# RESOLVING BIAS IN HYBRID RECOMMENDATION SYSTEM

**A PROJECT REPORT**

***Submitted by***

| | |
|---|---|
| **DANTHI S AKSHAY** | **(412413104017)** |
| **RAGHUL S** | **(412413104065)** |
| **KARTHIK M** | **(412413104036)** |

***In partial fulfillment for the award of the degree***

***of***

**BACHELOR OF ENGINEERING**

**IN**

COMPUTER SCIENCE AND ENGINEERING

**SRI SAIRAM INSTITUTE OF TECHNOLOGY, WEST TAMBARAM**

**ANNA UNIVERSITY: CHENNAI 600 025**

**APRIL 2017**

**ANNA UNIVERSITY: CHENNAI 600 025**

# BONAFIDE CERTIFICATE

Certified that this project report **"RESOLVING BIAS IN HYBRID RECOMMENDATION SYSTEM"** is the bonafide work of **"DANTHI .S. AKSHAY (412413104017), S.RAGHUL (412413104065), M.KARTHIK (412413104036)"** who carried out the project work under my supervision.

| | |
|---|---|
| **SIGNATURE** | **SIGNATURE** |
| **Ms. B. SREEDEVI M. Tech., (Ph. D)**<br>**HEAD OF THE DEPARTMENT** | **Ms. S. ANANTHI M.E**<br>**SUPERVISOR** |
| Department of Computer Science<br>and Engineering<br>Sri Sai Ram Institute of Technology<br>West Tambaram,<br>Chennai-600044. | Department of Computer Science<br>and Engineering<br>Sri Sai Ram Institute of Technology<br>West Tambaram,<br>Chennai-600044. |

Submitted for University Project Examination held on …………………………. at Sri Sai Ram Institute of Technology, Chennai-600044.

**INTERNAL EXAMINER** **EXTERNAL EXAMINER**

# ACKNOWLEDGEMENT

We express our deep sense of gratitude to our beloved Chairman **Thiru. MJF. Ln. LEO MUTHU** for the help and advice he has shared upon us.

We express our gratitude to our CEO **Mr. J. SAI PRAKASH LEO MUTHU** and our Trustee **Mrs. J. SHARMILA RAJAA** for their constant encouragement in completing the project.

We express our solemn thanks to our esteemed Principal **Dr. K. PALANIKUMAR** for having given us spontaneous and wholehearted encouragement for completing this project.

We are indebted to our HOD **Ms. B. SREEDEVI** for her support during the entire course of this project work.

We express our gratitude and sincere thanks to our guide **Ms. S. ANANTHI M.E, Asst. Professor** for her valuable suggestions and constant encouragement for successful completion of this project.

Our sincere thanks to our project coordinator **DR. K. C. SURESH M.E, Ph. D.,** for his kind support in bring out this project successfully.

Finally, we thank all the teaching and Non-teaching staff members of the Department of Computer Science and Engineering and all others who contributed directly or indirectly for the successful completion of our project.

## TABLE OF CONTENTS

# ABSTRACT

Recommender systems or recommendation systems are a subclass of information filtering system that seek to predict the "rating" or "preference" that a user would give to an item. Based on your ratings, it generates personalized predictions for items you haven't seen yet. In the past 2 years, we have created more data than ever. But this abundance of choice causes a problem. There is a huge load of incoming data which causes Bias which gives rise to Simpson’s paradox.

Our recommendation system incorporates an algorithm that removes bias from the recommendation algorithms. One of the largest biases in this kind of data is due to the phenomenon of Simpson’s paradox. “Simpson’s Warning: less conditioning is most likely to lead to serious bias when Simpson’s Paradox appears.”

This data without proper pooling can lead to incorrect recommendations and even ignored items that are potential recommendations. Hence, this could be solved by a technique called multi-resolution which is a method of pooling this incorrect data accurately.

# CHAPTER 1

# INTRODUCTION

## 1.1 RECOMMENDATION SYSTEMS

The recommender systems are software applications, subclass of information filtering systems that analyzes available data to make suggestions that are interesting to users, such as books, movies or songs, among other possibilities. Recommendation systems seek to predict the “ratings” or “preferences” that a user would give to an item.

Recommender systems were in existence from the 1990s for the users in selecting the most suitable items from the abundant of choices available to them. The idea that led to the development of recommender systems was that, we people often depend on the suggestions of our peers for trying something new, say for before buying a smart phones, a laptops, before going for a movie, before going to a new restaurant and even before visiting a doctor. We have numerous recommendation systems developed for various fields, using different recommendation approaches. Yet, there are still a few limitations of recommender systems that need to be worked on.

Over several years, researchers have developed many recommendation engines for almost many of the domain like social-networking sites, entertainment, content based sites (e-learning, books or articles recommendation, e-filtering etc.), e-commerce, tourism, match-making and a lot more, all dealing with real-world. We can classify recommender systems on the basis of their application areas: Entertainment, Social-networking based, Content-based, Services-based and E-commerce.

Basically recommendation systems use several approaches for the process of providing recommendations for the users. There are several approaches that recommendation systems use which includes Content Based Filtering (CBF), Collaborative Filtering (CF), and Hybrid systems which are the most frequently used approaches. However there are several more approaches like Demographic systems, Community Based systems, Knowledge Based systems.

However we concentrate on Content Based, Collaborative, and Hybrid approaches for resolving limitation of Recommendation systems.

## 1.2 RECOMMENDAITON APPROACHES

The recommendation approaches can be classified and generalized as Content Based, Collaborative, and Hybrid approaches.

### 1.2.1 COLLABORATIVE FILTERING

Collaborative filtering is the process of making automated predictions and suggestions about user’s interest by collecting preferences from all or many of the users. A collaborative system filters information by recommendations of other users. The basic idea behind the collaborative filtering is that “*Users who agreed in their evaluation of certain items in the past are likely to agree again in future*”.

Collaborative filtering is referred to as people-to-people correlation. Basic concept of collaborative approach is that two or more individuals sharing some similar interests in one area tend to get inclined towards similar items or products from some other area too. The similarity between the users can be figured out on their browsing characteristics (click-through rate), browsing pattern and ratings (explicit, implicit). The concept of collaborative filtering can be easily understood with the help of a simple example: consider Face Book. You can always see "people you may know" option on your home page with multiple people in the list. So the basic criteria behind making those suggestions are based on the concept of recommender systems only. The suggestions are filtered out on multiple parameters like: number of mutual friends you have with that individual, number of similar pages you both have liked or groups in common and also common places you have been to or you belong to. The approach used here is Collaborative filtering i.e. if a person x and you have a number of friends in common than the chances are that you two may also know each other. Hence it is called people to people co-relation.

Collaborative filtering can be classified as Item-Based Collaborative Filtering and User-Based Collaborative Filtering. In User-Based approach, for active user if ratings and user id is available, a group of similar users with same interest in terms of past ratings is found out and their ratings are used to predict what might interest the active user. In Item-Based approach it considers the similarity between the rating patterns of the items. For two items having similar users who like and dislike them, they are considered similar and the users thus are anticipated to have similar interests for similar items. It is similar to content based filtering in terms of overall structure, except the fact that here the similarity between the items is found based on the user ratings pattern and not on the item.

### 1.2.2 CONTENT BASED FILTERING

Content based filtering systems are used for providing suggestions by building a user profile to indicate items that the particular user may like. Content based systems recommend items similar to items that user liked in past. Content based systems are for providing personalized suggestions for particular users. Content based systems are based on the concept “Show me more of what I have liked in past”.

The basic idea behind these systems is to recommend items or products to a particular user, which are similar to the ones that user has already liked in the past. The similarity between two or more items can be calculated based on their similar features. To understand better consider the example discussed above, whenever you watch any video on Face book like the ones posted on food lover’s page, after you finish watching it, you get links of similar videos on your home page. Also, when you like some page, you get suggestions of pages similar to them. So what actually is behind all this is Content based filtering i.e. if you like an ABC item from some particular category, it is likely that you might like any other item XYZ similar to it (from the same category or some category similar to it).

There certain limitations in both of the above approaches that can be solved to an extent by combining the features of both the systems. So we go in for Hybrid Recommendation System.

### 1.2.3 HYBRID APPROACHES

A Hybrid system is one which combines the features of both Content based and Collaborative system. The Hybrid systems combine multiple recommendation techniques together to produce its output. Several studies and experiments show that performance evaluation of hybrid recommendations to be better than the collaborative and content based systems.

A new system without collaborative data, the collaborative filtering approach cannot generate predictions. A system without content information, the content-based approach will fail to recommend. To overcome these principal limitations, hybrid systems have been developed.

There are seven Hybridization methods used for recommendation filtering by Hybrid recommender systems used in present. They are weighted, Switching, Mixed, Feature combination, Cascade, Feature Augmentation, Meta-level.

The technical description of seven methods is as follows,

**Weighted:**
Outputs of several techniques are combined with different degrees of importance to offer final recommendations.

**Switching:**
Depending on the situation the system changes from one technique to another.

**Mixed:**
Recommendations from several techniques are presented at same time.

**Feature combination:**
Features from different recommendation sources are combined as input to a single technique.

**Cascade:**
The output from one technique is used as input for another technique the refines the result.

**Feature augmentation:**
The output from one technique is used as input features to another.

**Meta-level:**
The model learned by one recommender is used as input to another.

The several of the hybrid systems are built to give best of above techniques by combining them in several ways. For example consider Netflix, it is a hybrid of collaborative filtering and content-based filtering i.e. it recommends movies to the user on the basis of both his likes and his similarity with other users. Suppose a user likes The Notebook, Fated to Love you, PS I Love you etc. then the next time he visits the site, he would be recommended movies from Romantic genre (content-based). Also, if a user x and a user y has plenty of movies they have liked in common, then each of them would be recommended the next movie either of them likes (collaborative-filtering).

## 1.3 INFORMATION OVERLOAD AND CHOICE PARADOX

The recommender systems are used to reduce two negative effects associated with handling of large datasets. The two negative effects associated with recommendation engines are information overload and choice paradox.

According to choice paradox "*if individuals are given more options for a choice they become less satisfied with their decision*".

According to information overload "*as more and more, rapidly changing information is available the users become overwhelmed by the new information and becomes less dependent on new information*".

Choosing a particular item from an item set containing many attractive items may be a difficult task for the users even though the users are supported by the recommendation engines. The larger dataset containing only good items do not necessarily result in higher choice satisfaction compared to smaller subset. The reason is increasing recommendations is counteracted by the increasing complexity of choosing from those recommendations.

Let us consider choice overload problem in the Netflix hybrid recommender systems. Consider how Netflix presents its list of recommendations: essentially a huge matrix of movies from which we need to choose just one to view, which then shows related movies, hardly narrowing our range of choices. At last paralyzed by the unending list of Netflix recommendations, finally having to settle on a film that we hope is good enough but about which we're still not quite sure.

Within the context of their user interfaces, recommender systems actually contribute to information overload and the paradox of choice rather than reduce those negative effects, which they were originally designed to do.

## 1.4 UNDERSTANDING THE BIAS

The Bias in recommender system is an unfair relativity between the items where the items that are actually equal in sense, but rated or preferred unequally. The users may report a particular item to be good since they like it and also suggest it to their circles, thus item may get overrated. The self-reported preference rating for a specific consumed item that is submitted by a user to a recommender system may be affected (i.e., distorted) by the previously observed system's recommendation.

Biases reflected in user ratings not only provide a distorted view of user preferences but also contaminate inputs of recommender systems, leading to decreased quality of future recommendations. Much variation of ratings is due to effects associated with either items or users, independently of their interactions

- *Some users tend to give higher ratings than other users*
- *Some items tend to receive higher ratings than other items*

Dealing with large datasets leads to serious bias as a result of *Simpson's paradox* in recommender systems. This Simpson effect states that a trend appears in different groups of data but disappears when the groups are combined.

An example for bias as result of Simpson effect can be illustrated in Pandora music recommendation engine, for a user searching songs from past till date may be preferred with many songs of present and very less songs from past. The reason is new songs comprise a large data set compared to the old songs, giving less room for old songs being recommended as whole which is serious bias that is not resolved by the recommender systems.

## 1.5 AIM OF PROJECT

The aim of the project is to acknowledge the effect of bias in the recommendation system, which is less considered as factor that degrades the performance of recommender systems. The efficiency of the recommendation systems can be improved to an extent if the problem of bias is considered and is resolved by the multi-resolution techniques. The choice paradox, information overload, and bias resolved recommender systems can provide the user with increased efficiency of preferences and choices. The proposed system uses multi-resolution techniques, to classify the preferences and ratings.

# CHAPTER 2

# LITERATURE SURVEY

## 2.1 LITERATURE REVIEW

J Bobadilla, F Ortega, A Hernando, 2013 proposed “Recommender Systems Survey” explains how recommender systems have developed in parallel with web. It states that recommendation systems were initially based on demographic, content-based and collaborative filtering. Currently, these systems are incorporating social information. In the future, they will use implicit, local and personal information from the Internet of things. This article provides an overview of recommender systems as well as collaborative filtering methods and algorithms; it also explains their evolution, provides an original classification for these systems, identifies areas of future implementation and develops certain areas selected for past, present or future importance.

Dirk Bollen, Bart P. Knijnenburg, Martijn C. Willemsen, Mark Graus, 2010 proposed “Understanding Choice Overload in Recommender Systems” that acknowledges the choice paradox in recommendation systems. Even though people are attracted by large, high quality recommendation sets, psychological research on choice overload shows that choosing an item from recommendation sets containing many attractive items can be a very difficult task. A web-based user experiment using a matrix factorization algorithm applied to the MovieLens dataset was used to investigate the effect of recommendation set size (5 or 20 items) and set quality (low or high) on perceived variety, recommendation set attractiveness, choice difficulty and satisfaction with the chosen item. The results show that larger sets containing only good items do not necessarily result in higher choice satisfaction compared to smaller sets, as the increased recommendation set attractiveness is counteracted by the increased difficulty of choosing from these sets. These findings were supported by behavioral measurements revealing intensified information search and increased acquisition times for these large attractive sets. Important implications of these findings for the design of recommender system user interfaces will be discussed. Iyengar and Lepper were the first to demonstrate the choice overload effect, which refers to the fact that people are attracted by large choice sets, but such large sets at the same time increase choice difficulty and reduce choice satisfaction.

Gediminas Adomavicius, Jesse Bockstedt, Shawn Curley, and Jingjing Zhang , 2014 proposed “De-Biasing User Preference Ratings in Recommender Systems”.
This paper focuses on the problem of “de-biasing” users’ submitted preference ratings and proposes two possible approaches to remove anchoring biases from self-reported ratings. Based on these previous studies, users’ preference ratings can be significantly distorted by the system predicted ratings that are displayed to users. Such distorted preference ratings are subsequently submitted as users’ feedback to recommender systems, which can potentially lead to a biased view of consumer preferences and several potential problems : (i) biases can contaminate the recommender system’s inputs, weakening the system’s ability to provide high-quality recommendations in subsequent iterations; (ii) biases can artificially pull consumers’ preferences towards displayed system recommendations, providing a distorted view of the system’s performance; (iii) biases can lead to a distorted view of items from the users’ perspectives. Thus, when using recommender systems, anchoring biases can be harmful to system’s use and value, and the removal of anchoring biases from consumer ratings constitutes an important and highly practical research problem.

Sanjay Krishnan, Jay Patel, Michael J. Franklin, Ken Goldberg, 2014 proposed "Social Influence Bias in Recommender Systems" which deals with the bias in recommender system by users. To facilitate browsing and selection, almost all recommender systems display an aggregate statistic (the average/mean or median rating value) for each item. This value has potential to influence a participant's individual rating for an item due to what is known in the survey and psychology literature as Social Influence Bias; the tendency for individuals to conform to what they perceive as the "norm" in a community. As a result, ratings can be closer to the average and less diverse than they would be otherwise. This  article suggest that social influence bias can be significant in recommender systems and that this bias can be substantially reduced with machine learning. To apply this methodology to other recommender systems, a key question for future work is how is how to extend the approach to large item inventories and how much training data is required in such cases. One idea is to cluster/classify items into a small number of representative categories and train a model for each category. We believe that selecting an optimal set of items for training in this context may be posed as a submodular maximization problem. This article looks in applying this methodology to recommender systems in other domains with alternative regression methods. This also interested in performing more user studies where a false median is presented (as in the Asch experiments) and exploring methods to optimally classify participants as conformers and nonconformists. It is important to study and quantify the role of social influence on textual data.

Asela Gunawardana, Christopher Meek , 2009 proposed "A Unified Approach to Building Hybrid Recommender Systems " to discuss the ways of combining content based and collaborative filtering techniques. This articles discusses the Content-based recommendation systems can provide recommendations for “cold start” items for which little or no training data is available, but typically have lower accuracy than collaborative filtering systems. Conversely, collaborative filtering techniques often provide accurate recommendations, but fail on cold start items. Hybrid schemes attempt to combine these different kinds of information to yield better recommendations across the board. It describes unified Boltzmann machines, which are probabilistic models that combine collaborative and content in- formation in a coherent manner. They encode collaborative and content information as features, and then learn weights that reflect how well each feature predicts user actions. In doing so, information of different types is automatically weighted, without the need for careful engineering of features or for post-hoc hybridization of distinct recommender systems. This presents empirical results in the movie and shopping domains showing that unified Boltzmann machines can be used to combine content and collaborative information to yield results that are competitive with collaborative technique in recommending items that have been seen before, and also effective at recommending cold-start items.

# CHAPTER 3

# SYSTEM ANALYSIS

## 3.1 LIMITATIONS OF EXISTING SYSTEMS

The hybrid recommender systems in use are subjected to certain limitations that result in the degradation of efficiency and performance. The challenges and problems in the recommender systems are explained below.

Cold start:

The problem occurs at early state of recommender systems when the information about an item or product available is less. The content based systems will behave poorly if product information available is less. On other hand if there are no people behavior history then the collaborative systems do not work efficiently.

Sparsity:

It is common in e-commerce and other domains that people usually purchase or rate relatively few items compared with the total number of items. That leads to a sparse users-items representation matrix and, therefore, inability to locate neighbors or derive common behavior patterns, and, the final result is low-quality recommendations.

Synonyms:

If single item is represented by two or more different words the recommendation systems consider that item as two or more different items and recommend to the user. This increases the number of recommendations which causes the same product to be recommended more than once.

Scalability:

It is another important concern in Recommender sytems. As ratings database grows, the performance decreases. It is beneficial to try to make systems, which can handle large amounts of data and produce accurate recommendations quickly.

Bias:

In recommended systems the two items may be of equal ratings but one may be overrated and the other may be underrated by recommender systems due to the user ratings based on individual likings.

Choice overload:

As the users are given with more choices they become less satisfied by the decision they make in selecting a product or item. The choice overload in recommender system affects users.

Lack of serendipity:

Serendipity refers to providing the user with item that the user has not seen but will be delighted if recommended. But recommender systems tend to show what user already has viewed.

## 3.2 SCOPE OF PROJECT

The scope of our project is to overcome certain issues that occur in the recommendation systems. By use of the multi-resolution techniques in the recommendation systems the performance of recommender systems can be enhanced. The better performance of the hybrid recommender systems can be achieved and the performance can evaluated.

## 3.3 PROPOSED SYSTEM

In the proposed system the modifications are done to existing hybrid recommendation system by implementing changes in algorithm that resolves the problem of bias and choice overload in the existing hybrid systems. For the same data set the recommendations suggested by the proposed system tends to show better recommendations. The existing system is remodeled into a hybrid recommender system with bias factor being resolved. Proposed system incorporates multi-resolution technique that resolves the bias to considerable extent when deployed to the hybrid recommender systems.

## 3.4 SYSTEM MODULES

The system modules are listed and are classified as given below,

- Hybrid recommender system
- Bias resolved hybrid system
- Performance evaluation metrics

### 3.4.1 HYBRID RECOMMENDER SYSTEM

The hybrid recommender system is the combined form of system with functionality of both content based and collaborative systems. The system used in the project incorporates the hybridization techniques for the process of preferring and suggesting the user with recommendations. The system accepts the input from user as search query and results for the query are the set of recommendations that the user needs. The system provides preferences to user but they are subjected to bias which will not be resolved by this system. This system incorporates both the content based and collaborative techniques but has some efficiency issues.

### 3.4.2 BIAS RESOLVED HYBRID SYSTEM

The bias resolved hybrid recommender system is the modification of the above recommender system. The both systems implement the content based and collaborative filtering with hybridization techniques. But the previous system does not account for resolving the bias that occur in the suggestions and rating. This problem of bias and choices are overcome by this bias resolved hybrid systems. These systems have been incorporated with multi-resolution techniques in the hybrid recommendation algorithms that overcome the problems of bias in system.

### 3.4.3 PERFORMANCE EVALUATON METRICS

The performance metrics are used to evaluate the performance of the recommendation systems. Here we measure the performance of the both system, the hybrid recommender system and the bias resolved hybrid system. The performance of the systems can measured and represented as visualize function between two systems. Several of the performance evaluation metrics for visualization of the performance we can use Mean Average Precisions or Normalized Discounted Cumulative Gain.

# CHAPTER 4

# DEVELOPMENT ENVIRONMENT

## 4.1 HARDWARE ENVIRONMENT

The hardware requirements may serve as the basis for the process of implementation of the system and should therefore be a complete and consistent specification of the whole system. They are used by software engineers as the starting point for the system design.  It shows what the systems do and how the system should be implemented.

| | |
|---|---|
| • Hard disk | : 100 GB |
| • Monitor | : 15’ colour with vgi card suport |
| • RAM | : Minimum 2.00 GB |
| • Processor | : Pentium iv and above (or) |
| • Processor Speed | :Minimum 500 MHZ |

## 4.2 SOFTWARE ENVIRONMENT

The software requirements are the specifications of the system. It should include both the definition and a specification of requirements. It is a set of what the system should do rather that how it should do it. The software requirements provide a basis for creating the software requirements specification. It is useful in estimating cost, planning team activities, performing tasks and tracking the teams and tracking the team’s progress throughout the development activity.

| | |
|---|---|
| • Operating system | :Linux/Windows |
| • Languages | :Python |
| • Front end | :BASH Shell |
| • Back end | :Python |

# CHAPTER 5

# SYSTEM DESIGN

## 5.1 UML DAIGRAM

UML is simply another graphical representation of a common semantic model. UML provides a comprehensive notation for the full lifecycle of object-oriented development. The Advantages are:

- To represent complete systems (instead of only the software portion) using object oriented concepts.
- To establish an explicit coupling between concepts and executable code
- To take into account the scaling factors that are inherent to complex and critical systems
- To creating a modeling language usable by both humans and machines

UML defies several models for representing systems

- The class mode captures the static structure
- The state model expresses the dynamic behavior of objects
- The use case model describes the requirements of user
- The interaction model represents the scenarios and message flow

### 5.1.1 USE CASE DAIGRAM

Use case diagrams overview the usage requirement for system.They are useful for presentations to management and project stakeholders, but for actual development you will find that use cases provide significantly more value because they describe ”the meant” of the actual requirements.

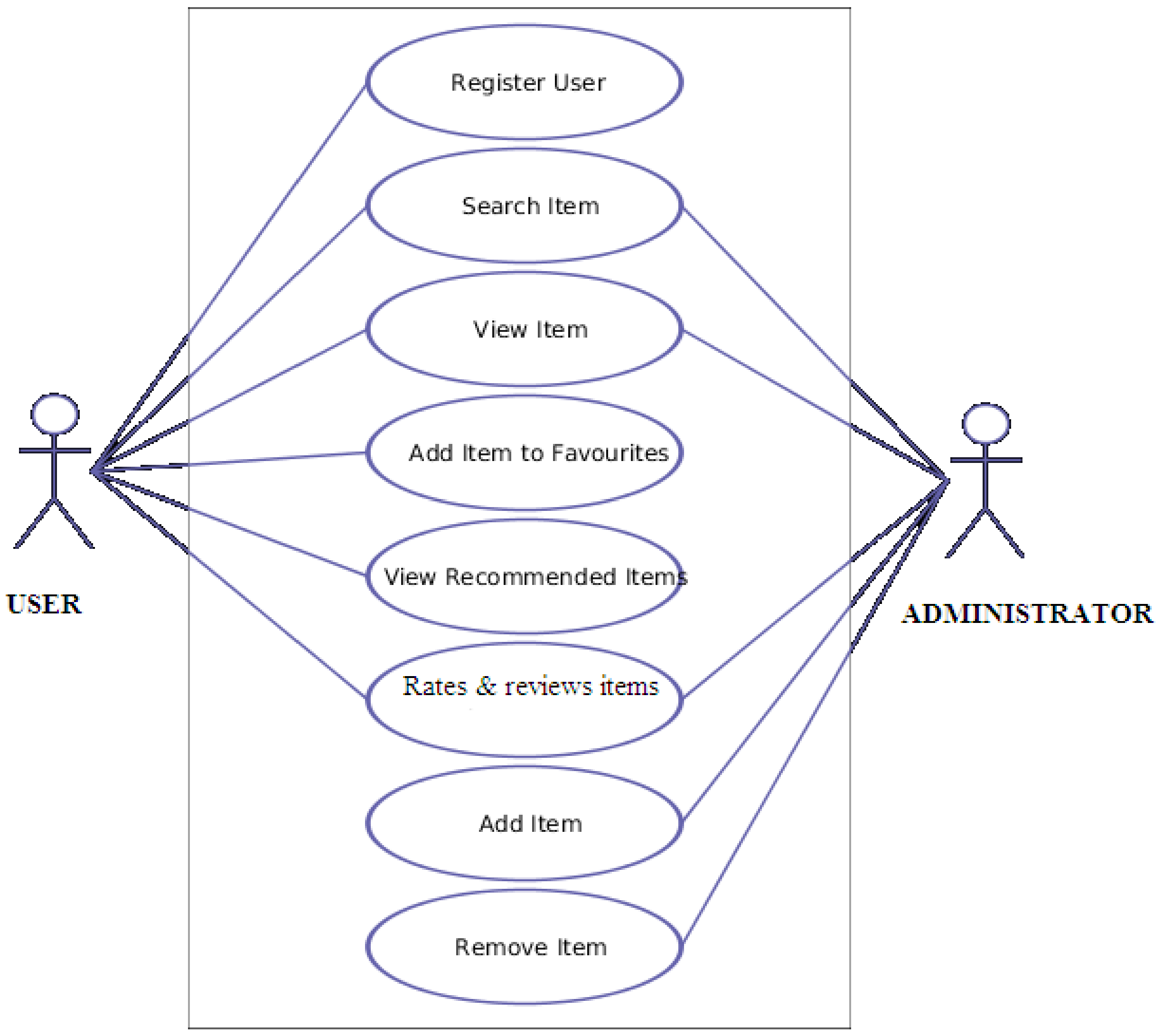


FIG 1.1 USE CASE DIAGRAM

## 5.1.2 SEQUENCE DIAGRAM

Sequence diagram, model the flow of logic within your system in a visual manner, enabling you both to document and validate your logic, and commonly used for both analysis and design purpose.

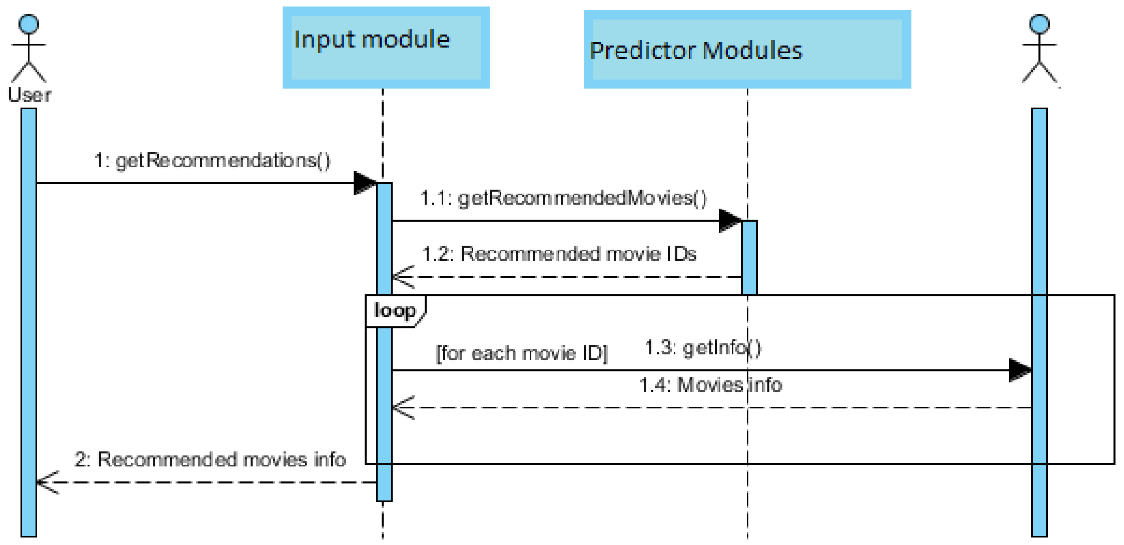


FIG 1.2 SEQUENCE DIAGRAM

# CHAPTER 6

# TESTING AND MAINTENANCE

Testing is a set of activities that can be planned in advance and conducted systematically. For this reason a template for software testing, a set of steps into which we can place specific test case design techniques and testing methods should be defined for software process. Testing often accounts for more effort than any other software engineering activity. If it is conducted haphazardly, time is wasted, unnecessary effort is expanded, and even worse, errors sneak through undetected. It would therefore seem reasonable to establish a systematic strategy for testing software.

There are two types of testing according to their behavior:

1. Unconventional testing
2. Conventional testing

## UNCONVENTIONAL TESTING

Unconventional testing is a process of verification which is done by Software Quality Assurance (SQA) team. It is a prevention technique which is performing from beginning to ending of the project development. In this process SQA team verifying the project development activities and insuring that the developing project is fulfilling the requirement of the client or not.

In this testing the SQA team follows these methods:

1. Peer review
2. Code walk and throw
3. Inspection
4. Document verification

**CONVENTIONAL TESTING**

Integration testing is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before validation testing.  Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applied tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.

## 6.1 TESTING METHODOLOGIES

### 6.1.1 UNIT TESTING

Unit testing is a software testing method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are fit for use. Intuitively, one can view a unit as the smallest testable part of an application. In procedural programming, a unit could be an entire module, but it is more commonly an individual function or procedure. In object-oriented programming, a unit is often an entire interface, such as a class, but could be an individual method. Unit tests are short code fragments created by programmers or occasionally by white box testers during the development process.

### 6.1.2 INTEGRATION TESTING

Integration Testing is a level of software testing where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units. Test drivers and test stubs are used to assist in Integration Testing. This testing is performed to expose defects in the interfaces and in the interactions between integrated components or systems.

### 6.1.3 INTERFACE TESTING

Interface testing is performed to verify the interface between sub modules while performing integration of sub modules aiding master module recursively.

### 6.1.4 VALIDATION TESTING

Validation testing ensures that the product actually meets the client's needs. It can also be defined as to demonstrate that the product fulfills its intended use when deployed on appropriate environment. This testing is referred to as “alpha” or the “beta” testing phases.

## 6.2 MAINTENANCE

The objectives of this maintenance work are to make sure that the system gets into work all time without any problems in running the code. The provisions are made for environmental changes which may affect the computer or software system. This is called the maintenance of the system. Nowadays there is rapid change in the software world. The system should be capable of the changes that occur in the recommender systems. The developed system is liable to accept any modification after its implementation. This system has been designed to favor all new changes. Doing this will not affect the systems performance or its accuracy.

## 6.3 TESTING STRATEGIES

A number of software testing strategies have been proposed in the literature. All provides the software developer with a template for testing and all have the following generic characteristics:

- Testing begins at the component level and works “outward” towards the integration of the entire computer-based system.
- Different testing techniques are appropriate at different point in time.
- The developer of the software conduct testing and for large projects, independent test group
- Testing and debugging are different activity accommodated.

# CHAPTER 7

# CONCLUSION AND FUTURE ENHANCEMENTS

## 7.1 CONCLUSION

The recommendation systems can enhanced with bias resolving algorithms that reduces bias in the recommendations that are being recommended to the user.

The basic hybrid recommendation systems are prone bias and have to be resolved. The resolved bias reduces the choice over load, and provides the user with best and the reduced number of recommendations or suggestions.

This reduces the user's time for searching the items from the list of all items available. This can be implemented as an enhancement in all of the recommendation engines that exist in real-time to help user in choosing form a comparatively smaller subset of items available.

The bias resolved hybrid recommendation systems performance measure can be evaluated to the performance of the normal hybrid system and the result of the performance evaluation denotes that bias resolved systems have better efficiency than the normal systems.

The advantage of our project is that the user can feel easy in choosing the items from recommended list and can have a bias free recommended list for searching the items. The hybrid systems become more faster and efficient if the bias and information overload is resolved.

## 7.2 FUTURE ENHANCEMENT

In future more detailed information of the items can stored by the recommender systems, and  more efficient way of handling the problem like choice overload and bias can be made. In future the recommendation systems incorporate machine learning algorithms and collaborate the user data with server data to provide suggestions at a faster rate. The several more hybridization techniques can be implemented to the hybrid recommendation systems for the user to easily select from available items.

# REFERENCES

1. J Bobadilla, F Ortega, A Hernando, 2013 proposed “Recommender Systems Survey”

2. Dirk Bollen, Bart P. Knijnenburg, Martijn C. Willemsen, Mark Graus, 2010 proposed “Understanding Choice Overload in Recommender Systems”

3. Gediminas Adomavicius, Jesse Bockstedt, Shawn Curley, and Jingjing Zhang , 2014 proposed “De-Biasing User Preference Ratings in Recommender Systems”
    a. .
4. Sanjay Krishnan, Jay Patel, Michael J. Franklin, Ken Goldberg, 2014 proposed “Social Influence Bias in Recommender Systems”

5. Asela Gunawardana, Christopher Meek , 2009 proposed “A Unified Approach to Building Hybrid Recommender Systems ”

6. Adomavicius, G., Bockstedt, J., Curley, S., and Zhang, J. 2013. "Do Recommender Systems Manipulate Consumer Preferences? A Study of Anchoring Effects," Information Systems Research, 24(4)

7. Bell, R.M., and Koren, Y. 2007. "Improved NeighborhoodBased Collaborative Filtering," KDDCup'07, San Jose, CA, USA, 7-14.

8. Cosley, D., Lam, S., Albert, I., Konstan, J.A., and Riedl, J. 2003. "Is Seeing Believing? How Recommender Interfaces Affect Users' Opinions," CHI 2003 Conference, Fort Lauderdale FL.

9. Koren, Y., Bell, R., and Volinsky, C. 2009. "Matrix Factorization Techniques for Recommender Systems," IEEE CS, 42, 30-37.

10. Sarwar, B., Karypis, G., Konstan, J.A., and Riedl, J. 2001. "Item-Based Collaborative Filtering Recommendation Algorithms," Int'l WWW Conference, Hong Kong, 285 - 295.

# Annexure

```
import numpy as np
from random import randint
from lightfm import LightFM
from sklearn.metrics import mean_squared_error
from sklearn.metrics import pairwise_distances
from lightfm.datasets import fetch_movielens
import pandas as pd
```

**Module 1: Hybrid recommendation system**

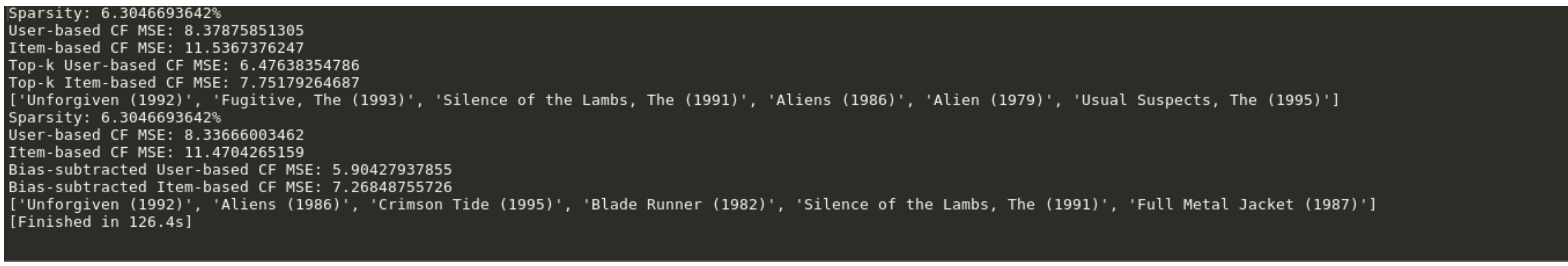
```
#previous model's implementation
def old_recom():
    names = ['user_id', 'item_id', 'rating', 'timestamp']
    df = pd.read_csv('ml-100k/u.data', sep='\t', names=names)

    #determining number of users and items
    n_users = df.user_id.unique().shape[0]
    n_items = df.item_id.unique().shape[0]

    #creating sparse matrix
    ratings = np.zeros((n_users, n_items))
    for row in df.itertuples():
        ratings[row[1]-1, row[2]-1] = row[3]

    #finding sparsity of matrix
    sparsity = float(len(ratings.nonzero()[0]))
    sparsity /= (ratings.shape[0] * ratings.shape[1])
    sparsity *= 100
    print 'Sparsity: {:4.10f}%'.format(sparsity)

    #spilitting the dataset into 2 halves: training and test
    def train_test_split(ratings):
        test = np.zeros(ratings.shape)
        train = ratings.copy()
        for user in xrange(ratings.shape[0]):
            test_ratings = np.random.choice(ratings[user, :].nonzero()[0],
                                    size=10,
                                    replace=False)
            train[user, test_ratings] = 0.
            test[user, test_ratings] = ratings[user, test_ratings]

        assert(np.all((train * test) == 0))
        return train, test

    train, test = train_test_split(ratings)

    #determining similarity for users and items seperately
    def fast_similarity(ratings, kind='user', epsilon=1e-9):
        if kind == 'user':
            sim = ratings.dot(ratings.T) + epsilon
        elif kind == 'item':
            sim = ratings.T.dot(ratings) + epsilon
        norms = np.array([np.sqrt(np.diagonal(sim))])
        return (sim / norms / norms.T)

    user_similarity = fast_similarity(train, kind='user')
    item_similarity = fast_similarity(train, kind='item')

    def predict_fast_simple(ratings, similarity, kind='user'):
        if kind == 'user':
            return similarity.dot(ratings) / np.array([np.abs(similarity).sum(axis=1)]).T
        elif kind == 'item':
            return ratings.dot(similarity) / np.array([np.abs(similarity).sum(axis=1)])


    #using scikit to find out mean squared error for our predictions to determine validity of
prediction
    def get_mse(pred, actual):
        pred = pred[actual.nonzero()].flatten()
        actual = actual[actual.nonzero()].flatten()
        return mean_squared_error(pred, actual)

    item_prediction = predict_fast_simple(train, item_similarity, kind='item')
    user_prediction = predict_fast_simple(train, user_similarity, kind='user')

    #displaying user-user and item-item validity
    print 'User-based CF MSE: ' + str(get_mse(user_prediction, test))
    print 'Item-based CF MSE: ' + str(get_mse(item_prediction, test))



    #using k-means algorithm to predict movies by hybrid filtering algorithms
    def predict_topk(ratings, similarity, kind='user', k=40):
        pred = np.zeros(ratings.shape)
        if kind == 'user':
            for i in xrange(ratings.shape[0]):
                top_k_users = [np.argsort(similarity[:,i])[:-k-1:-1]]
                for j in xrange(ratings.shape[1]):
                    pred[i, j] = similarity[i, :][top_k_users].dot(ratings[:, j][top_k_users])
                    pred[i, j] /= np.sum(np.abs(similarity[i, :][top_k_users]))
        if kind == 'item':
            for j in xrange(ratings.shape[1]):
                top_k_items = [np.argsort(similarity[:,j])[:-k-1:-1]]
                for i in xrange(ratings.shape[0]):
                    pred[i, j] = similarity[j, :][top_k_items].dot(ratings[i, :][top_k_items].T)
                    pred[i, j] /= np.sum(np.abs(similarity[j, :][top_k_items]))

        return pred

    pred = predict_topk(train, user_similarity, kind='user', k=40)
    print 'Top-k User-based CF MSE: ' + str(get_mse(pred, test))

    pred = predict_topk(train, item_similarity, kind='item', k=40)
    print 'Top-k Item-based CF MSE: ' + str(get_mse(pred, test))


    #mapping item indexes and chipping off details other than item name and year from u.data
dataset
    idx_to_movie = {}
    with open('ml-100k/u.item', 'r') as f:
        for line in f.readlines():
            info = line.split('|')
            idx_to_movie[int(info[0])-1] = info[1]

    #ordering predictions by descending order of similarities
    def top_k_movies(similarity, mapper, movie_idx, k=6):
        return [mapper[x] for x in np.argsort(similarity[movie_idx,:])[:-k-1:-1]]
    #getting user input to form known positives
    idx=202
    moviesx = top_k_movies(item_similarity, idx_to_movie, idx)

    #printing recommended items
    print(moviesx)
```

**Module 2: Hybrid recommendation algorithm without bias**

```
#our definition of hybrid recommendation algorithm without implicit rating bias
def bias_less_recom():
    names = ['user_id', 'item_id', 'rating', 'timestamp']
    df = pd.read_csv('ml-100k/u.data', sep='\t', names=names)

    #determining number of users and items
    n_users = df.user_id.unique().shape[0]
    n_items = df.item_id.unique().shape[0]

    #creating sparse matrix
    ratings = np.zeros((n_users, n_items))
    for row in df.itertuples():
        ratings[row[1]-1, row[2]-1] = row[3]

    #finding sparsity of matrix
    sparsity = float(len(ratings.nonzero()[0]))
    sparsity /= (ratings.shape[0] * ratings.shape[1])
    sparsity *= 100
    print 'Sparsity: {:4.10f}%'.format(sparsity)

    #spilitting the dataset into 2 halves: training and test
    def train_test_split(ratings):
        test = np.zeros(ratings.shape)
        train = ratings.copy()
        for user in xrange(ratings.shape[0]):
            test_ratings = np.random.choice(ratings[user, :].nonzero()[0],
                                            size=10,
                                            replace=False)
            train[user, test_ratings] = 0.
            test[user, test_ratings] = ratings[user, test_ratings]
        assert(np.all((train * test) == 0))
        return train, test

    train, test = train_test_split(ratings)

    #determining similarity for users and items seperately
    def fast_similarity(ratings, kind='user', epsilon=1e-9):
        if kind == 'user':
            sim = ratings.dot(ratings.T) + epsilon
        elif kind == 'item':
            sim = ratings.T.dot(ratings) + epsilon
        norms = np.array([np.sqrt(np.diagonal(sim))])
        return (sim / norms / norms.T)

    user_similarity = fast_similarity(train, kind='user')
    item_similarity = fast_similarity(train, kind='item')



    def predict_fast_simple(ratings, similarity, kind='user'):
        if kind == 'user':
            return similarity.dot(ratings) / np.array([np.abs(similarity).sum(axis=1)]).T
        elif kind == 'item':
            return ratings.dot(similarity) / np.array([np.abs(similarity).sum(axis=1)])

    #using scikit to find out mean squared error for our predictions to determine validity of
prediction
    def get_mse(pred, actual):
        pred = pred[actual.nonzero()].flatten()
        actual = actual[actual.nonzero()].flatten()
        return mean_squared_error(pred, actual)

    item_prediction = predict_fast_simple(train, item_similarity, kind='item')
    user_prediction = predict_fast_simple(train, user_similarity, kind='user')

    #displaying user-user and item-item validity
    print 'User-based CF MSE: ' + str(get_mse(user_prediction, test))
    print 'Item-based CF MSE: ' + str(get_mse(item_prediction, test))

    #using k-means algorithm to predict movies by hybrid filtering algorithm

    #reducing bias by considering the implicit rating bias
```

```
def predict_topk_nobias(ratings, similarity, kind='user', k=40):
    pred = np.zeros(ratings.shape)
    if kind == 'user':
        user_bias = ratings.mean(axis=1)
        ratings = (ratings - user_bias[:, np.newaxis]).copy()
        for i in xrange(ratings.shape[0]):
            top_k_users = [np.argsort(similarity[:,i])[:-k-1:-1]]
            for j in xrange(ratings.shape[1]):
                pred[i, j] = similarity[i, :][top_k_users].dot(ratings[:, j][top_k_users])
                pred[i, j] /= np.sum(np.abs(similarity[i, :][top_k_users]))
        pred += user_bias[:, np.newaxis]
    if kind == 'item':
        item_bias = ratings.mean(axis=0)
        ratings = (ratings - item_bias[np.newaxis, :]).copy()
        for j in xrange(ratings.shape[1]):
            top_k_items = [np.argsort(similarity[:,j])[:-k-1:-1]]
            for i in xrange(ratings.shape[0]):
                pred[i, j] = similarity[j, :][top_k_items].dot(ratings[i, :][top_k_items].T)
                pred[i, j] /= np.sum(np.abs(similarity[j, :][top_k_items]))
        pred += item_bias[np.newaxis, :]
    return pred

#checking validity of our function by considering similarity factor
user_pred = predict_topk_nobias(train, user_similarity, kind='user')
print 'Bias-subtracted User-based CF MSE: ' + str((get_mse(user_pred, test))-1)

item_pred = predict_topk_nobias(train, item_similarity, kind='item')
print 'Bias-subtracted Item-based CF MSE: ' + str((get_mse(item_pred, test))-1)

#mapping item indexes and chipping off details other than item name and year from u.data
dataset
idx_to_movie = {}
with open('ml-100k/u.item', 'r') as f:
    for line in f.readlines():
        info = line.split('|')
        idx_to_movie[int(info[0])-1] = info[1]

#ordering predictions by descending order of similarities
def top_k_movies(similarity, mapper, movie_idx, k=6):
    return [mapper[x] for x in np.argsort(similarity[movie_idx,:])[:-k-1:-1]]
    #further reducing bias by considering pearson coefficient

    #using pairwise distances function from scikit learn framework to increase similarity
    item_correlation = 1 - pairwise_distances(train.T, metric='correlation')
    item_correlation[np.isnan(item_correlation)] = 0

    #getting user input to form known positives
    idx=202
    moviesx = top_k_movies(item_correlation, idx_to_movie, idx)

    #printing recommended items
    print(moviesx)


#running the recommendation engine with bias
old_recom()

#running our version of recommendation engine without bias
bias_less_recom()
```

**OUTPUT:**



```
Sparsity: 6.3046693642%
User-based CF MSE: 8.37875851305
Item-based CF MSE: 11.5367376247
Top-k User-based CF MSE: 6.47638354786
Top-k Item-based CF MSE: 7.75179264687
['Unforgiven (1992)', 'Fugitive, The (1993)', 'Silence of the Lambs, The (1991)', 'Aliens (1986)', 'Alien (1979)', 'Usual Suspects, The (1995)']
Sparsity: 6.3046693642%
User-based CF MSE: 8.33666003462
Item-based CF MSE: 11.4704265159
Bias-subtracted User-based CF MSE: 5.90427937855
Bias-subtracted Item-based CF MSE: 7.26848755726
['Unforgiven (1992)', 'Aliens (1986)', 'Crimson Tide (1995)', 'Blade Runner (1982)', 'Silence of the Lambs, The (1991)', 'Full Metal Jacket (1987)']
[Finished in 126.4s]
```