

Western Kentucky University

TopSCHOLAR®

---

Masters Theses & Specialist Projects

Graduate School

---

Fall 2019

## A Comparative Study of Recommendation Systems

Ashwini Lokesh

Western Kentucky University, ashwinilok@gmail.com

Follow this and additional works at: <https://digitalcommons.wku.edu/theses>



Part of the [Computer Sciences Commons](#), and the [Engineering Commons](#)

---

### Recommended Citation

Lokesh, Ashwini, "A Comparative Study of Recommendation Systems" (2019). *Masters Theses & Specialist Projects*. Paper 3166.

<https://digitalcommons.wku.edu/theses/3166>

This Thesis is brought to you for free and open access by TopSCHOLAR®. It has been accepted for inclusion in Masters Theses & Specialist Projects by an authorized administrator of TopSCHOLAR®. For more information, please contact [topscholar@wku.edu](mailto:topscholar@wku.edu).

# A COMPARATIVE STUDY OF RECOMMENDATION SYSTEMS

A Thesis  
Presented to  
School of Engineering and Applied Sciences  
Western Kentucky University  
Bowling Green, Kentucky

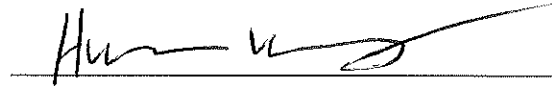
In Partial Fulfillment  
Of the Requirements for the Degree  
Master of Science

By  
Ashwini Lokesh

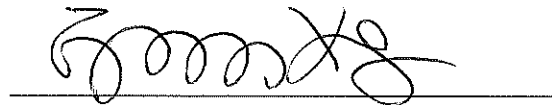
December 2019

A COMPARATIVE STUDY OF RECOMMENDATION SYSTEMS

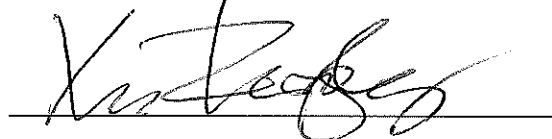
Date Recommended 11/21/2019




Dr. Huanjing Wang, Director of Thesis



Dr. Guangming Xing, Committee Member



Dr. Zhonghang Xia, Committee Member

 11/22/19  
Dean, Graduate Studies and Research      Date

I dedicate this thesis to my parents, Lokesh Narayanaswamy and Chandrika Lokesh, and my brother, Arjun Lokesh, who are a great inspiration to me and my biggest support system, also the reason for where I am today. A special mention to Shreyas Suresh Hassan, who has been ever so kind and patient throughout and kept me going. I would also like to thank my friends, Gayathri Suresh and Rutuja Chinchankar for guiding me through and being there for me during my stay at WKU.

## ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Huanjing Wang for the constant support and being so patient. I really appreciate the freedom she has given me with exploring different concepts in recommendation systems. I specially thank my committee members, Dr. Guangming Xing who has helped me immensely to deal and handle various difficult situations during my studies, and Dr. Zhonghang Xia who has taught me to challenge and push myself further with every project from CS 500 (Proficiency) to CS 543 (Advanced Database).

## TABLE OF CONTENTS

Chapter 1. Introduction.....	1
1.1 The Origin of Recommendation Systems .....	1
1.2 What Does A Recommendation System Do? .....	2
1.3 How Does A Recommendation System Work?.....	3
1.4 Types of Recommendation Systems.....	5
1.4.1 Collaborative Filtering Recommendation System .....	5
1.4.2 Content-Based Filtering Recommendation System .....	8
1.4.3 Hybrid Recommendation System .....	9
1.5 Overview.....	10
Chapter 2. Background.....	11
2.1 Collaborative Filtering Based Method.....	11
2.1.1 User Based Collaborative Filtering (UBCF).....	15
2.1.2 Item Based Collaborative Filtering (IBCF).....	16
2.2 Content-Based Filtering Method .....	17
2.2.1 Vectors .....	18
2.2.2 TF – IDF.....	18
2.2.3 Similarity Measures .....	20
Chapter 3. Evaluation Criterion.....	22
3.1 Cross Validation .....	22
3.2 Root Mean Square Error (RMSE) .....	23
3.3 Mean Absolute Error (MAE).....	25
3.4 Qualitative and Quantitative Analysis .....	25
Chapter 4. Literature Survey .....	26
Chapter 5. Experiments .....	28
5.1 Dataset .....	28
5.2 Methodology.....	32
5.2.1 Collaborative Filtering Based Recommendation .....	32
5.2.2 Content-Based Filtering Recommendation .....	32
5.2.3 Hybrid Recommendation .....	32
5.3 Result Analysis .....	33
5.3.1 Quantitative Analysis .....	33
5.3.2 Qualitative Analysis .....	38

Chapter 6. Conclusions And Future Work .....	41
References .....	42
APPENDIX .....	45

## LIST OF FIGURES

Figure 1. Objective Function of a Recommendation System.....	3
Figure 2. Collaborative Filtering based Recommendation System .....	6
Figure 3. User Based Collaborative Filtering (UBCF).....	7
Figure 4. Item Based Collaborative Filtering (IBCF) .....	7
Figure 5. Content-Based Filtering Recommendation System .....	9
Figure 6. Hybrid Recommendation System .....	10
Figure 7. Matrix Factorization Based Recommendation Algorithm .....	13
Figure 8. Histogram of Users' Average Ratings .....	29
Figure 9. Histogram of Items' Average Ratings .....	29
Figure 10. Histogram of Ratings .....	30
Figure 11. Histogram of Items Rated by Users .....	30
Figure 12. Histogram of Users who rated Items.....	31
Figure 13. Most Popular Genres.....	31
Figure 14. RMSE of Collaborative Filtering Based and Hybrid Recommendation System .....	33
Figure 15. Average RMSE of Collaborative Filtering Based and Hybrid Recommendation System .....	34
Figure 16. MAE of Collaborative Filtering Based and Hybrid Recommendation System .....	34
Figure 17. Average MAE of Collaborative Filtering Based and Hybrid Recommendation System .....	35
Figure 18. RMSE of Collaborative Filtering Based and Hybrid Recommendation System for 5 sets of users .....	36
Figure 19. Average RMSE of Collaborative Filtering Based and Hybrid Recommendation System for 5 sets of users .....	36
Figure 20. MAE of Collaborative Filtering Based and Hybrid Recommendation System for 5 sets of users .....	37
Figure 21. Average MAE of Collaborative Filtering Based and Hybrid Recommendation System for 5 sets of users .....	37



## LIST OF TABLES

Table 1. Some Popular Sites that Use Recommendation System .....	2
Table 2. Top 20 Recommended Movies for a Particular User by Collaborative Filtering Based Recommendation System .....	38
Table 3. Top 20 Recommended Movies for a Particular Movie by Content-Based Filtering Recommendation System .....	39
Table 4. Top 20 Recommended Movies for a Particular User and Movie by Hybrid Recommendation System .....	40

## A COMPARATIVE STUDY OF RECOMMENDATION SYSTEMS

Ashwini Lokesh

December 2019

49 Pages

Directed by: Dr. Huanjing Wang, Dr. Guangming Xing, Dr. Zhonghang Xia

School of Engineering and Applied Sciences

Western Kentucky University

Recommendation Systems or Recommender Systems have become widely popular due to surge of information at present time and consumer centric environment. Researchers have looked into a wide range of recommendation systems leveraging a wide range of algorithms. This study investigates three popular recommendation systems in existence, Collaborative Filtering, Content-Based Filtering, and Hybrid recommendation system. The famous MovieLens dataset was utilized for the purpose of this study. The evaluation looked into both quantitative and qualitative aspects of the recommendation systems. We found that from both the perspectives, the hybrid recommendation system performs comparatively better than standalone Collaborative Filtering or Content-Based Filtering recommendation system.

## **Chapter 1. Introduction**

In the present digital world, we all come across Recommendation System (RS) in most aspects. This chapter studies the origin of Recommendation Systems and what it does. The different categories are discussed in detail and the examples for each category gives a better overview and understanding. An empirical analysis has been conducted to compare the different types of recommendation systems and conclusions manifest which is the better one. Also, at the end of the chapter, we will see the organization of the thesis and what is included in each of the coming chapters.

### **1.1 The Origin of Recommendation Systems**

Since early civilization times, people have always depended on recommendations for every large or small decision. Whether it's an opinion (recommendation) coming from an experienced person or when more than two or three people recommend the same, the person is most likely to take their opinion. In the current internet era, recommendation systems came into existence rooted by the same idea as above. Recommendation Systems are tools that provide recommendations to the end-users based on their likes or based on the likes of the similar users. This divides the recommendation systems into two main categories: Content-Based Filtering Recommendation Systems and Collaborative Filtering Recommendation Systems. We will see each of these categories in the next sections. These divisions fall under similarity-measure-based approaches while now we have moved to more sophisticated approaches like Machine learning and Deep learning.

With the encouraging success of recommendation systems in e-commerce, movies, music, books and news recommendations, now it has become widespread across other

fields like tourism, banking amongst others. Some of the popular sites that use recommendation engines are shown in Table 1.

*Table 1. Some Popular Sites that Use Recommendation System*

Site	What is recommended?
Amazon	Consumer Product
Netflix	Movies and TV Series
Facebook	Friend Suggestion
CareerBuilder	Jobs
YouTube	Videos
Tinder	Dates

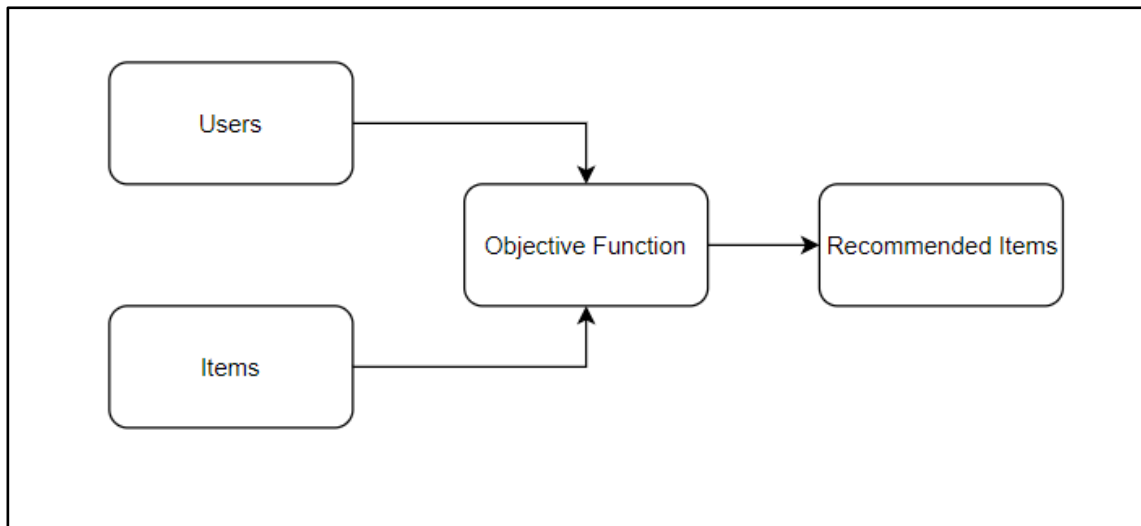
With the recent trend of an ever-increasing big data, it gave way to analyzing large datasets and implementing recommendation systems. Let us understand what a recommendation system does and how it works?

## **1.2 What Does A Recommendation System Do?**

By the definition in [1] “A recommender system or a recommendation system is a subclass of information filtering system that seeks to predict the 'rating' or 'preference' a user would give to an item”. And once a prediction has been made, according to the results of the predictions recommendations or suggestions are provided to the user. As simple as this sounds, there is a lot more that goes into how this exactly works, and we can see that in our next section.

### 1.3 How Does A Recommendation System Work?

The primary objective of a Recommendation System is to build an objective function or a mathematical model for the end-users and the specific items. This objective function should be able to measure the usefulness of the item for the user. This results in a Recommendation System and by optimizing the objective function we can obtain a better Recommendation System. Figure 1 shows the objective function. And in order to build this system, we first collect the user data. The data can be collected explicitly or implicitly. Implicit data could be gathered from order history, click on certain items or the number of times a song is listened to while the explicit data is obtained from ratings and feedback from the users.



*Figure 1. Objective Function of a Recommendation System*

The three main steps in building a Recommendation System are: loading and formatting the data, calculating the similarity between the users or between the items and predicting the unknown ratings for the users. As discussed earlier, the data can be collected explicitly or implicitly, and the recommendation system turns out to be more optimized with an increase in the available data. This data can be stored either in NoSQL format for unstructured data or as SQL tables for structured data. In the latest technology, the cloud is used to store the data and can be easily retrieved. Data will be formatted: in most cases, the user-item sets are converted into a matrix known as ratings matrix. Here the end-users are represented by rows while the products are represented by columns, each cell value points to the ratings given to the product by a particular user.

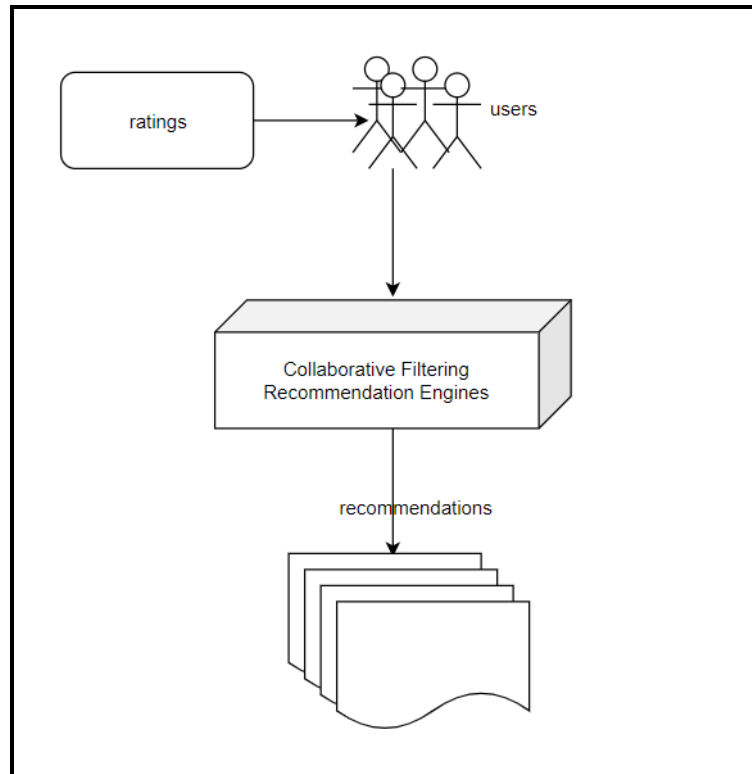
However, we should understand that not every user would have liked or rated every movie (product). So essentially this matrix is a very sparse matrix and now the question arises how do we find the missing values? Here is where we would use different similarity measures to find the missing values. Once the similarity is obtained between users or products, the unknown ratings can be predicted. For example, let us say we want to predict the ratings given by Abbey for the Product 1, we will find the similarity between Abbey and all the other users who are most similar to him. And on finding the top N users similar to Abbey, we will calculate the rating Abbey might have given to the product she hasn't rated. These steps can be repeated for each user in the data set. To calculate the unknown ratings, there are several methods and we can use something as simple as calculating the average of all the ratings for that product given by similar users. This explains the basis behind the working of recommendation systems.

## **1.4 Types of Recommendation Systems**

There are several ways to build a Recommendation System and these approaches are divided according to the needs of the application. Though there are several types of Recommendation Systems, they are mainly categorized into three types: Collaborative Filtering, Content-Based Filtering, and Hybrid Recommendation Systems. In this section, let us have a look at each of these types.

### **1.4.1 Collaborative Filtering Recommendation System**

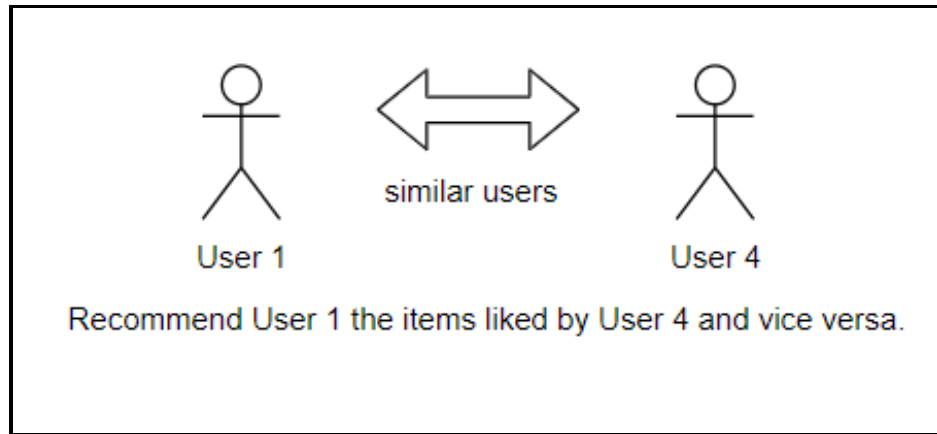
From the technology aspect, the recommenders are moving from Machine learning approaches to more advanced Neural Network and Deep Learning approaches. The idea of a Collaborative Filtering is very simple; given the ratings of a user, find all the users similar to the active user who had similar preferences in the past and then make predictions regarding all unknown products that the active user has not rated but are being rated in their neighborhood while considering the preferences or tastes of neighbors. We first calculate how similar the other users are to the active user and then unrated items from the user community are recommended to the user following predictions. Here the active user is the person to whom the system is serving recommendations. In these types of systems, the main actors are the users and the product information such as ratings, rankings and liking towards the product. Figure 2 represents Collaborative Filtering Based Recommendation System.



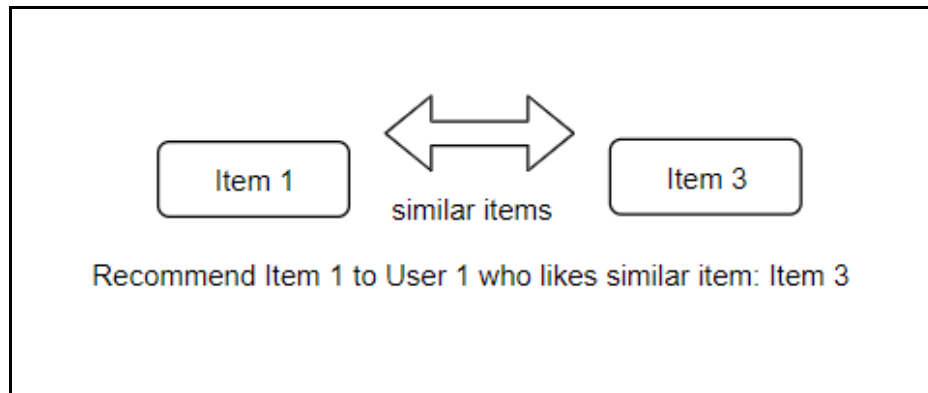
*Figure 2. Collaborative Filtering based Recommendation System*

There are two main flavors in Collaborative Filtering: User-based Collaborative Filtering (UBCF) as seen in Figure 3 and Item-based Collaborative Filtering (IBCF) denoted by Figure 4. As the names suggest, similarities are measured between users and in the former and items in the latter. The basic concept behind these RS is that if a user likes a certain item, he is most likely to like a similar item and if two users are similar, they're most likely to have a common interest. In UBCF, the similarities are calculated between users and the unrated items are recommended to a similar user. In IBCF, the similarities between items are calculated and the unrated similar item is recommended to the active user.





*Figure 3. User Based Collaborative Filtering (UBCF)*

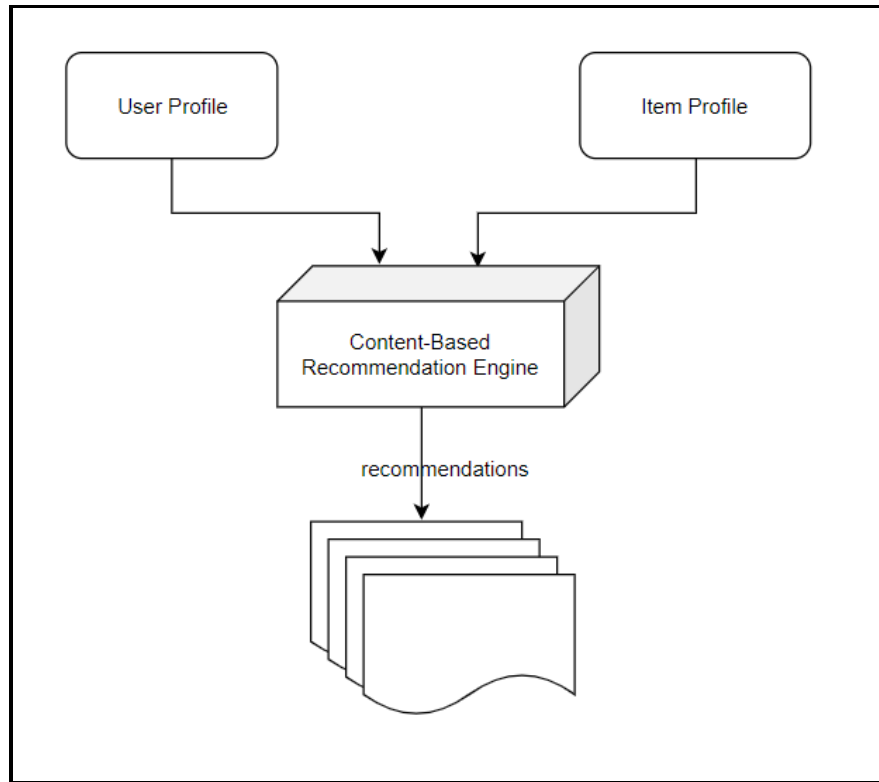


*Figure 4. Item Based Collaborative Filtering (IBCF)*

For Collaborative Filtering, we will require a collection of users who have interacted with the application, the item information and each users' ratings of items. There is no requirement for detailed information about the items in Collaborative Filtering and hence it can be an easy start. However, this kind faces a cold-start problem while recommending to new users who do not have any rating information.

### **1.4.2 Content-Based Filtering Recommendation System**

Figure 5 depicts building a Content-Based Filtering Recommendation Systems which involve three main steps: Generating content information for products, generating user profiles and preferences relative to the features of the product, and generating recommendations and predicting a list of items that the user might like. In the Item-Profile generation, the features of the product are extracted that represent the product. These features can be structured or unstructured data. For example, in the case of movies, an item profile for each movie can consist of different types of genres. This is carried out by creating a matrix with items as rows and genre as columns. Binary representation is used to show if the movie belongs to a certain genre (denoted by 1) or if it does not (denoted by 0). In the User Profile Generation step, a preference matrix is built matching the product content and the user profile and item profile are compared and calculate the similarity between them. The cold start problem is easily handled.

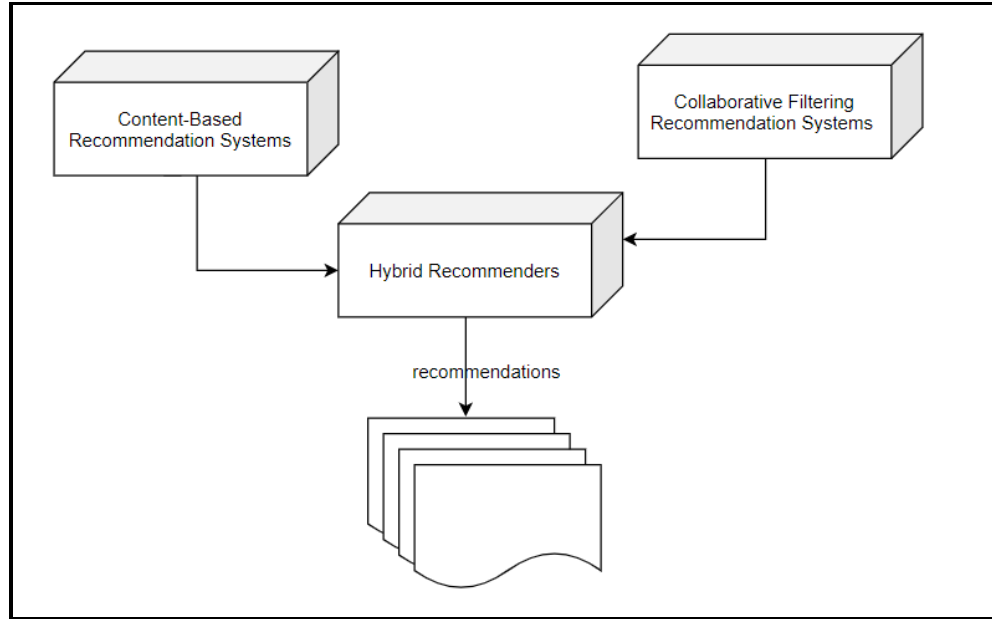


*Figure 5. Content-Based Filtering Recommendation System*

### **1.4.3 Hybrid Recommendation System**

Hybrid recommendation systems came into existence due to the limitations of each of the previous kinds. Hence there have been several strategies to combine Collaborative Filtering and Content-Based Filtering and is called Hybrid Recommender Engine as shown in Figure 6. Amongst the various approaches used, weighted method is the most common. In the beginning, the combination of the recommendation results is obtained from each and equal weights are distributed to each of these results and gradually the weights are adjusted after evaluating the responses from the users to the recommendations. Feature Combination method is another popular approach where the User profile from Content-Based Filtering

is combined with user-item ratings information and a new strategy is considered to build a Hybrid Recommendation System.



*Figure 6. Hybrid Recommendation System*

## **1.5 Overview**

In this thesis, different experimental results are obtained showing the performance of different types of recommendation systems. In Chapter 1, we saw what an RS does and it's working. Chapter 2 discusses the concepts necessary to implement the recommendation systems. In Chapter 3, we elaborate the evaluation techniques and metrics that we have used for the experiments. Chapter 4, describes all some prominent works in this area. The dataset, methodology and the results are analyzed in Chapter 5. We then make the conclusion and finish with future work in Chapter 6.

## **Chapter 2. Background**

### **2.1 Collaborative Filtering Based Method**

Collaborative recommender systems are one of the most commonly used systems at present, just like Content-Based Filtering recommender systems. According to the definition from [2], “Collaborative Filtering (CF) is the predictive process behind recommendation engines. Recommendation engines analyze information about users with similar tastes to assess the probability that a target individual will enjoy something, such as a video, a book or a product”. Collaborative Filtering is also known as social filtering. As we have already seen in chapter one, the first step in carrying out Collaborative Filtering to predict the unknown ratings of the users and these predictions are based on users’ historical behaviors; specifically, users’ preference for a set of items using the past experiences. Collaborative recommender systems can differ from each other in the way a rating is defined. They can be binary, focused on the shift in opinion over time, model or memory-based. There are several approaches to Collaborative Filtering in which the role changes depending on the approach.

A couple of very basic algorithms that can be used for Collaborative Filtering are NormalPredictor and BaselineOnly. NormalPredictor algorithm predicts a random rating based on the distribution of the training set, which is assumed to be normal. BaselineOnly algorithm predicts the baseline estimate for a given user and item.

There are also multiple k-NN based algorithms. k-NN is a non-parametric, lazy learning method. It uses a database in which the data points are separated into several clusters to make inferences for new samples. k-NN does not make any assumptions on the

underlying data distribution but it relies on item feature similarity. For example, when k-NN makes inference about a movie, k-NN will calculate the “distance” between the target movie and every other movie in its database, then it ranks its distances and returns the top k nearest neighbor movies as the most similar movie recommendations. There are variations of this algorithm that consider mean ratings or z-score normalization of each user.

Next, there are Matrix Factorization-based algorithms like Singular Value Decomposition (SVD), Singular Value Decomposition with an implicit rating and Non-negative Matrix Factorization. SVD algorithm is equivalent to Probabilistic Matrix Factorization. In this work, we use the SVD algorithm.

In the following algorithm (Figure 7), matrix of interactions is factorized into two small matrices one for users and one for items with a certain number of latent components (typically several hundred).

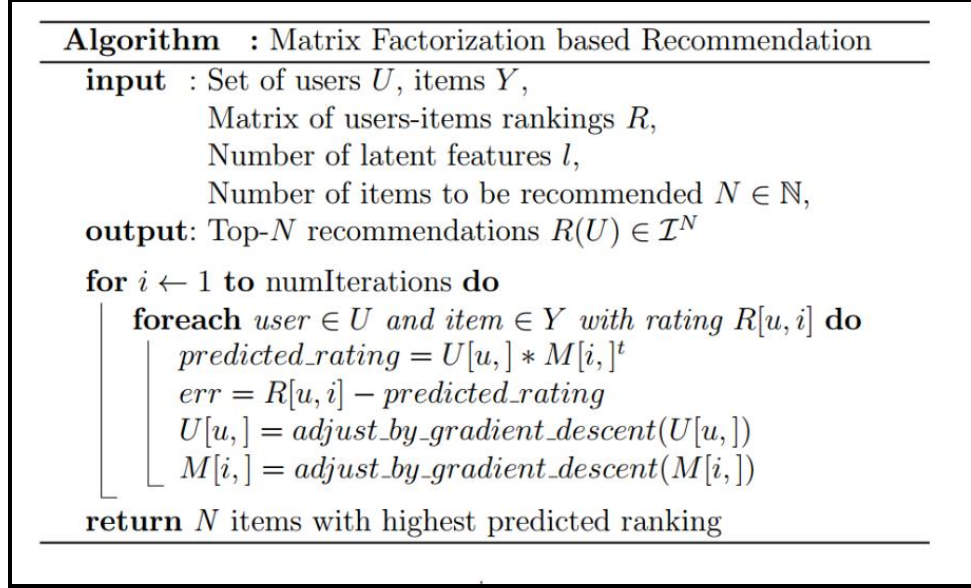


Figure 7. Matrix Factorization Based Recommendation Algorithm

The (u,i) rating is obtained by multiplying these two small matrices. There are several approaches to how to decompose matrices and train them. The one showed above is a simple gradient descent technique. The error can be minimized by Stochastic Gradient Descent, Alternating Least Squares or Coordinate Descent Algorithm. There are also SVD based approaches, where the ranking matrix is decomposed into three matrices.

SVD is a matrix factorization technique that is usually used to reduce the number of features of a data set by reducing space dimensions from  $N$  to  $K$  where  $K < N$ . For recommendation systems, however, we are only interested in the matrix factorization part keeping the same dimensionality. The matrix factorization is done on the user-item ratings matrix. From a high level, matrix factorization can be thought of as finding 2 matrices whose product is the original matrix.

Each item can be represented by a vector 'q<sub>i</sub>'. Similarly, each user can be represented by a vector 'p<sub>u</sub>' such that the dot product of those 2 vectors is the expected rating.

$$\text{expected rating} = \hat{r}_{ui} = q_i^T p_u \quad (1)$$

'q<sub>i</sub>' and 'p<sub>u</sub>' can be found in such a way that the square error difference between their dot product and the known rating in the user-item matrix is minimum.

$$\text{minimum}(p, q) \sum_{(u,i) \in K} (\hat{r}_{ui} - q_i^T p_u)^2 \quad (2)$$

For our model to be able to generalize well and not over-fit the training set, we introduce a penalty term to our minimization equation. This is represented by a regularization factor  $\lambda$  multiplied by the square sum of the magnitudes of user and item vectors.

$$\text{minimum}(p, q) \sum_{(u,i) \in K} (\hat{r}_{ui} - q_i^T p_u)^2 + \lambda (||q_i^2|| + ||p_u^2||) \quad (3)$$

To illustrate the usefulness of this factor imagine we have an extreme case where a low rating given by a user to a movie with no other rating from this user. The algorithm will minimize the error by giving 'q<sub>i</sub>' a large value. This will cause all ratings from this user to other movies to be very low. This is intuitively wrong. By adding the magnitude of the vectors to the equation, giving vectors large value will minimize the equation and thus such situations will be avoided (more here).

To reduce the error between the predicted and actual value, the algorithm makes use of some characteristics of the dataset. In particular, for each user-item (u, i) pair we can extract 3 parameters.  $\mu$  which is the average ratings of all items, 'b<sub>i</sub>' which is the average



rating of item  $i$  minus  $\mu$  and ' $b_u$ ' which is the average rating given by user  $u$  minus  $\mu$  which makes the expected rating:

$$\hat{r}_{ui} = q_i^T p_u + \mu + b_i + b_u \quad (4)$$

Thus, the final equation to minimize is:

$$\begin{aligned} \text{minimum } (p, q, b_i, b_u) \sum_{(u,i) \in K} (\hat{r}_{ui} - q_i^T p_u - \mu - b_i - b_u)^2 + \\ \lambda (||q_i^2|| + ||p_u^2|| + b_i^2 + b_u^2) \end{aligned} \quad (5)$$

The above equation is minimized using a stochastic gradient descent algorithm. From a high-level perspective, SGD starts by giving the parameters of the equation we are trying to minimize initial values and then iterating to reduce the error between the predicted and the actual value each time correcting the previous value by a small factor. This algorithm uses a factor called learning rate  $\gamma$  which determines the ratio of the old value and the newly computed value after each iteration. Practically, when using high  $\gamma$  one might skip the optimal solution whereas when using low  $\gamma$  values a lot of iterations are needed to reach optimal value.

### 2.1.1 User Based Collaborative Filtering (UBCF)

The nearest neighbor approach reveals that selecting all the users' rating information is not the most feasible solution. Hence, we use only top-N similar users' information and make predictions which increase the accuracy of the model. In UBCF, the recommendations are generated by considering the preferences in the users' neighborhood. It can be done in two steps:

1. Identify similar users based on similar user preferences

2. Recommend new items to an active user based on the ratings given by the similar users

In a simple example, if the user Ashley has rated ‘Star Wars’ and ‘The Empire Strikes Back’ movies with five stars and if the user Bob has rated ‘Star Wars’ with five stars too, then it is mostly liked for Ashley and Bob to be similar to each other and hence we can recommend the movie ‘The Empire Strikes Back’ to Bob. This similarity is measured using different methods which are discussed in the next chapter.

### **2.1.2 Item Based Collaborative Filtering (IBCF)**

The recommendations are created relative to the neighborhood of items. Unlike UBCF, we first find similarities between items and then recommend non-rated items that are similar to the items the active user has rated in the past. IBCF is constructed in two steps:

1. Calculate the item similarity based on the item preferences
2. Find the top similar items to the non-rated items by the active user

Let us assume that the similarity between Toy Story and Aladdin is calculated and they happen to be very similar, now when a new user likes either of these movies, the other movie can be recommended to the user. Hence, we eliminate the cold-start problem in IBCF which is an issue that is usually faced in UBCF. So, this kind of RS fails to recommend to the first-time users whose information is not available in the system.

Experiments are conducted for each of the collaborative based filtering methods with two different similarity measures and the results are obtained to prove which Collaborative Filtering method is better.

## **2.2 Content-Based Filtering Method**

The content of an item can be a very abstract thing and therefore gives us a lot of options in terms of variable that we can use. For example, for a movie we can consider the genre, the cast, the director/directors, the movie review, etc. We can choose just one or a combination of these to use in our algorithm.

Once we select the features that we want to use, we need to transform all this data into a Vector Space Model, an algebraic representation of text documents. This is usually done using a Bag of Words model, that represents documents ignoring the order of the words. In this model, each document looks like a bag containing some words. Therefore, this method allows word modeling based on dictionaries, where each bag contains a few words from the dictionary.

TF-IDF representation is a specific implementation of a Bag of Words. This model combines how important is the word in the document (local importance), with how important is the word in the corpus (global importance). Information retrieval systems have been using the concepts of Term Frequency (TF) and Inverse Document Frequency (IDF) for quite long and now Content-Based Filtering recommenders are also making use of them. They can be used to find out the relative importance of a something like a document or movie. Another important concept here is the similarity measure that can tell how similar

items are with respect to each other. Cosine Similarity is one of the popular ones in this aspect.

### 2.2.1 Vectors

The fundamental idea is to convert the texts or words into a vector and represent in a vector space model.

### 2.2.2 TF – IDF

TF- IDF stands for Term Frequency and Inverse Document Frequency. TF-IDF helps in evaluating the importance of a word in a document.

#### 2.2.2.1 TF — Term Frequency

TF is represented in the below and it says how frequent the term ‘t’ occurs in document d.

$$(f_{t,d} = \sum_{t \in d} f(t, d)) \quad (6)$$

In order to ascertain how frequent, the term/word appears in the document and also to represent the document in vector form, the following steps can be followed. Step 1 is to create a dictionary of words (also known as bag of words) present in the whole document space. Some common words also called as stop words e.g. the, of, a, an, is etc. are ignored in this process since these words are quite common and are not of any help in choosing important words.

In our recommendation system, we deal with movie genres. Let us consider a couple of movie genres which are ‘Romantic | Drama | Historical’ and ‘Sci-Fi | Animation | Drama’. So G1 is one document, G2 is the other document. Together G1 and G2 make up

the document space. So, these can be written as: G1 — Romantic Drama Historical, G2 — Sci-Fi Animation Drama. Now creating an index of these words: 1. Romantic 2. Drama 3. Historical 4. Sci-Fi 5. Animation. Step 2 is forming the vector. The Term Frequency helps us to identify how many times the term or word appears in a document but there is also an inherent problem, TF gives more importance to words/terms occurring frequently while ignoring the importance of rare words/terms. This is not an ideal situation as rare words contain more importance or signal. This problem is resolved by IDF.

Sometimes a word/term might occur more frequently in longer documents than shorter ones; hence Term Frequency normalization is carried out.

$$TF_n = \frac{(Number\ of\ times\ term\ t\ appears\ in\ a\ document)}{(Total\ number\ of\ terms\ in\ the\ document)} \quad (7)$$

where n represents normalized.

#### 2.2.2.2 IDF - Inverse Document Frequency

Formally IDF is defined as:

$$IDF(t, D) = \log \frac{N}{|\{d \in D: t \in d\}|} \quad (8)$$

Where, N is the total number of documents in the collection also known as the cardinality of document space.

$|\{d \in D: t \in d\}|$  is the number of documents where the term t is present.

A simpler definition of IDF can be:

$$IDF = \log \frac{(Total\ number\ of\ documents)}{(Total\ number\ of\ terms\ in\ the\ document)} \quad (9)$$

Now let's take an example from our own dictionary or bag of words and calculate the IDFs.

We had 5 terms or words which are as follows: 1. Romantic 2. Drama 3. Historical 4. Sci-Fi 5. Animation and our documents were: G1 — Romantic Drama Historical, G2 — Sci-Fi Animation Drama

Now  $IDF(w1) = \log \frac{2}{1}$ ;  $IDF(w2) = \log \frac{2}{2}$ ;  $IDF(w3) = \log \frac{2}{1}$ ;  $IDF(w4) = \log \frac{2}{1}$ ;  $IDF(w5) = \log \frac{2}{1}$

Here, natural logarithm being taken and  $w1, \dots, w5$  denotes words/terms. We then again get a vector as follows: (0.30, 0, 0.30, 0.30, 0.30)

### 2.2.2.3 TF-IDF Weight

Now the final step would be to get the TF-IDF weight. The TF vector and IDF vector are converted into a matrix.

Then TF-IDF weight is represented as:

$$TF - IDF Weight = TF(t, d) * IDF(t, D) \quad (10)$$

### 2.2.3 Similarity Measures

The similarity measures can be represented on a plot, with each user (or item) denoted by the coordinates. The distance between the two coordinates gives the similarity between them. Lesser the distance, the greater will be the similarity. The first step is to find similar users (or items) and these similarities are calculated by the ratings given by the users. Amongst the most common approaches, we have used the following: Cosine similarity.

### 2.2.3.1 Cosine Similarity

Cosine similarity calculates [3] the distance between two n-dimensional vectors by the angle between them in the vector space. When this is applied to RS, we consider the item (or user) to be the n-dimensional vector and the similarity between the two as the angle between them. The smaller the angle, the more similar are the items (or users).

Let's consider a two-dimensional vector to start with. The dot product between two vectors is equal to the projection of one of them on the other. Therefore, the dot product between two identical vectors (i.e. with identical components) is equal to their squared module, while if the two are perpendicular (i.e. they do not share any directions), the dot product is zero. Generally, for n-dimensional vectors, the dot product can be calculated as shown below.

$$u \cdot v = [u_1 \quad u_2 \quad \cdots \quad u_n] \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = u_1 v_1 + u_2 v_2 + \cdots + u_n v_n = \sum_{i=1}^n u_i v_i \quad (11)$$

The dot product is important when defining the similarity, as it is directly connected to it. The definition of similarity between two vectors  $u$  and  $v$  is, in fact, the ratio between their dot product and the product of their magnitudes.

$$similarity = \cos(\theta) = \frac{u \cdot v}{||u|| ||v||} = \frac{\sum_{i=1}^n u_i v_i}{\sqrt{\sum_{i=1}^n u_i^2} \sqrt{\sum_{i=1}^n v_i^2}} \quad (12)$$

By applying the definition of similarity, this will be in fact equal to 1 if the two vectors are identical, and it will be 0 if the two are orthogonal. In other words, the similarity is a number bounded between 0 and 1 that tells us how much the two vectors are similar.

## Chapter 3. Evaluation Criterion

Now, we will introduce different techniques that evaluate whether the model overfits or underfits. The ultimate goal for any model is to perform well for any future data. So, how do we go about this? The dataset that is used is divided into two sections: training and test. The training data is used to train the model while test data is used to evaluate it. In an ideal situation, we segregate the dataset in the ratio 8:2 with 80% of training and 20% is used for test. Letting the data to be distributed non-linearly and fitting it with a linear model can lead the data to be underfitting and this model does not work well with training data. Meanwhile, overfitting performs well with training data but performs badly with test data. Here model fits well over the data distribution area. Some of the evaluation methods we have used are as follows:

### 3.1 Cross Validation

Cross-validation is a statistical method used to estimate the skill of machine learning models. Two types of cross-validation can be distinguished: exhaustive and non-exhaustive cross-validation. Exhaustive cross-validation includes leave-one-out and leave-p-out cross-validation. On the other hand, non-exhaustive cross-validation includes k-fold cross-validation, Holdout method and Repeated random sub-sampling validation. In this thesis, we have used k-fold cross-validation.

The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as  $k = 10$  becoming 10-fold cross-validation.



Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model. It is a popular method because it is simple to understand and because it generally results in a less biased or less optimistic estimate of the model skill than other methods, such as a simple train/test split.

The general procedure is as follows:

- Shuffle the dataset randomly.
- Split the dataset into  $k$  groups
- For each unique group:
  - Take the group as a hold out or test data set
  - Take the remaining groups as a training data set
  - Fit a model on the training set and evaluate it on the test set
  - Retain the evaluation score and discard the model
- Summarize the skill of the model using the sample of model evaluation scores

Importantly, each observation in the data sample is assigned to an individual group and stays in that group for the duration of the procedure. This means that each sample is given the opportunity to be used in the hold out set 1 time and used to train the model  $k-1$  times.

### **3.2 Root Mean Square Error (RMSE)**

RMSE is a standard way to measure the error of a model in predicting quantitative data. Formally it is defined as follows:

$$RMSE = \sqrt{\sum_{i=1}^n \frac{(\hat{y}_i - y_i)^2}{n}} \quad (13)$$

Here,  $\hat{y}_1, \hat{y}_2, \dots, \hat{y}_n$  are predicted values.

$y_1, y_2, \dots, y_n$  are observed values.

$n$  is number of observations

The division by  $n$  under the square root in RMSE allows us to estimate the standard deviation  $\sigma$  of the error for a typical single observation rather than some kind of “total error”. By dividing by  $n$ , we keep this measure of error consistent as we move from a small collection of observations to a larger collection (it just becomes more accurate as we increase the number of observations). To phrase it another way, RMSE is a good way to answer the question: “How far off should we expect our model to be on its next prediction?”

RMSE is a good measure to use if we want to estimate the standard deviation  $\sigma$  of a typical observed value from our model’s prediction, assuming that our observed data can be decomposed as:

$$\begin{aligned} & \text{observed value} \\ &= \text{predicted value} \\ &+ \text{predictably distributed random noise with mean zero} \quad (14) \end{aligned}$$

The random noise here could be anything that our model does not capture (e.g., unknown variables that might influence the observed values). If the noise is small, as estimated by RMSE, this generally means our model is good at predicting our observed data, and if RMSE is large, this generally means our model is failing to account for important features underlying our data.

### 3.3 Mean Absolute Error (MAE)

MAE is one of the many metrics for summarizing and assessing the quality of a machine learning model. Here, error refers to the subtraction of Predicted value from Actual Value as below.

$$\text{Prediction Error} = \text{Actual Value} - \text{Predicted Value} \quad (15)$$

This prediction error is taking for each record after which we convert all error to positive. This is achieved by taking Absolute value for each error as below:

$$\text{Absolute Error} \rightarrow |\text{Prediction Error}| \quad (16)$$

Finally, we calculate the mean for all recorded absolute errors (Average sum of all absolute errors).

$$MAE = \frac{\sum_{i=1}^n |y_i - x_i|}{n} \quad (17)$$

Here,  $y_i$  is the predicted value,  $x_i$  is the actual value and  $n$  is the number of observations.

### 3.4 Qualitative and Quantitative Analysis

The comparison among the different systems in this study has two distinct aspects. The quantitative aspect relies on metrics like RMSE and MAE that were described in the previous subsections. But the qualitative aspect relies on the quality of the recommendation and we evaluate it by eyeballing the generated recommendation.

## Chapter 4. Literature Survey

Recommender systems have the capability to provide users with customized and tailored recommendations. Thus, it resolves the issue of information overload that plagues the users at the modern age. Recently, various approaches for building recommendation systems have been developed, which can utilize either Collaborative Filtering, Content-Based Filtering or hybrid filtering [4], [5], [6], [7].

Collaborative Filtering technique has reached a certain maturity and is one of the most commonly implemented systems. A multitude of different application areas has adopted Collaborative Filtering based recommendation systems. One of them is GroupLens, a news-based architecture that assists users to locate articles from huge news databases using collaborative methods [8]. Amazon improved its recommendation system by implementing topic diversification algorithms [9].

On the other hand, Content-Based Filtering techniques focus finding the similarities between content properties and user characteristics. Content-Based Filtering techniques normally base their predictions on user's information, and they ignore contributions from other users as with the case of collaborative techniques [10]. Letizia predicts the pages that a user may be interested in by tracing his movements through the sites and thus uses Content-Based Filtering method [11].

Despite the widespread success of these two filtering techniques, they have several limitations. While Content-Based Filtering techniques have issues like limited content analysis, overspecialization and sparsity of data [8], collaborative approaches have issues like cold-start, sparsity, and scalability. These issues make it difficult to use these systems

in live production. Hybrid filtering, using the combination of two or more filtering techniques in different ways to increase the performance and accuracy of recommender systems has thus been proposed to alleviate these issues. [12], [13]. These hybrid systems try to leverage the strength of each method without sacrificing any capability due to inherent weakness [14].

## Chapter 5. Experiments

### 5.1 Dataset

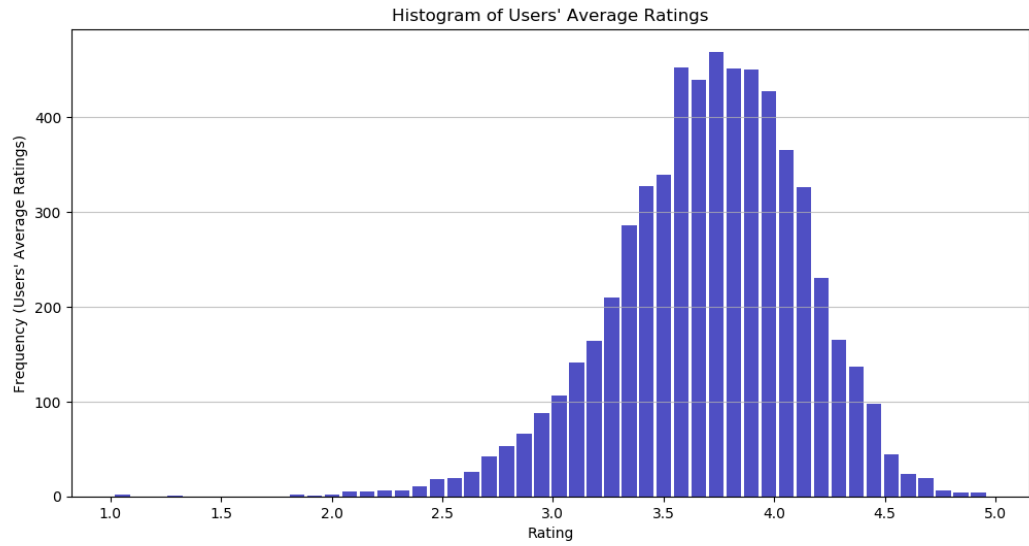
We used ‘MovieLens 1M Dataset’ [15] for our analysis. The dataset contains 1,000,209 anonymous ratings of approximately 3,900 movies made by 6,040 MovieLens users who joined MovieLens in 2000. We used two files in particular, namely ratings and movies. The ratings file contained 4 fields. They are UserID, MovieID, Rating and Timestamp.

- UserIDs range between 1 and 6040
- MovieIDs range between 1 and 3952
- Ratings are made on a 5-star scale (whole-star ratings only)
- Timestamp is represented in seconds since the epoch
- Each user has at least 20 ratings.

The movies file contained 3 fields. They are MovieID, Title and Genres

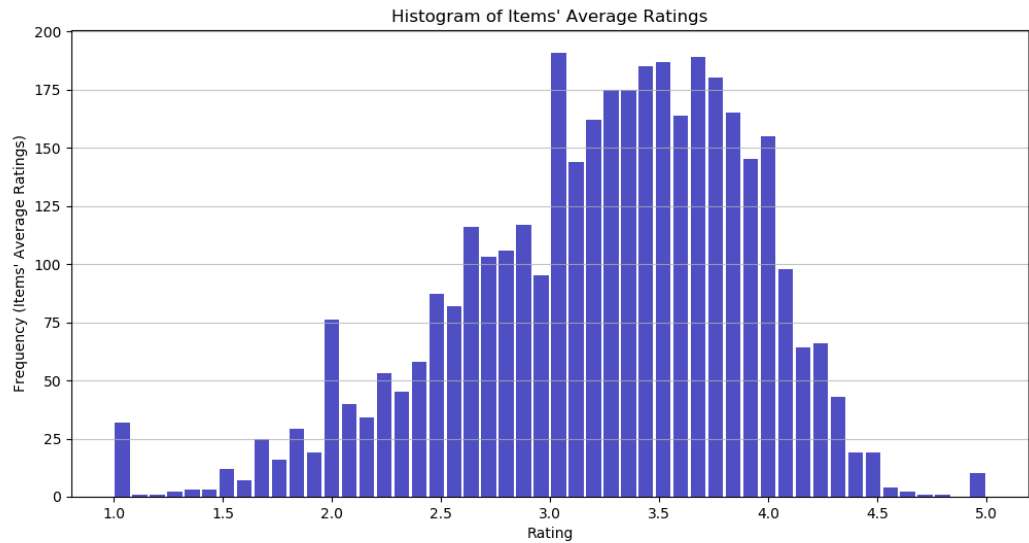
- Titles are identical to titles provided by the IMDB (including year of release)
- Genres are pipe-separated and are selected from the following genres: Action, Adventure, Animation, Children's, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, Musical, Mystery, Romance, Sci-Fi, Thriller, War, Western

We performed some initial exploratory analysis on the datasets. Figure 8 illustrates the histogram of average ratings given by the users. We can see this plot approximates a normal distribution with a left heavy tail. Most users’ average ratings fall between 3.5 and 4.



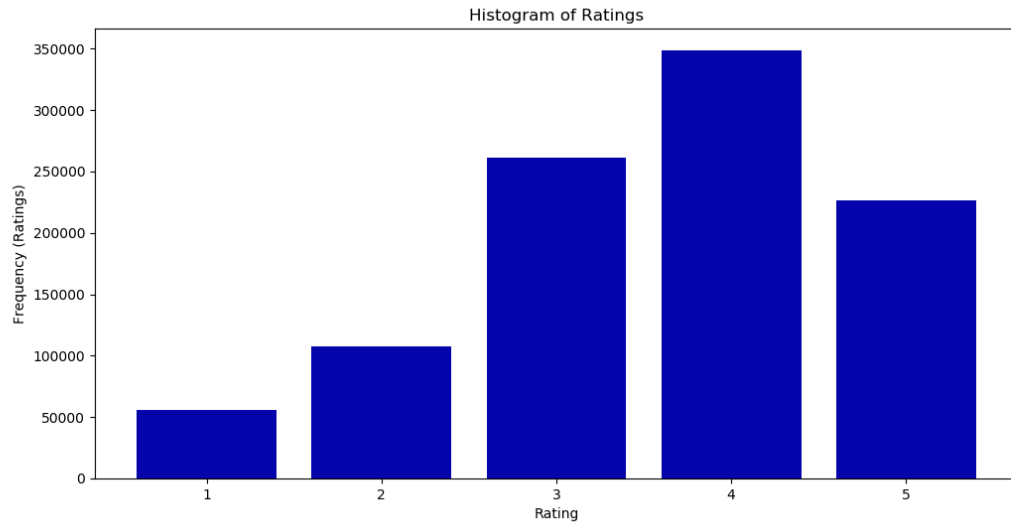
*Figure 8. Histogram of Users' Average Ratings*

Figure 9 illustrates the histogram of average ratings that items got. This plot also approximates a normal distribution with a left heavy tail. However, in this case, the values are more spread out. Most items have been rated between 3 to 4.



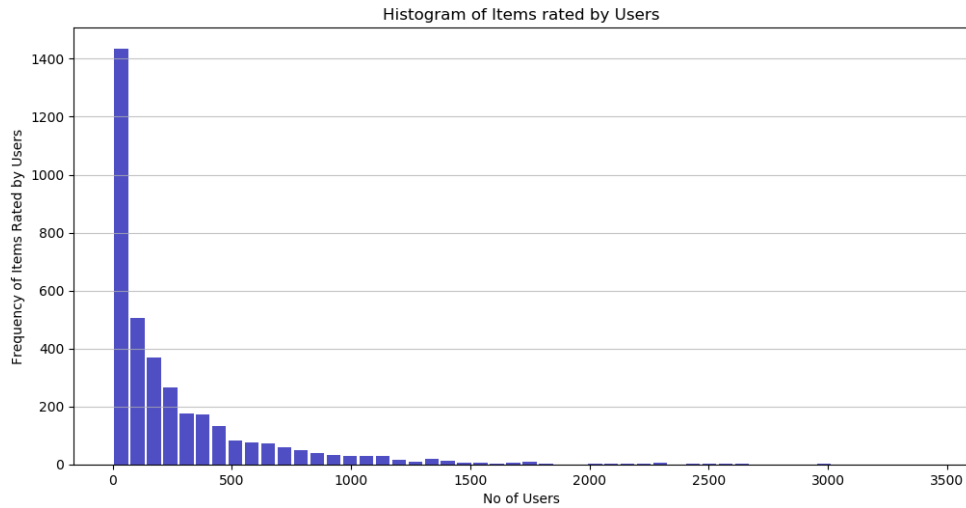
*Figure 9. Histogram of Items' Average Ratings*

Figure 10 shows the histogram of ratings. It is consistent with the previous two plots as we see that the most frequent ratings are 4 and 3 respectively.



*Figure 10. Histogram of Ratings*

Figure 11 and Figure 12 illustrates the histogram of items rated by users and users who rated items. As expected from these two plots, most users rate very few items.



*Figure 11. Histogram of Items Rated by Users*



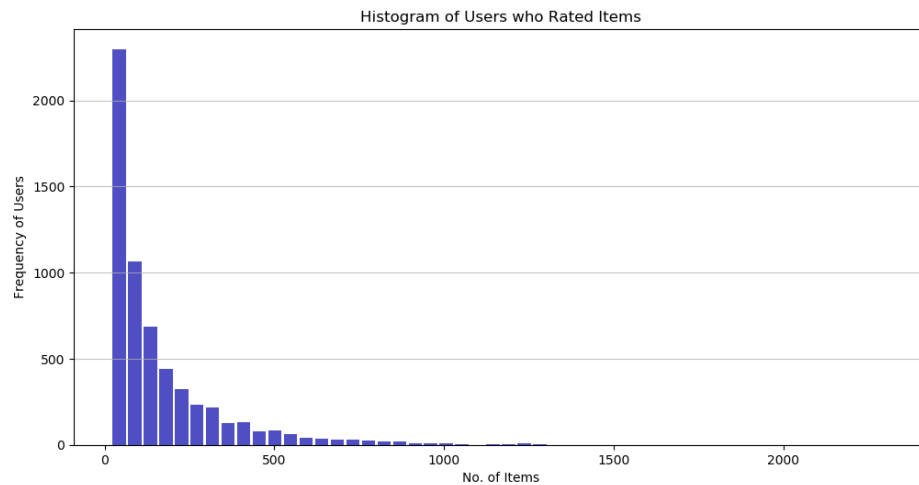


Figure 12. Histogram of Users who rated Items

We also generated a word cloud of the genres of the movies. A word cloud is a visual representation of text data, typically used to depict particular words where the importance of each word is shown with font size or color. This format is useful for quickly perceiving the most prominent terms to determine its relative prominence. Figure 13 shows some of the most popular genres. As we can see in the word cloud drama and comedy are the most common ones.



Figure 13. Most Popular Genres

## **5.2 Methodology**

### **5.2.1 Collaborative Filtering Based Recommendation**

We used SVD and 5-fold cross-validation for our Collaborative Filtering methods. As the end goal of the recommendation system is to recommend a particular number of movies to the users, we also devised a method to output the top 20 recommendations for a particular user.

### **5.2.2 Content-Based Filtering Recommendation**

We used cosine similarity and TF-IDF on movie genres for Content-Based Filtering and used Cosine similarity to measure the similarity between items and once again figured out top 20 recommendations for a particular user.

### **5.2.3 Hybrid Recommendation**

The input to our hybrid recommendation system is a user id and movie name. At the first stage, the system uses Content-Based Filtering method to figure out the most similar movies to that one. And in the next stage, it uses Collaborative Filtering to assign an estimated rating to those movies. And then we filter out the top ones and recommend them to the user.

## 5.3 Result Analysis

### 5.3.1 Quantitative Analysis

We first take a look at the comparison of RMSE and MAE errors between a Collaborative Filtering based and Hybrid system. Content-Based Filtering method has only a qualitative property and therefore we'll cover it in the next subsection.

Here we choose top-recommended movies by both systems for 10 users and calculate RMSE errors for each system for comparison. From the RMSE plot for 10 users in Figure 14, we see that the hybrid system has comparatively lower RMSE overall. The average RMSE plot in Figure 15 also shows the superiority of the hybrid system.

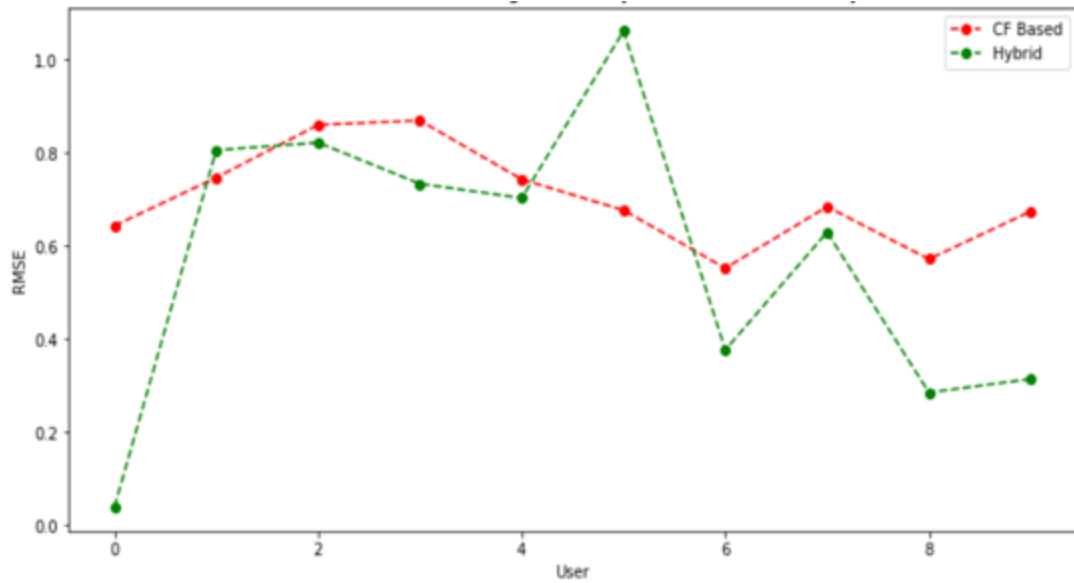
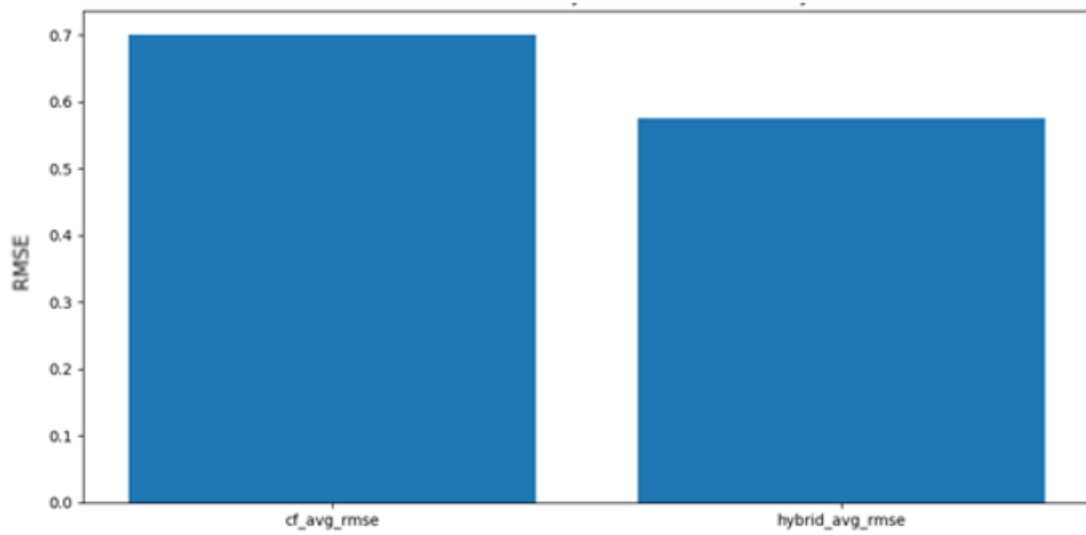
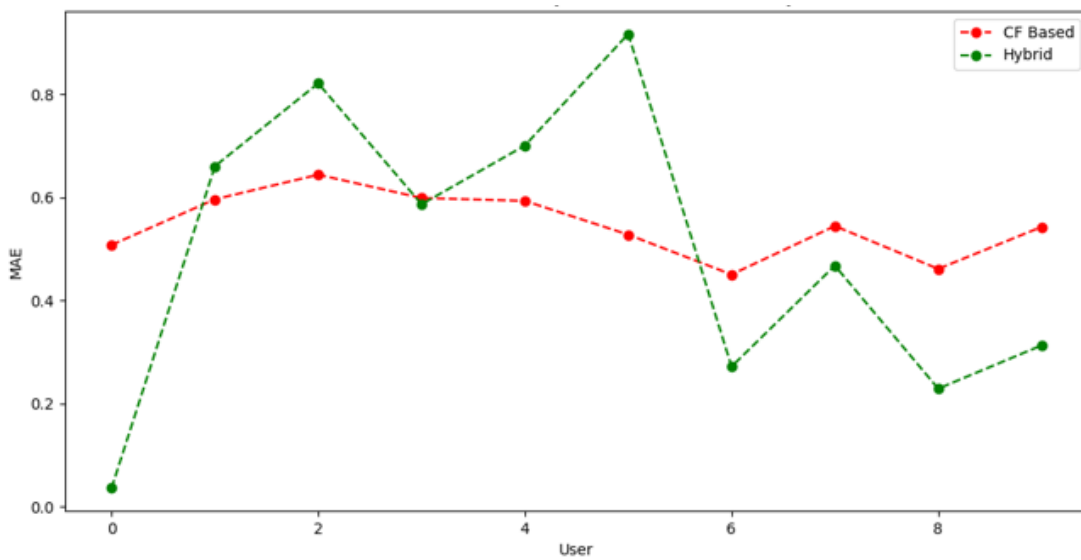


Figure 14. RMSE of Collaborative Filtering Based and Hybrid Recommendation System

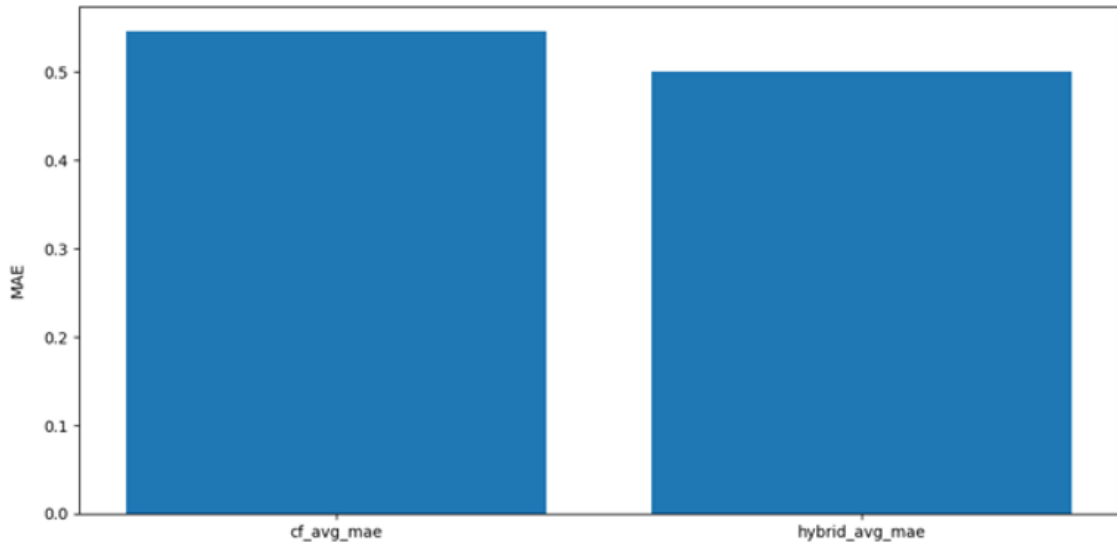


*Figure 15. Average RMSE of Collaborative Filtering Based and Hybrid Recommendation System*

We then do the same evaluation for MAE and from Figure 16 and Figure 17 see that the hybrid recommendation system has comparatively lower MAE, i.e., better accuracy.



*Figure 16. MAE of Collaborative Filtering Based and Hybrid Recommendation System*



*Figure 17. Average MAE of Collaborative Filtering Based and Hybrid Recommendation System*

Next, we consider 5 batches of users with each batch containing 5 users for whom we do the same test. We calculated the RMSE of these sets of users and the comparison shows Hybrid system performs comparatively better. The plot in Figure 18 shows the RMSE values. Figure 19 shows the average RMSE of Collaborative Filtering and Hybrid Recommendation System. We did the same for MAE with 5 sets of user groups that is shown in Figure 20. Hybrid system came on top here too. Figure 21 shows the average MAE of Collaborative Filtering and Hybrid Recommendation System.

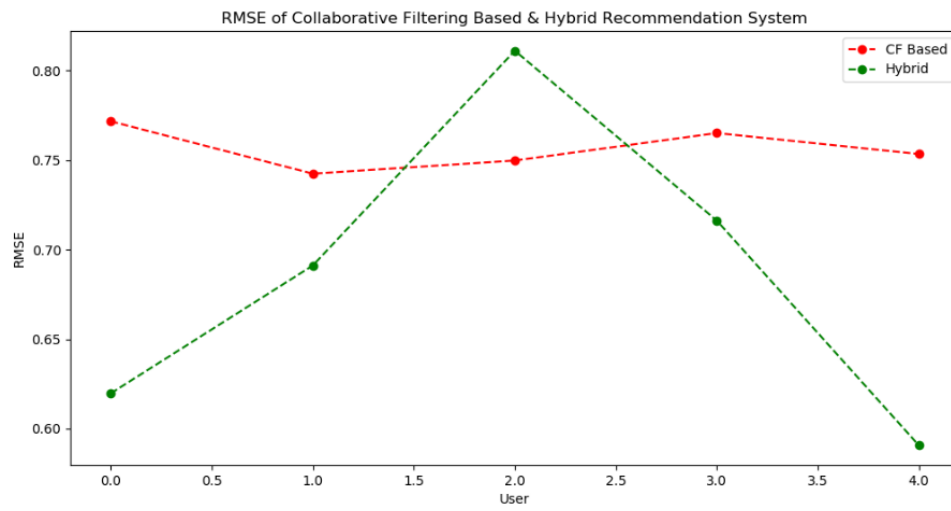


Figure 18. RMSE of Collaborative Filtering Based and Hybrid Recommendation System for 5 sets of users

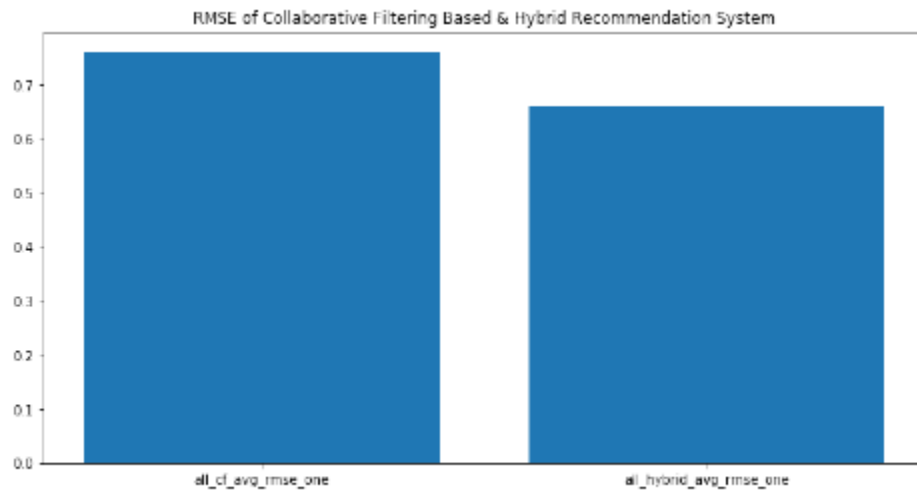
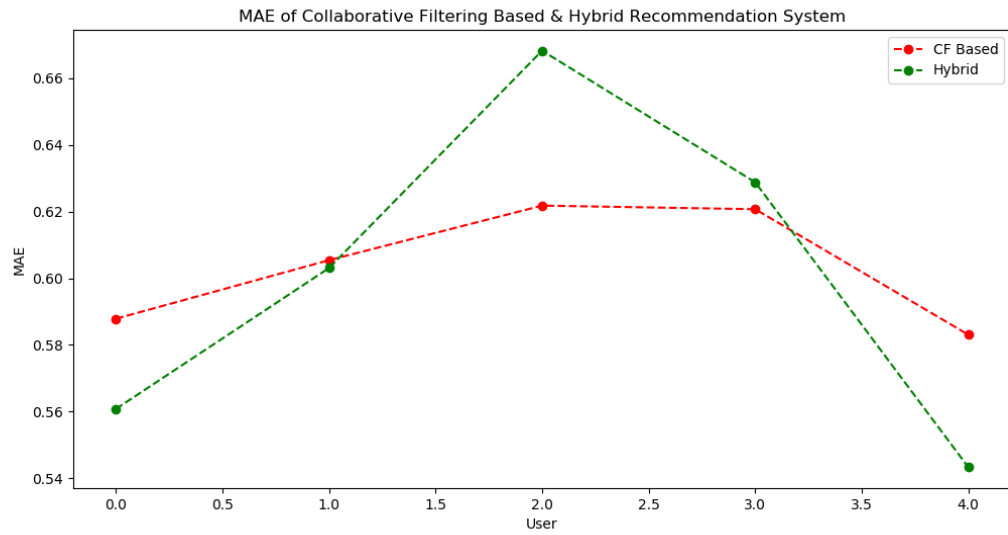
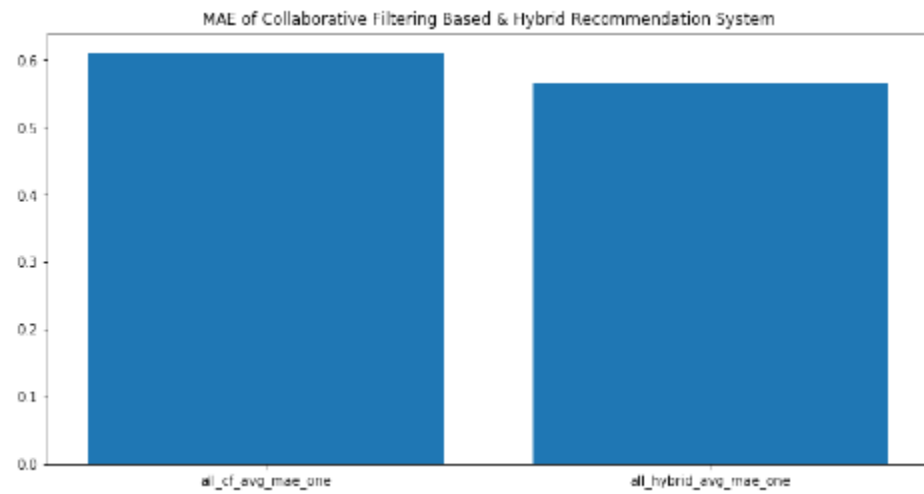


Figure 19. Average RMSE of Collaborative Filtering Based and Hybrid Recommendation System for 5 sets of users



*Figure 20. MAE of Collaborative Filtering Based and Hybrid Recommendation System for 5 sets of users*



*Figure 21. Average MAE of Collaborative Filtering Based and Hybrid Recommendation System for 5 sets of users*

### 5.3.2 Qualitative Analysis

From Table 2 we see that Collaborative Filtering can tell us the movies that a user is likely to rate higher. But it has no way of recommending similar movies to a particular one tailored for the specific user. As we can from the genre's column, the genres are all over the places. Here, we consider User 1 and recommend the top 20 movies he is likely to rate high.

*Table 2. Top 20 Recommended Movies for a Particular User by Collaborative Filtering Based Recommendation System*

movie_id	estimated_rating	title	actual_rating	genres
527	4.995416	[Schindler's List (1993)]	[5]	[Drama War]
318	4.958150	[Shawshank Redemption, The (1994)]	[]	[Drama]
1172	4.894460	[Cinema Paradiso (1988)]	[]	[Comedy Drama Romance]
2905	4.797475	[Sanjuro (1962)]	[]	[Action Adventure]
1269	4.727696	[Arsenic and Old Lace (1944)]	[]	[Comedy Mystery Thriller]
920	4.695179	[Gone with the Wind (1939)]	[]	[Drama Romance War]
2019	4.693533	[Seven Samurai (The Magnificent Seven) (Shichi...	[]	[Action Drama]
904	4.686336	[Rear Window (1954)]	[]	[Mystery Thriller]
922	4.654332	[Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)]	[]	[Film-Noir]
1234	4.651489	[Sting, The (1973)]	[]	[Comedy Crime]
1203	4.645870	[12 Angry Men (1957)]	[]	[Drama]
905	4.644219	[It Happened One Night (1934)]	[]	[Comedy]
858	4.639405	[Godfather, The (1972)]	[]	[Action Crime Drama]
1197	4.635691	[Princess Bride, The (1987)]	[3]	[Action Adventure Comedy Romance]
1233	4.632157	[Boat, The (Das Boot) (1981)]	[]	[Action Drama War]
356	4.629706	[Forrest Gump (1994)]	[]	[Comedy Romance War]
1198	4.626178	[Raiders of the Lost Ark (1981)]	[]	[Action Adventure]
953	4.613348	[It's a Wonderful Life (1946)]	[]	[Drama]
912	4.610552	[Casablanca (1942)]	[]	[Drama Romance War]
1242	4.608209	[Glory (1989)]	[]	[Action Drama War]

On the other hand, a Content-Based Filtering recommendation system has the option to find us the most similar movies to a given one as seen in Table 3, but it has no intuition into whether a user will like it or not. Here, we consider Movie Name: Toy Story



(1995) with Movie ID 1 and recommend the top 20 movies which are similar to the movie, Toy Story.

*Table 3. Top 20 Recommended Movies for a Particular Movie by Content-Based Filtering Recommendation System*

movie_index	similarity_score	title	movie_id	genres
1050	1.000000	Aladdin and the King of Thieves (1996)	1064	[[Animation', "Children's", 'Comedy]]
2072	1.000000	American Tail, An (1986)	2141	[[Animation', "Children's", 'Comedy]]
2073	1.000000	American Tail: Fievel Goes West, An (1991)	2142	[[Animation', "Children's", 'Comedy]]
2285	1.000000	Rugrats Movie, The (1998)	2354	[[Animation', "Children's", 'Comedy]]
2286	1.000000	Bug's Life, A (1998)	2355	[[Animation', "Children's", 'Comedy]]
3045	1.000000	Toy Story 2 (1999)	3114	[[Animation', "Children's", 'Comedy]]
3542	1.000000	Saludos Amigos (1943)	3611	[[Animation', "Children's", 'Comedy]]
3682	1.000000	Chicken Run (2000)	3751	[[Animation', "Children's", 'Comedy]]
3685	1.000000	Adventures of Rocky and Bullwinkle, The (2000)	3754	[[Animation', "Children's", 'Comedy]]
236	0.869805	Goofy Movie, A (1995)	239	[[Animation', "Children's", 'Comedy', 'Romanc...
12	0.826811	Balto (1995)	13	[[Animation', "Children's"]]
241	0.826811	Gumby: The Movie (1995)	244	[[Animation', "Children's"]]
310	0.826811	Swan Princess, The (1994)	313	[[Animation', "Children's"]]
592	0.826811	Pinocchio (1940)	596	[[Animation', "Children's"]]
612	0.826811	Aristocats, The (1970)	616	[[Animation', "Children's"]]
700	0.826811	Oliver & Company (1988)	709	[[Animation', "Children's"]]
876	0.826811	Land Before Time III: The Time of the Great Gi...	888	[[Animation', "Children's"]]
1010	0.826811	Winnie the Pooh and the Blustery Day (1968)	1023	[[Animation', "Children's"]]
1012	0.826811	Sword in the Stone, The (1963)	1025	[[Animation', "Children's"]]
1020	0.826811	Fox and the Hound, The (1981)	1033	[[Animation', "Children's"]]

A hybrid system gives us the best of both worlds. Table 4 shows that it can recommend similar movies to a particular one that the user is most likely to rate high. Here, we consider User ID 1, Movie Toy Story (1995) with Movie ID 1 and recommend top 20 movies which are similar to Toy Story and the movies which are likely to be rated high by the User 1.

*Table 4. Top 20 Recommended Movies for a Particular User and Movie by Hybrid Recommendation System*

movie_index	similarity_score	title	estimated_rating	actual_rating
2073	1.000000	American Tail: Fievel Goes West, An (1991)	4.107195	[]
1050	1.000000	Aladdin and the King of Thieves (1996)	4.077366	[]
2285	1.000000	Rugrats Movie, The (1998)	4.061477	[]
3685	1.000000	Adventures of Rocky and Bullwinkle, The (2000)	4.052081	[]
3542	1.000000	Saludos Amigos (1943)	3.732172	[]
3682	1.000000	Chicken Run (2000)	3.595624	[]
3045	1.000000	Toy Story 2 (1999)	3.319540	[]
2072	1.000000	American Tail, An (1986)	3.047335	[]
2286	1.000000	Bug's Life, A (1998)	2.749218	[]
236	0.869805	Goofy Movie, A (1995)	3.779034	[]
1949	0.826811	Bambi (1942)	4.424280	[]
2731	0.826811	Little Nemo: Adventures in Slumberland (1992)	4.384338	[]
2618	0.826811	Tarzan (1999)	4.315832	[]
2070	0.826811	Secret of NIMH, The (1982)	4.288048	[]
3730	0.826811	Pokémon the Movie 2000 (2000)	4.255108	[]
1012	0.826811	Sword in the Stone, The (1963)	4.199251	[]
2068	0.826811	Charlotte's Web (1973)	4.140761	[]
2692	0.826811	Iron Giant, The (1999)	4.049191	[4]
592	0.826811	Pinocchio (1940)	3.935920	[]
3546	0.826811	Dinosaur (2000)	3.922755	[]

Therefore, we can conclude that from both qualitative and quantitative perspective, a hybrid recommendation system performs comparatively better than standalone Collaborative Filtering or Content-Based Filtering recommendation system.

## **Chapter 6. Conclusions And Future Work**

In this study, we have explored different recommendation systems including Collaborative Filtering, Content-Based Filtering and Hybrid recommendation system using the well-known MovieLens dataset. We performed a qualitative and quantitative analysis using the dataset and doing so we compared these three recommendation systems. Conducting a mixed analysis of quantitative and qualitative manner comes from the need that Content-Based Filtering systems can't be easily quantified. Also, for a system like movie recommendation system the qualitative approach holds much importance. That is why we devised our own evaluation method along with traditional approach. We found out that in both the cases a hybrid recommendation system performs comparatively better.

There are opportunities for further analysis following the footsteps of this work. For example, we did not consider any demographic based information about the user in the recommendation system. However, considering this can add another layer of refinement in the hybrid recommendation system. Also, we considered only genre in Content-Based Filtering recommendation, but one can look into cast and crew and reviews of the movies for further similarity. Furthermore, a comparison among different Collaborative Filtering based methods and similarity measures may be interesting.

## References

- [1] V. M, and T. K, “History and overview of the recommender systems”, in Collaborative Filtering Using Data Mining and Analysis, V. Bhatnagar, Ed. Hershey: IGI Global, 2016, pp. 74-99.
- [2] M. Rouse, “What is collaborative filtering?”, WhatIs.com, para. 1, Aug. 11, 2017, [Online], Available: <https://whatis.techtarget.com/definition/collaborative-filtering>
- [3] C.S. Perone, “Machine Learning :: Cosine Similarity for Vector Space Models (Part III)”, Dec. 09, 2013, [Online], Available: <http://blog.christianperone.com/2013/09/machine-learning-cosine-similarity-for-vector-space-models-part-iii/>
- [4] B. Sarwar, G. Karypis, J. Konstan and R. John, “Item-based Collaborative Filtering recommendation algorithms”, 2001, pp. 285-295.
- [5] Z. Zhao, M. Shang, “User-based collaborative-filtering recommendation algorithms on Hadoop”, in Third International Conference on Knowledge Discovery and Data Mining, Phuket, Thailand, Jan. 09, 2010, IEEE, pp. 478-481.
- [6] R. V. Meteren, M. V. Someren, “Using Content-Based Filtering for recommendation”, 2000.
- [7] Y. Shih and D. Liu, “Hybrid recommendation approaches: Collaborative Filtering via valuable content information”, in Proceedings of the 38<sup>th</sup> Annual Hawaii International Conference on System Sciences, Big Island, USA, Jan. 06, 2005, IEEE, pp. 217b-217b.

- [8] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions", in *IEEE Transactions on Knowledge & Data Engineering*, Apr. 25, 2005, pp. 734-749.
- [9] C. Ziegler, S. M. McNee, J. A. Konstan and G. Lausen, "Improving recommendation lists through topic diversification", in *Proceedings of the 14th international conference on World Wide Web*, Chiba, Japan, May. 14, 2005, ACM, pp. 22-32.
- [10] S. Min and I. Han, "Detection of the customer time-variant pattern for improving recommender systems", *Expert systems with applications*, vol. 28, no. 2, pp. 189-199., Nov. 2004
- [11] H. Lieberman, "Letizia: An Agent That Assists Web Browsing", *IJCAI*, 1995, pp. 924-929.
- [12] B. Mobasher, "Recommender Systems," *Kunstliche Intelligenz*, Special Issue on Web Mining, BottcherIT Verlag, Bremen, Germany, 2007, pp. 41-43.
- [13] M. Göksedef and S. G. Ögüdücü, "Combination of Web page recommender systems", *Expert systems with applications*, vol. 37, no. 4, pp. 2911-2922., Apr 2010.
- [14] M. Y. H. Al-Shamri and K. K. Bharadwaj, "Fuzzy-genetic approach to recommender systems based on a novel hybrid user model", *Expert systems with applications*, vol. 35, no. 3, pp.1386-1399., Oct. 2008.
- [15] F. M. Harper and J.A. Konstan, "The MovieLens Datasets:History and Context", *ACM Transactions on Interactive Intelligent Systems (TiiS) - Regular Articles and*

Special issue on New Directions in Eye Gaze for Interactive Intelligent Systems (Part 1 of 2), Article 19, vol. 5, no. 4, pp. 12-14., Jan. 2016

## APPENDIX

The code can be accessed through the following GitHub link:

[www.github.com/AshwiniLokesh/Recommendation\\_System](https://www.github.com/AshwiniLokesh/Recommendation_System)

### Required Dependencies

```
import pandas as pd
from surprise import Dataset
from surprise import Reader
import matplotlib.pyplot as plt
import numpy as np
import math
from wordcloud import WordCloud
import string
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
import warnings
import random
```

Wordcloud shows us which genres are most prominent ones among the movies.

```
# wordcloud of genres to show prominent genres

x=movies_1m['Genres'].values.tolist()

all_genres=""
for line in x:
    words=str(line).split("|") #sometimes there were error: 'float' object has no attribute 'split'. That's why cast as a string
    for word in words:
        word=word.translate(str.maketrans('', '', string.punctuation))
        word=word.lower()
        all_genres=str(all_genres)+" "+str(word)

wordcloud = WordCloud(max_font_size=200, width=900, height=600,collocations=True).generate(all_genres)
plt.figure(figsize=(16,8))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.figtext(.5,.9,"Prominent Genres", fontsize=20, ha='center')
fig = plt.gcf()
fig.savefig('wordcloud.png', dpi=100)
plt.show()
```

### Collaborative Filtering Based Recommendation System

```
# collaborative filtering using surprise with SVD algorithm

from surprise import Reader, Dataset
from surprise import SVD
from surprise.model_selection import cross_validate

reader = Reader()
data = Dataset.load_from_df(ratings_1m[["UserID", "MovieID", "Rating"]], reader)

# use SDV algorithm
algo = SVD()

# run 5-fold cross-validation
x=cross_validate(algo, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

## Getting top 20 recommendation for a particular user

```
# get top 20 recommendation for a user
titles = movies_1m['Title']
indices = pd.Series(movies_1m.index, index=titles)

# function that get movie recommendations based on the cosine similarity score of movie genres
def cf_top_reco(userid):
    cf_df=pd.DataFrame()\

    movie_id_list=ratings_1m['MovieID'].unique().tolist()
    est=[]
    movie_name_list=[]
    actual_score_list=[]
    genre_list=[]
    for i in movie_id_list:
        pred = algo.predict(userid, i, verbose=False)
        est.append(pred[3])
        movie_name = movies_1m[movies_1m.MovieID==i]['Title'].values
        movie_name_list.append(movie_name)
        actual_score = ratings_1m[(ratings_1m.MovieID==i)&(ratings_1m.UserID==userid)]['Rating'].values
        actual_score_list.append(actual_score)
        genre = movies_1m[movies_1m.MovieID==i]['Genres'].values
        try:
            genre_list.append(genre)
        except:
            genre_list.append('Unavailable')
    cf_df['movie_id']=movie_id_list
    cf_df['estimated_rating']=est
    cf_df['title']=movie_name_list
    cf_df['actual_rating']=actual_score_list
    cf_df['genres']=genre_list
    cf_df=cf_df.sort_values(['estimated_rating'], ascending=[False])
    return cf_df

# the function takes userid as input & generates top 20 recommendation for that user. In Actual Rating field we don't
# have all values populated. This is because not all users rated all movies.
userid=1
cf_top_reco(userid).head(20)
```

## Content Based Recommendation System

```
# break up the big genre string into a string array
movies_1m['Genres'] = movies_1m['Genres'].str.split('|')
# convert genres to string value
movies_1m['Genres'] = movies_1m['Genres'].fillna('').astype('str')

from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd

tf = TfidfVectorizer(analyzer='word', ngram_range=(1, 2), min_df=0, stop_words='english')
tfidf_matrix = tf.fit_transform(movies_1m['Genres'])
df = pd.DataFrame(tfidf_matrix.toarray(), columns = tf.get_feature_names())
print(df)
#tfidf_matrix.shape
```



```

cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
cosine_sim[:5, :5]

array([[1.          , 0.14193614, 0.09010857, 0.1056164 , 0.23523322],
       [0.14193614, 1.          , 0.          , 0.          , 0.          ],
       [0.09010857, 0.          , 1.          , 0.1719888 , 0.38306058],
       [0.1056164 , 0.          , 0.1719888 , 1.          , 0.4489859 ],
       [0.23523322, 0.          , 0.38306058, 0.4489859 , 1.          ]])

# build an 1-dimensional array with movie titles
titles = movies_1m['Title']
movie_id = movies_1m['MovieID']
indices = pd.Series(movies_1m.index, index=movies_1m['Title'])

# function that get movie recommendations based on the cosine similarity score of movie genres
def genre_recommendations(title):
    idx = indices[title]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    # print(sim_scores)
    sim_scores = sim_scores[1:]
    movie_indices = [i[0] for i in sim_scores]
    cb_df = pd.DataFrame(sim_scores)
    cb_df['title'] = titles.iloc[movie_indices].values
    cb_df['movie_id'] = movie_id.iloc[movie_indices].values
    cb_df = cb_df.rename(columns={0: "movie_index", 1: "similarity_score"})
    genre_list = []
    for i in cb_df['movie_id'].values:
        genre = movies_1m[movies_1m.MovieID==i]['Genres'].values
        try:
            genre_list.append(genre)
        except:
            genre_list.append('Unavailable')
    cb_df['genres'] = genre_list
    return cb_df
title = 'Toy Story (1995)'
genre_recommendations(title).head(20)

```

## Hybrid Recommendation System

```

# an 1-dimensional array with movie titles
titles = movies_1m['Title']
indices = pd.Series(movies_1m.index, index=movies_1m['Title'])

# function that get movie recommendations based on the cosine similarity score of movie genres and estimate the rating
# given the particular user
def hybrid_recommendations(userId, title):
    idx = indices[title]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:100]
    movie_indices = [i[0] for i in sim_scores]
    hybrid_df = pd.DataFrame(sim_scores)
    hybrid_df['title'] = titles.iloc[movie_indices].values
    est = []
    actual_score_list = []
    hybrid_df = hybrid_df.rename(columns={0: "movie_index", 1: "similarity_score"})
    for i in movie_indices:
        pred = algo.predict(userId, i, verbose=False)
        est.append(pred[3])
        actual_score = ratings_1m[(ratings_1m.MovieID==i)&(ratings_1m.UserID==userId)]['Rating'].values
        actual_score_list.append(actual_score)
    hybrid_df['estimated_rating'] = est
    hybrid_df['actual_rating'] = actual_score_list
    hybrid_df = hybrid_df.sort_values(['similarity_score', 'estimated_rating'], ascending=[False, False])
    return hybrid_df

userid = 1
title = 'Toy Story (1995)'
hybrid_recommendations(userid, title).head(20)

```

## Comparison between Collaborative Filtering Based & Hybrid Recommendation System

Here we choose top recommended movies by both system for 10 users & calculate RMSE errors for each system for comparison.

```
warnings.filterwarnings("ignore")
```

```
rmse_val_sum=0
rmse_count=0
hybrid_rmse_val=[]

def rmse(predictions, targets):
    return np.sqrt(((predictions - targets) ** 2).mean())

# random.seed(0)
# random_users = random.sample(range(6040), 10)

# for j in random_users:
for j in range(20):
    x=hybrid_recommendations(j, 'Toy Story (1995)')
    rating_sum=0
    count=0
    y_actual=[]
    y_pred=[]
    for i in range(len(x)):
        if x.iloc[i]['actual_rating'].size>0:
            y_actual.append((int(x.iloc[i]['actual_rating'])))
            y_pred.append((x.iloc[i]['estimated_rating']))

    rmse_val = rmse(np.array(y_pred), np.array(y_actual))

    y = float(rmse_val)
    if math.isnan(y)==False:
        hybrid_rmse_val.append(rmse_val)
        rmse_val_sum=rmse_val_sum+rmse_val
        rmse_count=rmse_count+1
    if rmse_count==10:
        break

hybrid_avg_rmse=rmse_val_sum/rmse_count
print("Average RMSE for hybrid system:",hybrid_avg_rmse)
```

Average RMSE for hybrid system: 0.6103837913822747

```
rmse_val_sum=0
rmse_count=0
cf_rmse_val=[]

def rmse(predictions, targets):
    return np.sqrt(((predictions - targets) ** 2).mean())

# random.seed(0)
# random_users = random.sample(range(6040), 10)

# for j in random_users:
for j in range(20):
    x=cf_top_reco(j)
    rating_sum=0
    count=0
    y_actual=[]
    y_pred=[]
    for i in range(len(x)):
        if x.iloc[i]['actual_rating'].size>0:
            y_actual.append((int(x.iloc[i]['actual_rating'])))
            y_pred.append((x.iloc[i]['estimated_rating']))

    rmse_val = rmse(np.array(y_pred), np.array(y_actual))

    y = float(rmse_val)
    if math.isnan(y)==False:
        cf_rmse_val.append(rmse_val)
        rmse_val_sum=rmse_val_sum+rmse_val
        rmse_count=rmse_count+1
    if rmse_count==10:
        break

cf_avg_rmse=rmse_val_sum/rmse_count
print("Average RMSE for collaborative filtering based system:",cf_avg_rmse)
```

Average RMSE for collaborative filtering based system: 0.7195832091444867

Here we choose top recommended movies by both system for 10 users & calculate MAE errors for each system for comparison.

```
mae_val_sum=0
mae_count=0
hybrid_mae_val=[]

def mae(predictions, targets):
    return (abs(predictions - targets).mean())

# random.seed(100)
# random_users = random.sample(range(6040), 10)

# for j in random_users:
for j in range(20):
    x=hybrid_recommendations(j, 'Toy Story (1995)')
    rating_sum=0
    count=0
    y_actual=[]
    y_pred=[]
    for i in range(len(x)):
        if x.iloc[i]['actual_rating'].size>0:
            y_actual.append((int(x.iloc[i]['actual_rating'])))
            y_pred.append((x.iloc[i]['estimated_rating']))

    mae_val = mae(np.array(y_pred), np.array(y_actual))

    y = float(mae_val)
    if math.isnan(y)==False:
        hybrid_mae_val.append(mae_val)
        mae_val_sum=mae_val_sum+mae_val
        mae_count=mae_count+1
    if mae_count==10:
        break

hybrid_avg_mae=mae_val_sum/mae_count
print("Average MAE for hybrid system:",hybrid_avg_mae)
```

Average MAE for hybrid system: 0.5368483608080034

```
mae_val_sum=0
mae_count=0
cf_mae_val=[]

def mae(predictions, targets):
    return (abs(predictions - targets).mean())

# random.seed(0)
# random_users = random.sample(range(6040), 10)

# for j in random_users:
for j in range(20):
    x=cf_top_reco(j)
    rating_sum=0
    count=0
    y_actual=[]
    y_pred=[]
    for i in range(len(x)):
        if x.iloc[i]['actual_rating'].size>0:
            y_actual.append((int(x.iloc[i]['actual_rating'])))
            y_pred.append((x.iloc[i]['estimated_rating']))

    mae_val = mae(np.array(y_pred), np.array(y_actual))

    y = float(mae_val)
    if math.isnan(y)==False:
        cf_mae_val.append(mae_val)
        mae_val_sum=mae_val_sum+mae_val
        mae_count=mae_count+1
    if mae_count==10:
        break

cf_avg_mae=mae_val_sum/mae_count
print("Average MAE for collaborative filtering based system:",cf_avg_mae)
```

Average MAE for collaborative filtering based system: 0.5654935584286477