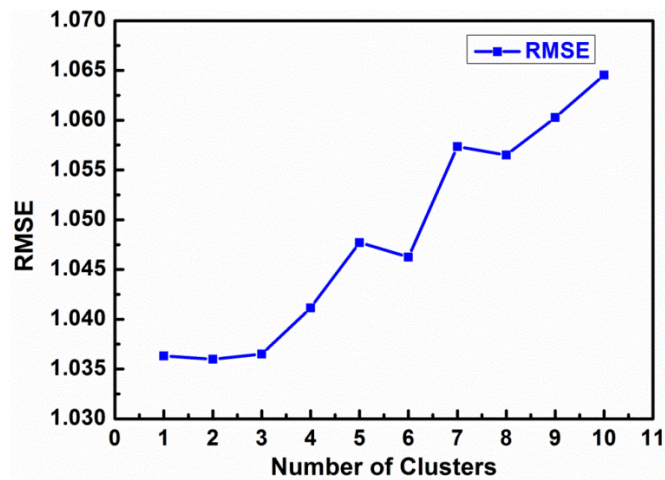


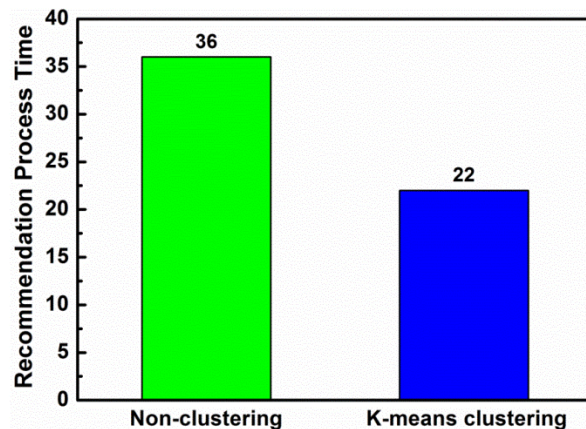
Figure 18. RMSE of K-means Clustering

The K-means clustering algorithm has been implemented and tested with python functions. From Figures 17 and 18, we can see the general trend of the graph lines first drops then rises as the number of clusters increases. When k=1 has applied to the dataset, the average of MAE is 0.821 and the average of RMSE is 1.036. As the k value increases while the MAE curved line has dropped to the lowest point when k = 3 where MAE is 0.819. In the meantime, RMSE drops to the lowest point while k = 2. When k value is larger than 3, the average MAE and RMSE are

progressively increased until to the top of curved lines which get rate 0.8362 and 1.065, denoting the lowest recommendation quality in the experimental test. By comparing the optimal k value to the other k vales in experiment, it can be observed a significant improvement to the recommendation quality.

It clearly shows that the recommendation accuracy is affected by the number of clusters. Starting from small number of k value and progressively increasing them, there is an initial drop followed by an increase. When k value is equal to 1, which generates one big cluster including large amount of data in cluster, it results in poor quality. When k increases to a proper size of two and three where items inside the same clusters were highly similar, it helps the missing value prediction and subsequently the recommendation quality. Then, as the k value keeps increasing, the available data in each cluster are limited and inadequate to make correct predictions as denoted by the MAE and RMSE metrics, contributing to a poor recommendation quality

## 5.6 Recommendation Time Efficiency



*Figure 19. K-means clustering compared to traditional method*

As discussed from pervious chapter, the recommendation system suffers from the scalability problem, especially in large-scale dataset. The time of generating recommendations with the high amount of missing values in dataset could not meet the online recommendation requirements. Therefore, the proposed method uses K-means clustering, which is effective by comparing the process speed with the traditional non-clustered method. From Figure 19, we find that the total speed of offline clustering plus recommendation process was improved by 163%. Since the offline k-means clustering pre-process includes all similar items into the same groups, the similarity process will be less time consuming. In such a way, the clustering process contributes to the overall time efficiency by speeding up the calculations and improving the online scalability allowing real-time recommendations.

## **5.7 Overall recommendation accuracy**

### **5.7.1 Recall and Precision**

Typically, the Recall and Precision are measured over the total number of recommendations list  $L$ , while a larger  $L$  could improve the recall results but likely reduce the precision. Therefore, this section is evaluating the recommendation approaches over a range of  $L$ , rather than using a fixed length.

From previous experimental test results, the optimal  $k$  clusters, neighbours and the matrix density of missing value prediction have been applied to the process. The list length has been tested by trial and error in a range from 50 to 500 items for observing the recall rate by increasing 50 items in each iteration. And the precision rate will be observed by trial and error with the list length in a range starting from 100 until 1000 items with increases 100 items in each iteration. The experimental results have compared to the traditional approach.

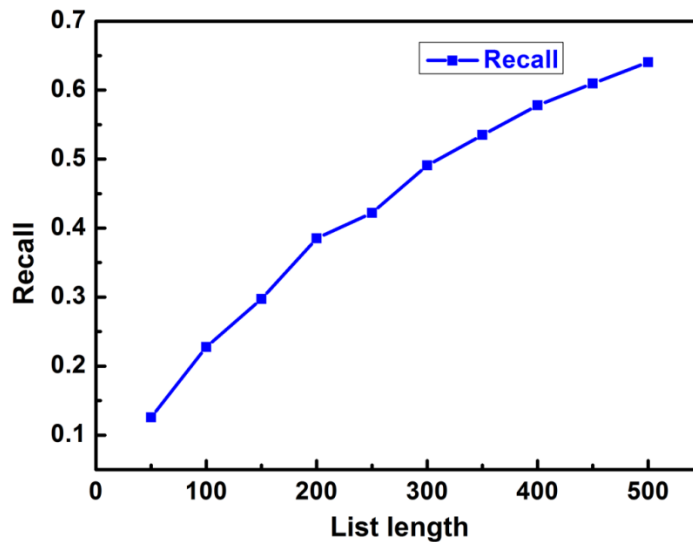


Figure 20. The recall rate of list range 50 – 500.

As we can see from Figure 20, the general trend of the recall curved line has risen perpendicularly. When  $L = 100$  items, the recall rate is 0.201, and for  $L = 500$  items the recall rate is 0.64. It clearly shows that the recall rate has rapid raise in list range  $[50,200]$  which has good recall rate and short list in both. Generally speaking, when the length of the recommendation list is increased, the recall rate has increased as well. When recommended with more items, the users are more likely to like one or several of them. However, the recommendation list cannot be excessively expanded, in that case, the information overload problem makes less meaning to the recommendation system. Thus, a quality recommendation system containing short list and high recall rate.

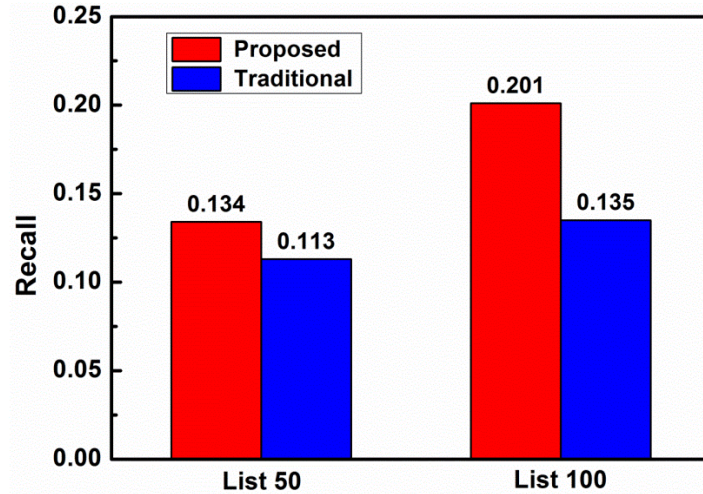


Figure 21. The Comparison of proposed method to Traditional method

From Figure 21, it shows the comparison of the recall rate from the proposed method along a traditional method. When  $L = 50$  the proposed method gets recall rate 0.134 with the traditional one being at 0.113, indicating an improved recall rate by 18.5%. With  $L = 100$  the recall rate has improved by 48.8%. This proves the superiority of the proposed method against to the traditional one.

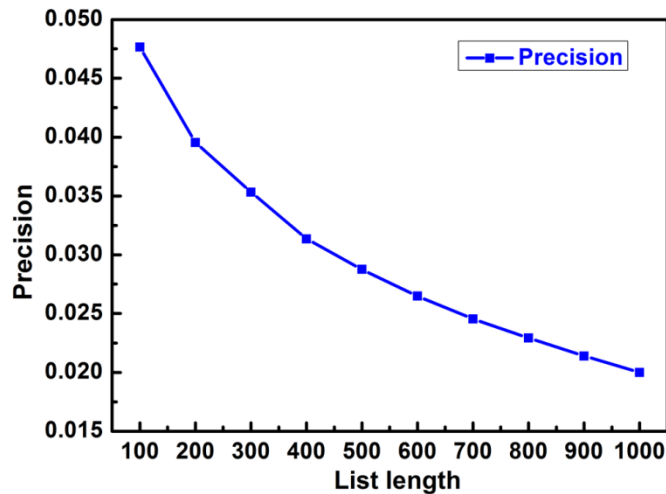
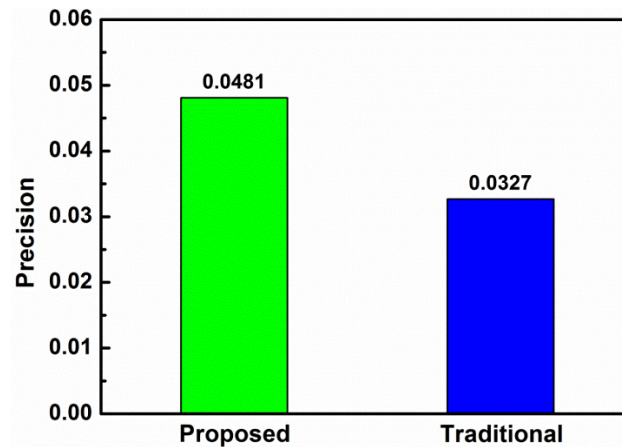


Figure 22. The precision rate of list range 100-1000

Theoretically, the recall rate has a negative correlation with the precision rate, that a longer list increases the recall rate while it reduces the precision rate at the same time. From the Figure 22, we can see the precision curved line with a list range from 100 -1000 items, when  $L = 100$  has rate 0.0481 which is on the highest point at the line. Then, the list size increases while the precision rate has persistently reduced. The range of precision rate is between 0-1, due to the fact that the experimental dataset does not have enough sample which makes the results smaller than usual, therefore, in this test the precision rate has been compared to the traditional method.



*Figure 23. The precision comparison on list 100*

From Figure 23, we can see the precision rate comparison of the proposed method with the traditional one. On best result length  $L = 100$ , the precision rate reaches its value 0.0481, and the traditional method has 0.0327, proving that the proposed method achieves a 47.1% improvement.

Based on the overall experimental tests, the proposed method presents a better recall and precision rate compared to the traditional method. Moreover, among several recommendation list

values, the number of list = 100 is suggested as the optimal length for use in recommendation algorithms.

### 5.7.2 The F1 Score

As we can see from the recall and precision experimental test, the proposed method can be improved on certain length of the result list. What is more, by comparing with the best recall and precision result with the traditional result, it shows good improvement to the recommendation results. To further evaluate the recommendation accuracy, this section takes recall and precision both into consideration. Thus, the F1 score has been tested and the result has been compared to the traditional method.

The experimental results from previous test has been applied. Afterwards, the list length has been tested in a range from 50 to 1000 items to observing the change of F1 score.

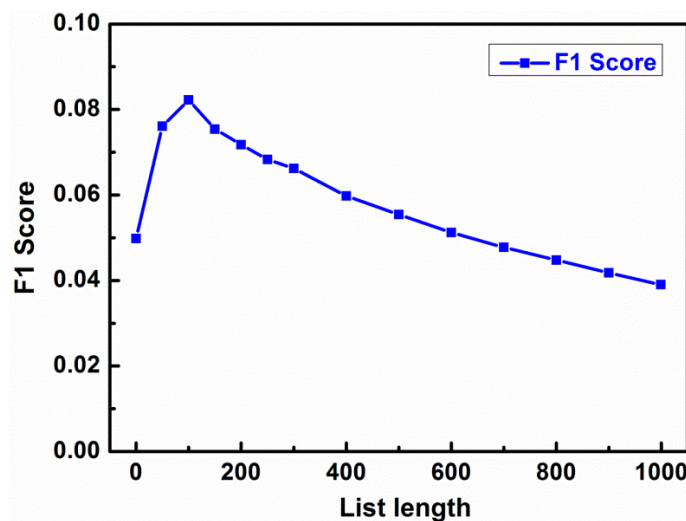
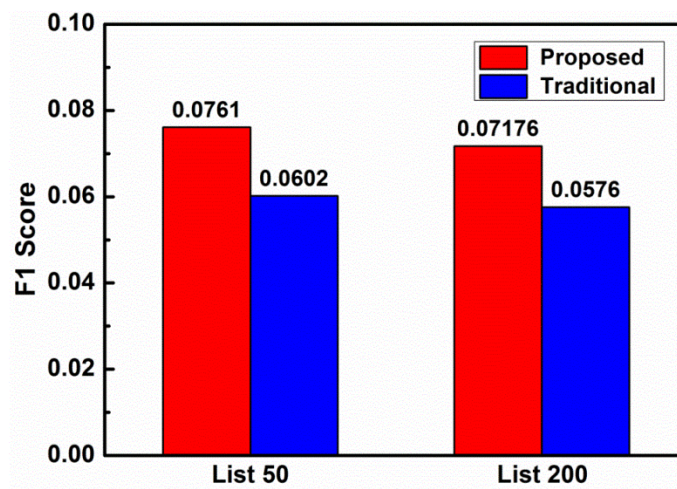


Figure 24. The F1 Score

The Figure 24 has showed the F1 graph for list range 50 – 1000, the curved line first raises to L = 100 which is the highest point, then it downward oscillations to L = 300, afterward the

curved line has dropped to  $L=1000$  reached the lowest point. From the graph, we can see the list range  $[50,200]$  keeps good F1 rate and manage the list in short range simultaneously. Meanwhile, by considering the results from previous section, I find the list range  $[50,200]$  is the optimal range for this dataset, which clearly shows a better result on recall, precision and F1 scores. Moreover, as we can see from Figure 25, the F1 scores comparison of the proposed method to the traditional method showed below.



*Figure 25. F1 score comparison to the traditional method*

In the optimal range  $[50,200]$ , the Figure 25 shows the proposed method has  $F1 = 0.0761$  on  $L = 50$ , and  $0.07176$  on  $L = 200$ , compared to traditional method which has  $L = 50$  has  $0.0602$  and  $L = 200$  has  $0.0576$  has improved 24% in average.

In this chapter, the k-means clustering and similarity prediction methods have been used to experiment and verified the factors which may affect the recommended accuracy, including the ratio of matrix estimation density, the number of neighbors, the k value of k-means clustering and the length of recommendation list.



The experimental results show that 1. The optimal matrix density at 5% of missing value predicted has the best recommendation accuracy. However, as perdition percentage increases while the final recommendation quality decreases due to the reason of perdition error accumulation. 2. The number of the optimal neighbours improved the recommendation accuracy, experiment shows that the MAE and RMSE graph line reaches the peak when the number of nearest neighbours is in range of 30 – 40. As the number continues increases while the results first raises then trending to smooth. 3. The offline clustering algorithm saves computation time while improves the recommendation efficiency and online scalability. When the optimal k value = 2 has the best recommended quality. However, by continuing to increase the k value results, the accuracy decreases as large number of clusters reduce the relevant data in each cluster. 4. The metrics including precision, recall rate and F1 score being used to measure the overall recommendation accuracy, and compared it with the traditional method, the proposed method has significant advantage and more than 20% overall improvements.

# Chapter 6. Conclusion

---

Typical challenges faced by traditional collaborative filtering recommendation system are information dataset sparsity, online scalability and time complexity problems which affect the recommendation quality and decrease the user's satisfaction. This thesis has presented a novel collaborative filtering approach which uses clustering technology, then predicts the missing value using similarity measure. The clustering and similarity measures are commonly used in data mining tasks. Meanwhile, it is suitable in recommendation system as well.

## 6.1 Conclusion

In the implementation of the clustering with missing value prediction approach, the algorithm compares the similarity between users and items to determine whether user or item information should be used to predict the missing ratings that showed improvements to the recommendation quality. The clustering method is partitioning the user-item rating matrix using K-means algorithm to identify similar items that grouped the most similar items (movies) into one cluster which address the data sparsity, online scalable problems and increase the time efficiency.

The proposed method is developed from traditional collaborative filtering combined with data mining technologies. The contributions of this thesis are provided below:

- The data mining technology K-means clustering is applied to the dataset grouping the similar objects into one cluster, which addresses the data sparsity problem. More, the similarity calculation will be processed in each cluster instead of the whole dataset, which could save a significant amount of time.

- An excellent and scalable recommendation system is able to provide the real-time recommendation to the users. Applying a clustering method could increase the system online scalability and decrease the processing time.

- This thesis uses the effective optimal missing value prediction, neighbours and length of list to the sparsity dataset. The pre-calculated similarity is used to compare and determine whether user-based or item-based criteria will be used for the rating forecast, which improves the prediction quality.

- According to the experiments by Python and R language, this thesis found out: 1. the result of missing value prediction method has better result compared to non-predicted dataset. 2. Prove that the clustering method could improve the recommendation speed and investigate how different k clusters affect the recommendation quality. 3. By overall evaluation in RMSE, Recall, Precision and F1 measure, the proposed method has 20% improvement compared to the traditional collaborative filtering method.

Moreover, from user and business side, this thesis presents a solution of collaborative filtering recommendation system which helps users to select the interesting items. The system supports the user in decision making process, increases the business sale amount, and strengthens

user's satisfaction and royalty. This approach could be applied to the recommendation system in large-scale business environment which dataset is sparse.

## **6.2 Future Work**

The proposed method which pre-processes the dataset based on clustering method; however, the k-means clustering has its own limitation on optimal clusters and its performance depends on the center initialization, which does not work adequately on high-dimensional data. Moreover, the missing value prediction method is based on the accuracy of how user and item similarity is computed. More research is needed to conduct on the relationship between user and item information. Therefore, the following work will be developed in future:

- To implement the approach in Apache hadoop framework which is suitable for very large data environment.
- To experiment different data mining algorithms, such as SVD, decision tree, deep learning, and other clustering algorithms. Compare and study how they improve the recommendation system.
- To test hybrid recommendation system that combine content-based recommendation system with collaborative filtering recommendation system , add or reduce data dimension by applying text mining processing techniques.

# Bibliography:

- [1] WorldWideWebSize.com (2016), "*The size of the World Wide Web (The Internet)*" [Online]. Available: <http://www.worldwidewebsize.com/>.
- [2] Cisco (2016), "*The Zettabyte Era—Trends and Analysis*" [Online] Available: <http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html>
- [3] G. Simos (2015), "*How Much Data Is Generated Every Minute on Social Media?*" *WeRSM | We Are Social Media*, [Online]. Available: <http://wersm.com/how-much-data-is-generated-every-minute-on-social-media>.
- [4] Web Scraping and Data Scraping (2015). "*Walmart Earnings Disappoint – How To Catch Up With Amazon*" [online]. Available at: <http://learn.scrapehero.com/walmart-earnings-disappoint-how-to-catch-up-with-amazon/>
- [5] F. Ricci, L. Rokach and B. Shapira, "*Recommender Systems Handbook*," Springer, pp.1-35, 2010.
- [6] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "GroupLens: An Open Architecture for Collaborative Filtering of Netnews", in *Proc. of 1994 ACM Conference on Computer Supported Cooperative Work*, Chapel Hill, NC, pp.175-186, 1994.
- [7] C. Anderson, "The Long Tail: Why the Future of Business is Selling Less of More," New York, NY: *Hyperion*. ISBN 1-4013-0237-8, 2006.

- [8] B. Sarwar, G. Karypis, J. Konstan and J. Riedl, "Item-Based Collaborative Filtering Recommendation Algorithms," in *Proc. of 2001 the 10th international conference on World Wide Web*, 2001.
- [9] J. Bennett and S. Lanning "The Netflix Prize," in *Proc. of KDD Cup and Workshop*, pp.35, 2007.
- [10] V, Zanardi, "Addressing The Cold Start Problem in Tag-based Recommender Systems," Doctoral Dissertation, University College London, UK, 2011.
- [11] D. Goldberg, D. Nichols, B.M. Oki, and D. Terry, "Using Collaborative Filtering to Weave an Information Tapestry," *Communications of the ACM*. no12, pp. 61-70, Dec. 1992.
- [12] J.S. Breese, D. Heckerman, and C. Kadie, "Empirical Analysis of Predictive Algorithms for Collaborative Filtering," in *Proc .of the Fourteenth conference on Uncertainty in artificial intelligence*, Morgan Kaufmann Publishers Inc, pp. 43-52,1998.
- [13] J.B. Schafer, J.A. Konstan, and J. Riedl, "E-commerce Recommendation Applications," In *Applications of Data Mining to Electronic Commerce*. Springer US, pp.115-153, 2001.
- [14] G. Linden, B. Smith, and J. York. "Amazon. com Recommendations: Item-to-Item Collaborative Filtering," *Internet Computing, IEEE* vol.7, no. 1, pp. 76-80, 2003.
- [15] A.S. Das, M. Datar, A. Garg, and S. Rajaram, "Google News Personalization: Scalable Online Collaborative Filtering," in *Pro. of the 16th International Conference on World Wide Web*, ACM, pp. 271-280, 2007.

- [16] S.K. Tiwari and S.K. Shrivastava, "An Approach for Recommender System by Combining Collaborative Filtering with User Demographics and Items Genres," *International Journal of Computer Applications, Volume 128- no.13*, pp.16-24, 2015.
- [17] R.R. Sinha, and K. Swearingen, "Comparing Recommendations Made by Online Systems and Friends," in *Proc. of ELOS Workshop: Personalisation and Recommender Systems in Digital Libraries, Vol. 1*. 2001.
- [18] J.A. Al-Sharawneh, and M. Williams. "Credibility-aware Web-based Social Network Recommender: Follow the Leader," in *Proc. of 2010 ACM Conference on Recommender Systems*, pp. 1–8, 2010.
- [19] X. Amatriain, N. Lathia, JM. Pujol, H. Kwak, and N. Oliver, "The Wisdom of the Few: A Collaborative Filtering Approach Based on Expert Opinions From the Web," in *Proc. of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp.532-539, 2009.
- [20] J.A. Konstan, B.N. Miller, D. Maltz, J.L. Herlocker, L.R. Gordon, and J. Riedl. "GroupLens: Applying Collaborative Filtering to Usenet News," *Communications of the ACM*. no3, pp. 77-87, Mar. 1997.
- [21] C. Smith, (2014), "Amazing Amazon Statistics," [online] DMR. Available at: <http://expandedramblings.com/index.php/amazon-statistics/>.
- [22] G. Linden, B. Smith, and J. York, "Amazon. com Recommendations: Item-to-Item Collaborative Filtering," *Internet Computing, IEEE* vol.7, no. 1, pp. 76-80, 2003.

- [23] O. Georgiou, and N. Tsapatsoulis, "Improving the Scalability of Recommender Systems by Clustering using Genetic Algorithms," in *Proc. of International Conference on Artificial Neural Networks*, Springer Berlin Heidelberg, pp. 442-449, 2010.
- [24] P.T. Braak, N. Abdullah and Y. Xu, "Improving the Performance of Collaborative Filtering Recommender Systems Through User Profile Clustering," in *Proc of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology-Volume 03*, IEEE Computer Society, pp. 147-150, 2009.
- [25] K. Babas, G. Chalkiadakis, and E. Tripolitakis. "You are What You Consume: A Bayesian Method for Personalized Recommendations," in *Proc. of the 7th ACM Conference on Recommender Systems*, ACM, pp. 221-228, 2013.
- [26] C.C. Aggarwal, J. L. Wolf, K. L. Wu, and P.S. Yu, "Horting Hatches an Egg: A New Graph-theoretic Approach to Collaborative Filtering," in *Proc. of the fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, pp. 201-212, 1999.
- [27] K. Zhou, SH. Yang, and H.Y. Zha, "Functional Matrix Factorizations for Cold-start Recommendation," in *Proc. of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 315-324, 2011.
- [28] G.R. Xue, C. Lin, Q. Yang, W. Xi, H.J. Zeng, Y. Yu, and Z. Chen, "Scalable Collaborative Filtering using Cluster-based Smoothing," in *Proc. of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ACM, pp. 114-121, 2005.



- [29] K. Yu, X. Xu, J. Tao, M. Ester, and H. Kriegel, "Instance Selection Techniques for Memory-Based Collaborative Filtering," in *Proc. of the Second Siam Intl. Conf. on Data Mining, SDM*, 2002.
- [30] A.I.M. Rashid, S.K. Lam, G. Karypis, and J. Riedl, "ClustKNN: A Highly Scalable Hybrid Model-& Memory-based CF Algorithm," in *proc. of WebKDD*, 2006.
- [31] G.J Gan, C.Q. Ma, and J.H. Wu, "Data Clustering: Theory, Algorithms, and Applications," *vol. 20*, Siam, 2007.
- [32] A. Fiasconaro, M. Tumminello, V. Nicosia, V. Latora, and R.N. Mantegna. "Hybrid recommendation methods in complex networks," *Physical Review E:92.1:012811*, 2015.
- [33] Y. Zou, A.J. An, and X.J. Huang, "Evaluation and Automatic Selection of Methods for Handling Missing Data." *Granular Computing, IEEE International Conference on* (Vol. 2, pp. 728-733), 2005.
- [34] Y. Liu, X.J. Huang, and A.J. An, "Personalized Recommendation with Adaptive Mixture of Markov Models." *Journal of the American Society for Information Science and Technology*, 58(12), pp.1851-1870, 2007.
- [35] J. Wei, J. He, K. Chen, Y. Zhou, and Z. Tang, "Collaborative Filtering and Deep Learning Based Recommendation System for Cold Start Items." *Expert Systems with Applications*, 69, pp.29-39, 2017.
- [36] A. Forestiero, "Multi-agent Recommendation System in Internet of Things." in *Proc. of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (pp. 772-775). IEEE Press, 2017.

- [37] H.J. Ahn, "A New Similarity Measure for Collaborative Filtering to Alleviate the New User Cold-starting Problem," *Information Sciences*, 178(1), pp.37-51, 2008.
- [38] P. Melville, and V. Sindhwani, "Recommender Systems," *Encyclopedia of machine learning*, Springer US, pp. 829-838, 2011.
- [39] A.K. Jain, "Data Clustering: 50 Years Beyond K-means," *Pattern Recognition Letters* 31, no.8, pp.651-666, 2010.
- [40] J.L. Schafer and J.W. Graham. "Missing Data: our view of the state of the art," *Psychological methods*, 7, no. 2, pp.147, 2002.
- [41] J.H. Holland, "Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence," U Michigan Press, 1975.
- [42] D.E. Goldberg, "Genetic Algorithms in Search, Optimisation and Machine Learning," Addison Wesley Longman, Inc. 1989.
- [43] J.H. Wolfe, "A Computer Program for the Computation of Maximum-likelihood Analysis of types," *U.S. Naval Personnel Research Activity, Technical Report*, SRM, pp. 65-112, San Diego, 1965.
- [44] I. Cadez, D. Heckerman, C. Meek, P. Smyth, and S. White, "Model-based Clustering and Visualization of Navigation Patterns on a web site," *Data Mining and Knowledge Discovery*, 7(4), pp.399-424,2003.
- [45] M. Nilashi, O.B. Ibrahim, N. Ithnin, and N.H. Sarmin, "A Multi-criteria Collaborative Filtering Recommender System for the Tourism Domain using Expectation Maximization (EM) and PCA-ANFIS," *Electronic Commerce Research and Applications*, 14(6), pp.542-562, 2015.

- [46] M. Verma, M. Srivastava, N. Chack, A.K. Diswar, and N. Gupta, "A Comparative Study of Various Clustering Algorithms in Data Mining," *International Journal of Engineering Research and Applications (IJERA)*, 2(3), pp.1379-1384, May 2012.
- [47] G. McLachlan and T. Krishnan, "The EM Algorithm and Extensions Vol. 382 of Wiley series in Probability and Statistics," 2007.
- [48] T.K. Moon, "The Expectation-maximization Algorithm," *IEEE Signal Processing Magazine*, 13, no. 6, pp. 47-60, 1996.
- [49] J. MacQueen, "Some Methods for Classification and Analysis of Multivariate Observations." in *Proc. of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, no. 14, pp. 281-297, 1967.
- [50] R. Salman, V. Kecman, Q. Li, R. Strack, and E. Test, "Fast K-means Algorithm Clustering." *arXiv preprint arXiv:1108.1351*, 2011.
- [51] K.A. Nazeer, and M.P. Sebastian, "Improving the Accuracy and Efficiency of the K-means Clustering Algorithm," in *Proc. of the World Congress on Engineering*, Vol. 1, pp.1-3, May 2009.
- [52] Q. Li, and B.M. Kim, "Clustering Approach for Hybrid Recommender System," *In Web Intelligence 2003*, in *Proc. IEEE/WIC International Conference*, pp. 33-38. IEEE, 2003.
- [53] I.H. Witten, and E. Frank, "Data Mining: Practical Machine Learning Tools and Techniques," Morgan Kaufmann. July 2005.

[54] A. Herrero, J. Sedano, B. Baroque, H. Quintián, and E. Corchado, “10th International Conference on Soft Computing Models in Industrial and Environmental Applications” vol. 368, Springer, pp. 122, 2015.

[55] H. Ma, I. King and M.R. Lyu, “Effective Missing Data Prediction for Collaborative Filtering,” in *Proc. of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ACM, pp. 39-46, July 2007.

[56] Python, “Python FAQ” [Online]. Available: <http://python.net/~gherman/Python.html>.

[57] R. Gentleman, W. Huber, and V.J. Carey, “R Language. In International Encyclopedia of Statistical Science,” Springer Berlin Heidelberg, 2011.

[58] O.R. Zaiane, “Building a Recommender Agent for E-learning Systems,” In Computers in Education, in *Proc. of International Conference*, IEEE, 2002. [59] S.Y. Chou, Y.H. Yang, and Y.C. Lin, “Evaluating Music Recommendation in A Real-world Setting: on Data Splitting and Evaluation Metrics,” *IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1-6, IEEE, June 2015.

[60] Y. Zheng, B. Mobasher, and R. Burke, “Carskit: A Java-based Context-aware Recommendation Engine,” *IEEE International Conference on Data Mining Workshop (ICDMW)*, pp. 1668-1671, IEEE, November 2015.

[61] I.Y. Choi, M.G. Oh, J.K. Kim, and Y.U. Ryu, “Collaborative Filtering with Facial Expressions for Online Video Recommendation,” *International Journal of Information Management*, 36(3), pp.397-402, 2016.

[62] D. Wu, G. Zhang, and J. Lu, "A Fuzzy Preference Tree-based Recommender System for Personalized business-to-business E-services," *IEEE Transactions on Fuzzy Systems*, 23(1), pp.29-43, 2015.

[63] G. Linden (2009), "*What is a Good Recommendation Algorithm?*" [Online]. Available: <http://cacm.acm.org/blogs/blog-cacm/22925-what-is-a-good-recommendation-algorithm/fulltext>.

[64] J.L. Herlocker, J.A. Konstan, "An Empirical Analysis of Design Choices in Neighborhood-based Collaborative Filtering Algorithms.," *Information Retrieval*, 5, 287–310, 2002.

## Appendix: Coding Reference

The function below is coded by Python.

### A. Clustering

This function implements the K-means clustering to group items into k clusters.

```
import sys

import random

import copy

def init_arr(r, c):
    for i in range(c):
        tmp_arr = [0] * c
        arr.append(tmp_arr)

    return arr

def kmeans_init(data_arr, clus_k):
    users_num = len(data_arr)
    items_num = len(data_arr[0])
    center = []
    seed = set()
    resd = [-1] * user_num
    for i in range(clus_k):
        idx = int(random.uniform(0, users_num))
        while idx in seed:
            idx = int(random.uniform(0, users_num))
        seed.add(idx)
        print idx
    center.append(copy.deepcopy(data_arr[idx]))
    print len(center)
    print center[0][:20]
```

```

return center, resd

def power(data_list):
res = 0.0
for it in range(len(data_list)):
res += data_list[it] * data_list[it]
return res

def getDistance(element, center):
todo: the i-th calculate
res = 0.0
for i in range(len(elements)):
dif = element[i] - center[i]
res += dif * dif
#res = math.sqrt(res)
return res

def kmean(data_arr, k):
r = len(data_arr)
if r < 1:
print "data error, stop cluster"
return None
c = len(data_arr[0])
try:
center,resd = kmeans_init(data_arr, k)
except:
return None
count = 0
flag = True
while flag and count < 100:

```

```

flag = False
print 'round :' + str(count)
#cal dist
for i in range(0, r):
    dist = sys.maxint
    cent_id = 0
    for j in range(0, k):
        dist_item = getDistance(data_arr[i], center[j])
        if dist_item < dist:
            dist = dist_item
            cent_id = j
    if cent_id != resd[i]:
        resd[i] = cent_id
    flag = True
#cal new center
for j in range(0, k):
    cluster_num = 0
    tmp_center = [0] * c
    for it in range(len(resd)):
        if resd[it] == j:
            cluster_num += 1
    for l in range(c):
        tmp_center[l] += data_arr[it][l]
    if cluster_num == 0:
        print 'xxxx'
        raise Exception
    for l in range(c):

```



```
tmp_center[l] /= float(cluster_num)
center[j] = tmp_center
count += 1
return resd
```

## B. Similarity Calculation

This function stores the pre-calculated user and item similarity and mean in array.

```
import sys
import copy
import random

def estimate(data_arr, resd, aver_data_arr, users_mean, items_mean, pcnt):
    users_num = len(data_arr)
    item_num = len(data_arr[0])
    estm_arr = copy.deepcopy(data_arr)
    item_estm_list = [0] * item_num
    for j in range(item_num):
        user_count = 0.0
        for i in range(user_num):
            if data_arr[i][j] > 0:
                item_estm_list[j] += ave_data_arr[i][j]
                user_count += 1
        if users_count == 0:
            user_count += 1
        item_estm_list[j] /= user_count

    for i in range(user_num):
        for j in range(item_num):
            if estm_arr[i][j] == 0:
                ran = random.random()
                if ran > pcnt
                    continue
            estm_arr[i][j] = user_mean[i] + item_estm_list[j]
```

```

if estim_arr[i][j] < 0:
    estim_arr[i][j] = 1
elif estim_arr[i][j] > 5:
    estim_arr[i][j] = 5
return estim_arr

def estimate_new(data_arr, resd, ave_data_arr, user_mean, item_mean, pcnt):
    user_num = len(data_arr)
    item_num = len(data_arr[0])
    estim_arr = copy.deepcopy(data_arr)
    users_list = []
    for i in range(user_num):
        tmp_set = set()
        for j in range(user_num):
            if i != j and resd[i] == resd[j]:
                tmp_set.add(j)
        user_list.append(tmp_set)
    item_list = []
    for j in range(item_num):
        item_set = set()
        for k in range(item_num):
            if j != k and abs(item_mean[j] - item_mean[k]) < 0.1:
                item_set.add(k)
        item_list.append(item_set)
    for i in range(user_num):
        if estim_arr[i][j] == 0:
            estim_arr[i][j] = user_mean[i]
    user_v = 0

```

```

user_counts = 0.0
for k in user_list[i]:
    if data_arr[k][j] > 0:
        user_v += ave_data_arr[k][j]
    user_count += 1
    if user_count == 0:
        user_count += 1
    user_v /= user_count
    estm_arr[i][j] += 0.5 * user_v
    item_v = 0
    itemsw_count = 0.0
    for m in item_list[j]:
        if data_arr[i][m] > 0:
            item_v += ave_data_arr[i][m]
        item_count += 1
        if item_count == 0:
            item_count += 1
        item_v /= item_count
    estm_arr[i][j] += 0.5 * item_v
    if estm_arr[i][j] < 0:
        estm_arr[i][j] = 1
    elif estm_arr[i][j] > 5:
        estm_arr[i][j] = 5
return estm_arr

```

## C. Evaluation

This file evaluates the recommendation results using MAE, RMSE, Precision, Recall and F1 score.

```
import sys

import time

import math

def rmse(test_arr, res_arr):

    r = len(test_arr)

    c = len(test_arr[0])

    sum_res = 0.0

    count = 0

    test_user = []

    for i in range(r):

        for j in range(c):

            if test_arr[i][j] > 0:

                test_user.append(i)

        for i in test_user:

            for j in range(0, c):

                if test_arr[i][j] > 0:

                    sum_res += (res_arr[i][j] - test_arr[i][j]) ** 2

            count += 1

        if count == 0:

            raise Exception,Exception('No test data', 'in evaluate.py')

    return math.sqrt(sum_res / count)

def mae(test_arr, res_arr):

    r = len(res_arr)

    c = len(res_arr[0])
```

```

sum_res = 0.0
count = 0
test_user = []
for i in range(r):
    for j in range(c):
        if test_arr[i][j] > 0:
            test_user.append(i)
            break
    for i in test_user:
        for j in range(0, c):
            if test_arr[i][j] > 0:
                sum_res += abs(res_arr[i][j] - test_arr[i][j])
                count += 1
    if count == 0:
        raise Exception,Exception('No test data', 'in evaluate.py')
    print count
    return sum_res / count

def cal(res_arr, test_arr, outfile):

    res_rmse = rmse(res_arr, test_arr)
    res_mae = mae(res_arr, test_arr)
    outfile.write("exp result is : \nrmse : " + str(res_rmse) + "\nmae : " + str(res_mae) + "\n")
    return res_rmse, res_mae

def pre_recal(test_arr, rec_items):
    pres = 0.0
    recall = 0.0
    f1 = 0.0

```

```

r = len(test_arr)
c = len(test_arr[0])
tp = 0
fp = 0
test_user = []
for i in range(r):
    for j in range(c):
        if test_arr[i][j] > 0:
            test_user.append(i)
            break

for i in test_user:
    for j in range(len(rec_items[i])):
        if test_arr[i][rec_items[i][j]] > 0:
            tp += 1
        else:
            fp += 1
    for j in range(len(test_arr[i])):
        if test_arr[i][j] > 0 and j not in rec_items[i]:
            tn += 1

print "tp " + str(tp)
print "tn " + str(tn)
print "fp " + str(fp)
pres = float(tp) / float(tp + fp)
recall = float(tp) / float(tp + tn)
if pres + recall == 0:
    f1 = 0

```

else:

$f1 = 2 * pres * recall / (pres + recall)$

return pres, recall, f1



## D. Graph Generation

This file generates the graph of the recommendation results.

```
# -*- coding:utf-8 -*-  
  
import matplotlib.pyplot as plt  
  
import sys  
  
def draw(image_path, xlist, ylist, xlabel, ylabel):  
    colorList = ['b','g','r','c','m','y','k']  
    plt.xlabel(xlabel)  
    plt.ylabel(ylabel)  
    plt.title('recsys')  
    lines = []  
    titles = []  
    line1, = plt.plot(xlist, ylist)  
    plt.setp(line1, color=colorList[0], linewidth=2.0)  
    titles.append(ylabel)  
    lines.append(line1)  
    plt.legend(lines, titles)  
    plt.savefig(image_path, dpi=120)  
    plt.close()
```

## E. Main Function

```
import sys
import math
import time
import estimation
import cluster
import copy
import evaluate
import res_pic

def init_arr(r, c):
    """
    init array with 0
    """
    arr = []
    for i in range(r):
        tmp_arr = [0] * c
        arr.append(tmp_arr)
    return arr

def init_arr_n(r, c):
    """
    init array with -1
    """
    arr = []
    for i in range(r):
        tmp_arr = [-1] * c
```

```

arr.append(tmp_arr)
return arr

def get_input(filename, item_num, user_num):
    """
    get input data
    """
    data_arr = init_arr(user_num, item_num)
    with open(filename) as f:
        for lines in f:
            arr = lines.strip().split('\t')
            u = int(arr[0]) - 1
            i = int(arr[1]) - 1
            r = int(arr[2])
            data_arr[u][i] = r
    return data_arr

def get_ave_data_arr(data_arr, user_mean):
    """
    get average rank ( data - user_mean)
    """
    user_num = len(data_arr)
    item_num = len(data_arr[0])
    ave_data_arr = copy.deepcopy(data_arr)
    for i in range(user_num):
        for j in range(item_num):
            ave_data_arr[i][j] -= user_mean[i]
    return ave_data_arr

```

```

def get_pear_data_arr(data_arr, item_mean):
    """
    get average rank array ( data - item_mean): used for pearson similarity calculation
    """
    user_num = len(data_arr)
    item_num = len(data_arr[0])
    pear_data_arr = copy.deepcopy(data_arr)
    for i in range(user_num):
        for j in range(item_num):
            pear_data_arr[i][j] -= item_mean[j]
    return pear_data_arr

def get_sqr_data_arr(ave_data_arr):
    """
    get square of data
    """
    user_num = len(ave_data_arr)
    item_num = len(ave_data_arr[0])
    sqr_data_arr = copy.deepcopy(ave_data_arr)
    sqr_data_list = [0] * item_num
    for i in range(user_num):
        for j in range(item_num):
            sqr_data_arr[i][j] = ave_data_arr[i][j] * ave_data_arr[i][j]
    for l in range(item_num):
        sq_sum = 0
        for m in range(user_num):

```

```

sq_sum += sqr_data_arr[m][l]
sqr_data_list[l] = math.sqrt(sq_sum)
if sqr_data_list[l] == 0:
    sqr_data_list[l] = 1
return sqr_data_list

def get_user_mean(data_arr):
    """
    get user mean
    """
    user_num = len(data_arr)
    item_num = len(data_arr[0])
    user_mean = [0] * user_num
    for i in range(user_num):
        user_ave = 0.0
        user_count = 0.0
        for j in range(item_num):
            if data_arr[i][j] > 0:
                user_ave += data_arr[i][j]
                user_count += 1
        if user_count > 0:
            #user_mean[i] = user_ave / user_count
            user_mean[i] = user_ave / item_num
        else:
            user_mean[i] = 0
    return user_mean

def get_item_mean(data_arr):
    """

```

```

get item mean
"""
user_num = len(data_arr)
item_num = len(data_arr[0])
item_mean = [0] * item_num
for j in range(item_num):
    user_ave = 0.0
    user_count = 0.0
    for i in range(user_num):
        if data_arr[i][j] > 0:
            user_ave += data_arr[i][j]
            user_count += 1
    if user_count > 0:
        #item_mean[i] = user_ave / user_count
        # item_mean[] = user_ave / user_num
    else:
        item_mean[j] = 0
return item_mean

def cosin_similarity(ave_data_arr, sqr_data_list, data_arr):
    """
    calculate cosin similarity
    """
    user_num = len(ave_data_arr)
    item_num = len(ave_data_arr[0])
    sim_arr = init_arr(item_num, item_num)
    for i in range(item_num):

```

```

for j in range(0, i):
    sim_arr[i][j] = sim_arr[j][i]
    sim_arr[i][i] = 1
for j in range(i + 1, item_num):
    ms = 0.0
    for l in range(user_num):
        # sim_arr[i][j] += ave_data_arr[l][i] * ave_data_arr[l][j]
        # sim_arr[i][j] /= (sqr_data_list[i] * sqr_data_list[j])
    return sim_arr

def sim_sort(sim_arr):
    """
    sorted by similarity for each item
    """
    item_num = len(sim_arr)
    # sim_sorted = []
    for i in range(item_num):
        temp = {j:sim_arr[i][j] for j in range(0, len(sim_arr[i]))}
        sim_s = sorted(temp.items(), key = lambda k:k[1], reverse=True)
        tmp_list = [sim_s[j][0] for j in range(0, len(sim_arr))]
        sim_sorted.append(tmp_list)
    return sim_sorted

def rec(data_arr, sim_sorted, sim_arr, top_rec):
    """
    calculate the recomand result
    """

```

```

res_arr = copy.deepcopy(data_arr)
user_num = len(data_arr)
item_num = len(data_arr[0])
for i in range(user_num):
    for j in range(item_num):
        if res_arr[i][j] == 0:
            user_count = 0
            sim_sum = 0.0
            for k in sim_sorted[j]:
                if k == j:
                    continue
                if user_count >= top_rec:
                    break
                if res_arr[][k] > 0:
                    user_count += 1
                res_arr[i][j] += sim_arr[j][k] * data_arr[i][k]
                sim_sum += abs(sim_arr[j][k])
            if sim_sum == 0:
                sim_sum = 1
            res_arr[i][j] /= sim_sum
            if res_arr[i][j] < 1:
                #print " " + str(res_arr[i][j])
                res_arr[i][j] = 1
            elif res_arr[i][j] > 5:
                #print " " + str(res_arr[i][j])
                res_arr[i] = 5
    return res_arr

```



```

def get_rec_item(data_arr, res_arr, k_rec):
    """
    get the k_rec items for recommendation
    """
    user_num = len(data_arr)
    item_num = len(data [0])
    rec_items = init_arr_n(user_num, k_rec)
    for i in range(user_num):
        tmp_list = []
        for j in range(item_num):
            if data_arr[i][j] == 0 and res_arr[i][j] > 0:
                tmp_list.append([j, res_arr[i][j]])
        tmp_sorted_list = sorted(tmp_list, key = lambda k:k[1], reverse=True)
        tmp_sorted_list = tmp_sorted_list[:k_rec]
        for m in range(len(tmp_sorted_list)):
            rec_items[i][m] = tmp_sorted_list[m][0]
    return rec_items

def main():
    """
    single round calculation
    """
    in_file = sys.argv[1]
    test_file = sys.argv[2]
    #out_file = sys.argv[3]
    #user_num = 5
    #item_num = 8

```

```

user_num = 943
item_num = 1682

data_arr = get_input(in_file, item_num, user_num)
test_arr = get_input(test_file, item_num, user_num)
user_mean = get_user_mean(data_arr)
item_mean = get_item_mean(data_arr)
cluster_num = 1
k_neighbour = 40
#k_neighbour = int(sys.argv[3])
k_rec = 10
#pcnt = float(sys.argv[3])
pcnt = 1

cluster_res = cluster.kmeans(data_arr, cluster_num)
ave_data_arr = get_ave_data_arr(data_arr, user_mean)
estm_arr = estimation.estimate_new(data_arr, cluster_res, ave_data_arr, user_mean,
item_mean, pcnt)

ave_data_arr = get_ave_data_arr(data_arr, user_mean)
sqr_data_list = get_sqr_data_arr(ave_data_arr)

ave_data_arr = get_ave_data_arr(estm_arr, user_mean)
sqr_data_list = get_sqr_data_arr(ave_data_arr)

#pear_data_arr = get_pear_data_arr(estm_arr, item_mean)
#pear_sqr_data = get_sqr_data_arr(pear_data_arr)

```

```

sim_arr = cosin_similarity(ave_data_arr, sqr_data_list, estm_arr)
#sim_arr = cosin_similarity(pear_data_arr, pear_sqr_data, estm_arr)
#sim_arr = cosin_similarity(pear_data_arr, pear_sqr_data, data_arr)
#sim_arr = cosin_similarity(ave_data_arr, sqr_data_list, data_arr)
sim_sorted = sim_sort(sim_arr)
#res_arr = rec(data_arr, sim_sorted, sim_arr, k_neighbour)
res_arr = rec(estm_arr, sim_sorted, sim_arr, k_neighbour)
rec_items = get_rec_item(data_arr, res_arr, k_rec)
rmse = evaluate.rmse(test_arr, res_arr)
print 'rmse: %.5f' % rmse
mae = evaluate.mae(test_arr, res_arr)
print 'mae: %.5f' % mae
pres, recall, f1 = evaluate.pre_recall(test_arr, rec_items)
print 'precise: %.5f' % pres
print 'recall: %.5f' % recall
print 'f1: %.5f' % f1
def run(cluster_num, k_neighbour, k_rec, pcnt, data_arr, test_arr, user_mean, item_mean):
    """
    run the process
    """
    ave_data_arr = get_ave_data_arr(data_arr, user_mean)
    estm_arr = estimation.estimate_new(data_arr, cluster_res, ave_data_arr, user_mean,
item_mean, pcnt)
    ave_data_arr = get_ave_data_arr(data_arr, user_mean)
    sqr_data_list = get_sqr_data_arr(ave_data_arr)
    ave_data_arr = get_ave_data_arr(estm_arr, user_mean)
    sqr_data_list = get_sqr_data_arr(ave_data_arr)
    #pear_data_arr = get_pear_data_arr(estm_arr, item_mean)

```

```

#pear_sqr_data = get_sqr_data_arr(pear_data_arr)
sim_arr = cosin_similarity(ave_data_arr, sqr_data_list, estim_arr)
#sim_arr = cosin_similarity(pear_data_arr, pear_sqr_data, estim_arr)
#sim_arr = cosin_similarity(pear_data_arr, pear_sqr_data, data_arr)
#sim_arr = cosin_similarity(ave_data_arr, sqr_data_list, data_arr)
sim_sorted = sim_sort(sim_arr)
#res_arr = rec(data_arr, sim_sorted, sim_arr, k_neighbour)
res_arr = rec(estim_arr, sim_sorted, sim_arr, k_neighbour)
rec_items = get_rec_item(data_arr, res_arr, k_rec)
#rmse = evaluate.rmse(test_arr, res_arr)
#mae = evaluate.mae(test_arr, res_arr)
#pres, recall, f1 = evaluate.pre_recal(test_arr, rec_items)

rmse = evaluate.rmse(test_arr, res_arr)
print 'rmse: %.5f' % rmse
mae = evaluate.mae(test_arr, res_arr)
print 'mae: %.5f' % mae
pres, recall, f1 = evaluate.pre_recal(test_arr, rec_items)
print 'precise: %.5f' % pres
print 'recall: %.5f' % recall
print 'f1: %.5f' % f1

return rmse, mae, pres, recall, f1
def test():
in_file = sys.argv[1]
u = 6
i = 8

```

```

data_arr = get_input(in_file, i, u)
test_arr = get_input(test_infile, i, u)
#data_arr = init_arr(3, 4)
#data_arr = init_arr_n(3, 4)
print 'data:'
print data_arr
user_mean = get_user_mean(data_arr)
print 'user_mean:'
print user_mean
item_mean = get_item_mean(data_arr)
print 'item_mean:'
print item_mean
ave_data_arr = get_ave_data_arr(data_arr, user_mean)
print 'ave_data_arr:'
print ave_data_arr
pear_data_arr = get_pear_data_arr(data_arr, item_mean)
print 'pear_ave_data_arr:'
print pear_data_arr
sqr_data_arr = get_sqr_data_arr(ave_data_arr)
print 'sqr_data_arr:'
print sqr_data_arr
sim_arr = cosin_similarity(ave_data_arr, sqr_data_arr, data_arr)
print 'sim_arr:'
print sim_arr

sim_arr_sort = sim_sort(sim_arr)
print 'sim_sorted:'

```

```

print sim_arr_sorted

res = rec(data_arr, sim_arr_sorted, sim_arr, 2)
print 'res arr:'
print res
print test_arr
rmse = evaluate.rmse(test_arr, res)
print 'rmse: %.5f' % rmse
mae = evaluate.mae(test_arr, res)
print 'mae: %.5f' % mae
#pres, recall, f1 = evaluate.pre_recal(test_arr, rec_items)
pic_file = './ad'
axis_list = [1,2, 3,4,5,6,7,8]
res_pic.draw(pic_file + '.rmse.png', axis_list, item_mean, 'pcent' , 'rmse')
print axis_list
res_pic.draw(pic_file + '.f1.png', axis_list, test_arr[0], 'pcent' , 'f1')
print axis_list
#print user_mean
#print item_mean
data_arr[2][3] = 1
#print data_arr
def process():
in_file = sys.argv[1]
test_infile = sys.argv[2]
pic_file = sys.argv[3]
user_num = 943
item_num = 1682

```

```

data_arr = get_input(in_file, item_num, user_num)
test_arr = get_input(test_infile, item_num, user_num)
user_mean = get_user_mean(data_arr)
item_mean = get_item_mean(data_arr)
cluster_num = 3
k_neighbour = 40
k_rec = 50
pcnt = 0.06
rmse_list = []
mae_list = []
pres_list = []
recall_list = []
f1_list = []
axis_list = []
for i in range(10):
    k_rec = i * 100 + 100
    rmse, mae, pres, recall, f1 = run(cluster_num, k_neighbour, k_rec, pcnt, data_arr, test_arr,
user_mean, item_mean)
    rmse_list.append(rmse)
    mae_list.append(mae)
    pres_list.append(pres)
    recall_list.append(recall)
    f1_list.append(f1)
    res_pic.draws(pic_file + '.rmse.png', axis_list, rmse_list, 'List' , 'RMSE')
    res_pic.draw(pic_file + '.mae.png', axis_list, mae_list, 'List' , 'MAE')
    res_pic.draw(pic_file + '.pres.png', axis_list, pres_list, 'List' , 'pres')
    res_pic.draw(pic_file + '.recall.png', axis_list, recall_list, 'List' , 'recall')
    res_pic.draw(pic_file + '.f1.png', axis_list, f1_list, 'List' , 'f1')

```

```
if __name__ == '__main__':  
    main()  
    #test()  
    #process()
```



## F. Non-clustering Method

Non-clustering method used for comparison.

```
import sys
import time
import cofilter
import datasetInput
import evaluate
import ConfigParser
from numpy import *

def getConfig(ConfigPath):
    config = ConfigParser.ConfigParser()
    config.read(ConfigPath)
    return

def getDictConf(ExpConfig, section):
    conf_dict = dict()
    try:
        para = ExpConfig.items(section)
    except Exception as e:
        print 'Error conf item: ' + section
    return conf_dict
    for item in para:
        conf_dict[item[0]] = item[1]
    return conf_dict

def main():
    ExpConfig = getConfig('./recommand.conf')
```

```

user_num = int(ExpConfig.get('Dataset', 'user_num'))
item_num = int(ExpConfig.get('Dataset', 'item_num'))
para = getDictConf(ExpConfig, 'Parameters')
res_file = ExpConfig.get('Files', 'output')
data_file = ExpConfig.get('Files', 'train')
test_file = ExpConfig.get('Files', 'test')
# pic_file = ExpConfig.get('Files', 'pic')

if 'clus_k' not in para:
    print 'Error Parameters Conf: lost cluster k'
    return -1
clus_k = int(para['clus_k'])
if 'top_k' not in para:
    print 'Error Parameters Conf: lost top k'
    return -1
top_k = int(para['top_k'])
if 'rec_l' not in para:
    print 'Error Parameters Conf: lost rec l'
    return -1
rec_l = int(para['rec_l'])
data_arr = zeros((user_num, item_num))
res_arr = zeros((user_num, item_num))
test_arr = zeros((user_num, item_num))
datasetInput.load(data_file, data_arr)
datasetInput.load(test_file, test_arr)
pic_axis = list()
rmse_list = list()

```

```

mae_list = list()

outfile_name = res_file + time.strftime("%Y%m%d-%H%M%S",
time.localtime(time.time()))

outfile = open(outfile_name, 'w')

for ti in range(top_k, 25, 10):
outfile.write("exp condition is : \n")
para['top_k'] = ti

for item in para:
outfile.write(item + " : " + str(para[item]) + "\n")
res_arr, simi_arr = cofilter.run(para, data_arr, res_arr)
savetxt(outfile_name + '.res_arr_' + str(ti), res_arr)
savetxt(outfile_name + '.simi_arr_' + str(ti), simi_arr)
res_rmse, res_mae = evaluate.cal(res_arr, test_arr, outfile)
pic_axis.append(ti)
mae_list.append(res_mae)
rmse_list.append(res_rmse)

#res_pic.draw(pic_file + '.mae.png', pic_axis, mae_list, 'clus_k', 'mae')
#res_pic.draw(pic_file + '.rmse.png', pic_axis, rmse_list, 'clus_k', 'rmse')

outfile.close()

if __name__ == '__main__':
main()

```