ABSTRACT

On the Internet, where the number of choices is overwhelming, there is need to filter, prioritize and efficiently deliver relevant information in order to alleviate the problem of information overload, which has created a potential problem to many Internet users. Recommender systems solve this problem by searching through large volume of dynamically generated information to provide users with personalized content and services. This report shows implementation of two types of recommendation system, i.e. content based and collaborative filtering based in order to serve as a compass for practice in the field of recommendation systems.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

S. No.	Abbreviations	Description
1.	Pandas	Python Data Analysis Library
2.	NumPy	Numeric Python used for multi-dimensional arrays
3.	SVD	Singular Value Decomposition
4.	RBM	Restricted Boltzmann Machines
5.	RMSE	Root mean squared error
6.	CineMatch	Cinema Matching Algorithm
7.	CF	Collaborative Filtering

LIST OF FIGURES

CHAPTER : 1

INTRODUCTION

Recommender Systems are an important class of machine learning algorithms which captures the pattern of people's behaviour and use it to predict what else they might like. For eg. – watching a movie and liking it most probably gives a chance that a movie of the same genre can be watched in the future. There are a wide variety of applications for recommendation systems. These have become increasingly popular over the last few years and are now utilized in most online platforms that we use. The content of such platforms varies from movies, music, books and videos, to friends and stories on social media platforms, to products on e-commerce websites, to people on professional and dating websites, to search results returned on Google.

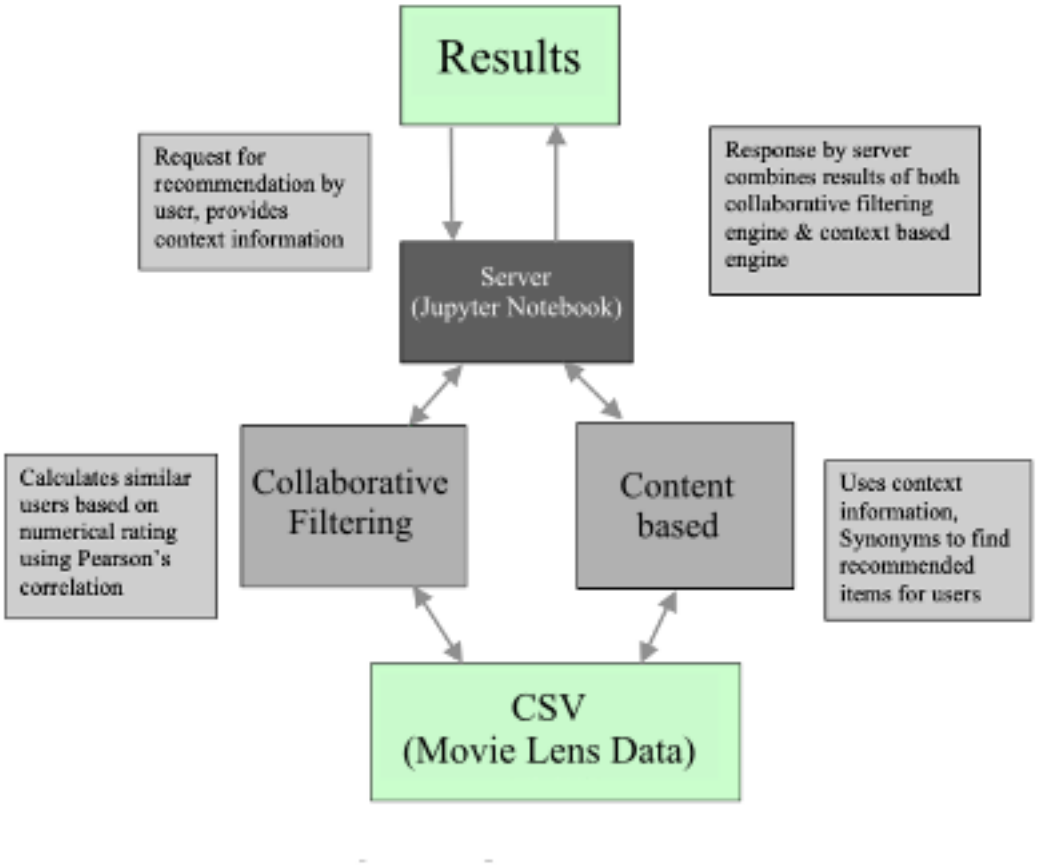
1.1PURPOSE

The goal of this project is to make a movie recommendation engines by making use of user based collaborative filtering technique and combining content based results along with it. The system makes use of numerical ratings of similar items between the active user and other users of the system to assess the similarity between users' profiles to predict recommendations of unseen items to active user. The system makes use of Pearson's correlation to evaluate the similarity between users. The results show that the system rests in its assumption that active users will always react constructively to items rated highly by similar users, shortage of ratings of some items, adapt quickly to change of user's interest, and identification of potential features of an item which could be of interest to the user.

This project will focus on making use of context based approach in addition to Collaborative Filtering approach to recommend quality content to its user

1.2OVERVIEW

Figure 2 : System Architecture



Description:

1 Previous Ratings provided by random Users are stored in a csv provided by Movielens.

2 Ratings for some favourite movies are input in the Jupyter Notebook.

3 There are two options, I.e. collaborative filtering or content based recommender systems.

4 First, the movies are recommended through content based recommender system based on genres.

5 Second, the movies are recommended through collaborative filtering-based recommender system using Pearson correlation coefficient.

6 The ratings added in the jupyter notebook can be changed as per use and results would be provided accordingly.

1.3ABOUT THE TECHNOLOGY

1.3.1 Python

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

1.3.2 Jupyter Notebook

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

1.3.3 Pandas

Pandas is a python package used for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. Common Features: DataFrame object for data manipulation with integrated indexing, Reshaping and pivoting of data sets, Data set merging and joining, Provides data filtration, etc.

1.3.4 numpy

Numpy is a python package adding support for large, multi-dimensional arrays and matrices, along

with a large collection of high-level mathematical functions to operate on these arrays. Common Features: working with n-dimensional array data structure, comprehensive mathematical functions, random number generators, linear algebra routines, etc.

CHAPTER : 2

LITERATURE SURVEY

2.1 MOVIEREC

MOVREC is a movie recommendation system presented by D.K. Yadav et al. based on collaborative filtering approach. Collaborative filtering makes use of information provided by user. That information is analyzed and a movie is recommended to the users which are arranged with the movie with highest rating first.

Luis M Capos et al has analyzed two traditional recommender systems i.e. content based filtering and collaborative filtering. As both of them have their own drawbacks he proposed a new system which is a combination of Bayesian network and collaborative filtering.

A hybrid system has been presented by Harpreet Kaur et al. The system uses a mix of content as well as collaborative filtering algorithm. The context of the movies is also considered while recommending. The user - user relationship as well as user - item relationship plays a role in the recommendation.

The user specific information or item specific information is clubbed to form a cluster by Utkarsh Gupta et al. using chameleon. This is an efficient technique based on Hierarchical clustering for recommender system. To predict the rating of an item voting system is used. The proposed system has lower error and has better clustering of similar items.

Urszula Kużelewska et al. proposed clustering as a way to deal with recommender systems. Two methods of computing cluster representatives were presented and evaluated. Centroid-based solution and memory-based collaborative filtering methods were used as a basis for comparing effectiveness of the proposed two methods. The result was a significant increase in the accuracy of the generated recommendations when compared to just centroid-based method.

Costin-Gabriel Chiru et al. proposed Movie Recommender, a system which uses the information known about the user to provide movie recommendations. This system attempts to solve the problem of unique recommendations which results from ignoring the data specific to the user. The psychological profile of the user, their watching history and the data involving movie scores from other websites is collected. They are based on aggregate similarity calculation. The system is a hybrid model which uses both content based filtering and collaborative filtering.
To predict the difficulty level of each case for each trainee Hongli LIn et al. proposed a method called contentboosted collaborative filtering (CBCF).The algorithm is divided into two stages, First being the content-based filtering that improves the existing trainee case ratings data and the second being collaborative filtering that provides the final predictions. The CBCF algorithm involves the advantages of both CBF and CF, while at the same time, overcoming both their disadvantages.

2.2 CASE STUDY

2.2.1 NETFLIX RECOMMENDATIONS : BEYOND THE 5 STARS

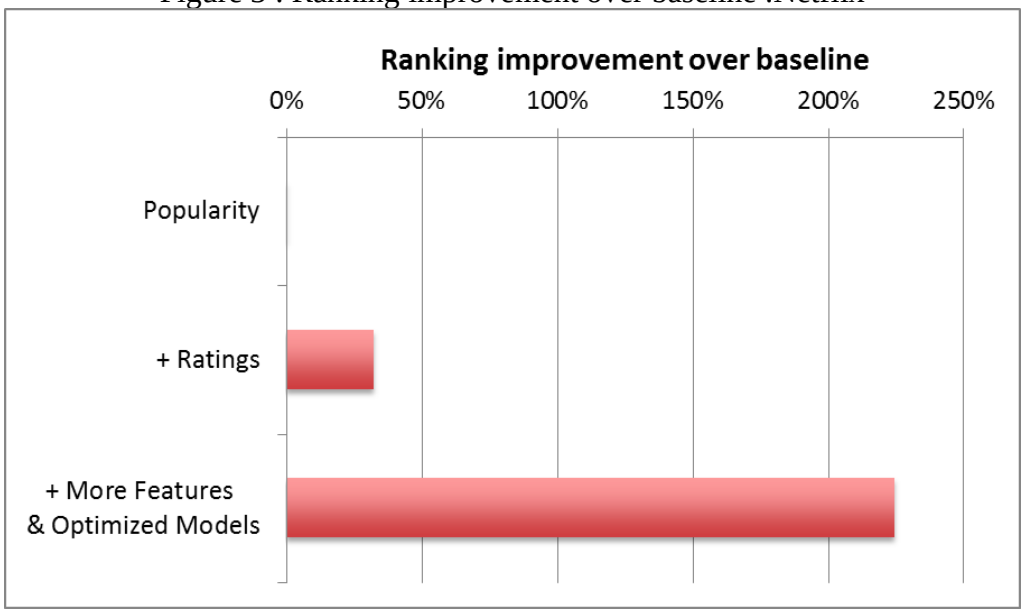
The first algorithm behind these recommendation was called CineMatch. The CineMatch algorithm had a long run and proved fairly successful at predicting movies that subscribers would like. According to Netflix, these predictions were accurate within half a star 75 percent of the time, and half of Netflix users who rented CineMatch-recommended movies gave them a five-star rating. The algorithm took into account:

- The films themselves, which are arranged as groups of common movies
- The customers' ratings, rented movies and current queue
- The combined ratings of all Netflix users

In 2006, Netflix announced the Netflix Prize, a machine learning and data mining competition for movie rating prediction. For the result, Netflix looked at the two underlying algorithms with the best

performance in the ensemble: Matrix Factorization (which the community generally called SVD, Singular Value Decomposition) and Restricted Boltzmann Machines (RBM). SVD by itself provided a 0.8914 root mean squared error, while RBM alone provided a competitive but slightly worse 0.8990 RMSE. A linear blend of these two reduced the error to 0.88. In order to put these algorithms to use, Netflix had to work to overcome some limitations, such as h aving been built to handle 100 million ratings instead of the more than 5 billion we have, and not b eing built to adapt as members added more ratings. But after Netflix resolved those obstacles, they put in place the two algorithms where they are now being used as part of our recommendation engi ne. Netflix launched an instant streaming service in 2007, one year after the Netflix Prize began. apart from Netflix's popularity and rating prediction, we have tried many other features at Netflix. Some have shown no positive effect while others have improved our ranking accuracy tremendously. The graph below shows the ranking improvement we have obtained by adding different features and optimizing the machine learning algorithm.

Figure 3 : Ranking improvement over baseline :Netflix



These of some of the list of methods Netflix uses to work in machine learning for personalization:

- Linear regression
- Logistic regression
- Elastic nets
- Singular Value Decomposition
- Restricted Boltzmann Machines
- Markov Chains
- Latent Dirichlet Allocation
- Association Rules
- Gradient Boosted Decision Trees
- Random Forests
- Clustering techniques from the simple k-means to novel graphical approaches such as Affinity Propagation
- Matrix factorization

CHAPTER : 3

REQUIREMENT ANALYSIS

For the successful development of any project, it is very necessary to study about all the requirements of the project. This requirement may be related to resource, hardware, software, functionalities, etc. This requirement analysis is as important as the implementation of the project. The requirement analysis basically emphasizes on the listing of the necessary requirements required for the development of the project.

3.1 HARDWARE SPECIFICATIONS

PC	:	Pentium IV
Processor	:	1 GHz CPU
RAM	:	512 MB
Hard disk	:	5 GB

3.2 SOFTWARE SPECIFICATIONS

Operating System	:	Windows 2000/XP/8.1/10
Software	:	Jupyter Notebook
Languages	:	Python, Libraries used : Pandas, Numpy, MatplotLib
Browser	:	Opera, Mozilla Firefox, Google Chrome

CHAPTER : 4

METHODOLOGIES USED

4.1 Matrix Factorization

Matrix factorization was popularly used during the Netflix recommendation challenge, especially singular value decomposition and a more practical version for recommender systems.

Singular value decomposition (SVD) decomposes the preference matrix as

$$P_{m\times n} = U_{m\times m} \sum_{m\times n} V_{n\times n}$$

U and V are unitary matrices. For 4 users and 5 items, it looks like

$$P_{4\times 5} = \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ u_{21} & u_{22} & u_{23} & u_{24} \\ u_{31} & u_{32} & u_{33} & u_{34} \\ u_{41} & u_{42} & u_{43} & u_{44} \end{bmatrix} \bullet \begin{bmatrix} \sigma_1 & 0 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 & 0 \\ 0 & 0 & \sigma_3 & 0 & 0 \\ 0 & 0 & 0 & \sigma_4 & 0 \end{bmatrix} \bullet \begin{bmatrix} v_{11} & v_{12} & v_{13} & v_{14} & v_{15} \\ v_{21} & v_{22} & v_{23} & v_{24} & v_{25} \\ v_{31} & v_{32} & v_{33} & v_{34} & v_{35} \\ v_{41} & v_{42} & v_{43} & v_{44} & v_{45} \\ v_{51} & v_{52} & v_{53} & v_{54} & v_{55} \end{bmatrix}$$

where sigma_1 > sigma_2 > sigma_3 > sigma_4.

The preference of the first user for the first item can be written as

$$p_{11} = \sigma_1 u_{11} v_{11} + \sigma_2 u_{12} v_{21} + \sigma_3 u_{13} v_{31} + \sigma_4 u_{14} v_{41}$$

This can be presented as vectors

$$p_{11} = \overrightarrow{\sigma} \circ \overrightarrow{u_1} \bullet \overrightarrow{v_1}$$

An entrywise product is applied between the sigma vector and the first user vector, and then a dot product with the first item vector. It can be seen u and v have the same length, i.e., they are in the same latent feature space. The sigma vector represents the importance of each feature.

Now let’s select the top two features based on the sigma’s

$$p_{11} \approx \sigma_1 u_{11} v_{11} + \sigma_2 u_{12} v_{21}$$

which can be presented as the item and user vectors each has a length of two.

4.2 Pearson Correlation Coefficient

The proposed system makes use of Pearson's correlation to implement User based collaborative filtering, and context, Synonym Finder to implement Context based filtering techniques to generate recommendations for the active user.

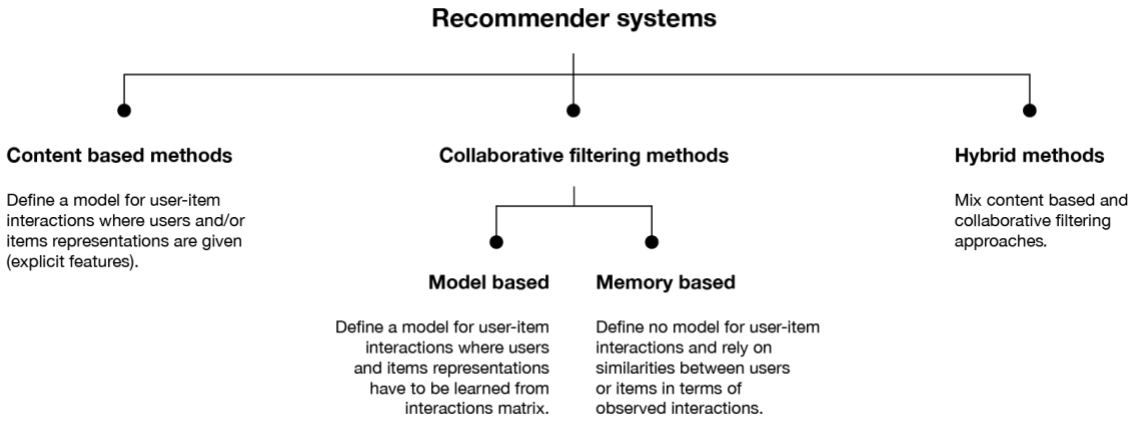
Pearson Correlation Coefficient - It is the test statistics that measures the statistical relationship, or association, between two continuous variables. It is known as the best method of measuring the association between variables of interest because it is based on the method of covariance. It gives information about the magnitude of the association, or correlation, as well as the direction of the relationship.

CHAPTER : 5

SYSTEM DESIGN

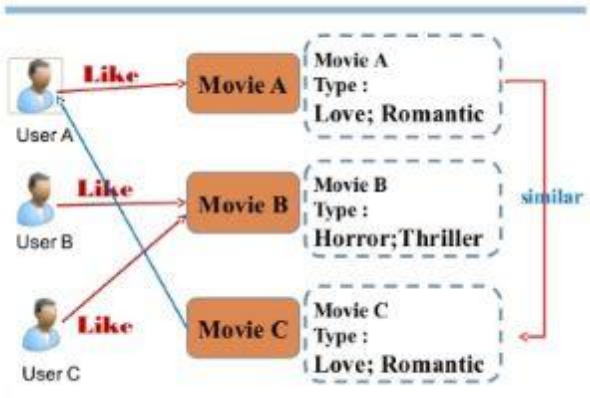
5.1 WORKFLOW

Figure 4 : Types of Recommender Systems



5.1.1 Content-based Filtering Systems: In content-based filtering, items are recommended based on comparisons between item profile and user profile. A user profile is content that is found to be relevant to the user in form of keywords(or features). A user profile might be seen as a set of assigned keywords (terms, features) collected by algorithm from items found relevant (or interesting) by the user. A set of keywords (or features) of an item is the Item profile. For example, consider a scenario in which a person goes to buy his favorite cake ‘X’ to a pastry. Unfortunately, cake ‘X’ has been sold out and as a result of this the shopkeeper recommends the person to buy cake ‘Y’ which is made up of ingredients similar to cake ‘X’. This is an instance of content-based filtering.

Figure 5 : Content Based Filtering



It goes without saying that the quality of our recommender would be increased with the usage of better metadata. That is exactly what we are going to do in this section. We are going to build a recommender based on the following metadata: the 3 top actors, the director, related genres and the movie plot keywords.

From the cast, crew and keywords features, we need to extract the three most important actors, the director and the keywords associated with that movie. Right now, our data is present in the form of "stringified" lists , we need to convert it into a safe and usable structure.

Advantages of content-based filtering are:

- They capable of recommending unrated items.
- We can easily explain the working of recommender system by listing the Content features of an item.
- Content-based recommender systems use need only the rating of the concerned user,and not any other user of the system.

Disadvantages of content-based filtering are:

- It does not work for a new user who has not rated any item yet as enough ratings are required contentbased recommender evaluates the user preferences and provides accurate

recommendations.

- No recommendation of serendipitous items.
- Limited Content Analysis- The recommend does not work if the system fails to distinguish the items hat a user likes from the items that he does not like.

5.1.2 Collaborative filtering based systems: Our content based engine suffers from some severe limitations. It is only capable of suggesting movies which are close to a certain movie. That is, it is not capable of capturing tastes and providing recommendations across genres.

Also, the engine that we built is not really personal in that it doesn't capture the personal tastes and biases of a user. Anyone querying our engine for recommendations based on a movie will receive the same recommendations for that movie, regardless of who she/he is.

Therefore, in this section, we will use a technique called Collaborative Filtering to make recommendations to Movie Watchers. It is basically of two types:-

5.1.2.1 User based filtering- These systems recommend products to a user that similar users have liked. For measuring the similarity between two users we can either use pearson correlation or cosine similarity. This filtering technique can be illustrated with an example. In the following matrix's, each row represents a user, while the columns correspond to different movies except the last one which records the similarity between that user and the target user. Each cell represents the rating that the user gives to that movie. Assume user E is the target.

Figure 6 : User Based Filtering

	The Avengers	Sherlock	Transformers	Matrix	Titanic	Me Before You	Similarity(i, E)
A	2		2	4	5		NA
B	5		4			1	
C			5		2		
D		1		5		4	
E			4			2	1
F	4	5		1			NA

Since user A and F do not share any movie ratings in common with user E, their similarities with user E are not defined in Pearson Correlation. Therefore, we only need to consider user B, C, and D. Based on Pearson Correlation, we can compute the following similarity.

	The Avengers	Sherlock	Transformers	Matrix	Titanic	Me Before You	Similarity(i, E)
A	2		2	4	5		NA
B	5		4			1	0.87
C			5		2		1
D		1		5		4	-1
E			4			2	1
F	4	5		1			NA

5.1.2.2 Item Based Collaborative Filtering- Instead of measuring the similarity between users, the item-based CF recommends items based on their similarity with the items that the target user rated. Likewise, the similarity can be computed with Pearson Correlation or Cosine Similarity. The major difference is that, with item-based collaborative filtering, we fill in the blank vertically, as oppose to the horizontal manner that user-based CF does. The following table shows how to do so for the movie Me Before.

Figure 7 : Item Based Filtering

	The Avengers	Sherlock	Transformers	Matrix	Titanic	Me Before You
A	2		2	4	5	2.94*
B	5		4			1
C			5		2	2.48*
D		1		5		4
E			4			2
F	4	5		1		1.12*
Similarity	-1	-1	0.86	1	1	

It successfully avoids the problem posed by dynamic user preference as item-based CF is more static. However, several problems remain for this method. First, the main issue is scalability. The computation grows with both the customer and the product. The worst case complexity is O(mn) with m users and n items. In addition, sparsity is another concern. Take a look at the above table again. Although there is only one user that rated both Matrix and Titanic rated, the similarity between them is 1. In extreme cases, we can have millions of users and the similarity between two fairly different movies could be very high simply because they have similar rank for the only user who ranked them both.

5.2 OUTPUT

Output is based on the dataset provided by MovieLens in which list of movies with their ratings is provided.

Firstly, user input is taken in the form of an array as presented below : -

Figure 8 : User Input of Movies

```
In [10]:
userInput = [
            {'title':'Fast & Furious (Fast and the Furious 4, The)', 'rating':4
},
            {'title':'Jumanji: Welcome to the Jungle', 'rating':4.5},
            {'title':'Secret Superstar', 'rating':3},
            {'title':"Captain Marvel", 'rating':3.5},
            {'title':'Intern, The', 'rating':5}
         ]
inputMovies = pd.DataFrame(userInput)
inputMovies
```

The steps continue as pre-processing the data, applying the model and getting the ouput is performed in both the types of recommendation system. Output is provided in the screenshots available in the system development chapter. This section just discussed about how the output is inferred from the input and what the relation between the two signifies in this scenario.

CHAPTER : 6

SYSTEM DEVELOPMENT & IMPLEMENTATION

6.1 SCREENSHOTS

Acquiring the Data from MovieLens

In [1]:

```
!wget -O ml-25m.zip http://files.grouplens.org/datasets/movielens/ml-25m.zip
!unzip -o -j ml-25m.zip
```

```
--2020-05-22 15:20:57--  http://files.grouplens.org/datasets/moviele
ns/ml-25m.zip
Resolving files.grouplens.org (files.grouplens.org)... 128.101.65.15
2
Connecting to files.grouplens.org (files.grouplens.org)|128.101.65.1
52|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 261978986 (250M) [application/zip]
Saving to: 'ml-25m.zip'

100%[======================================>] 261,978,986 78.8MB/s
in 3.2s

2020-05-22 15:21:01 (78.8 MB/s) - 'ml-25m.zip' saved [261978986/2619
78986]

Archive:  ml-25m.zip
  inflating: tags.csv
  inflating: links.csv
  inflating: README.txt
  inflating: ratings.csv
  inflating: genome-tags.csv
  inflating: genome-scores.csv
  inflating: movies.csv
```

PreProcessing the Data

In [2]:

```
#Importing Data manipulation library
import pandas as pd
#Importing Array manipulation library
import numpy as np
#Importing Graphical plotting library
import matplotlib.pyplot as plt
```

In [3]:

```
#Storing the movie information into a pandas variable
movies_data = pd.read_csv('movies.csv')
#Storing the user information into a pandas variable
ratings_data = pd.read_csv('ratings.csv')
```

In [4]:

```
#Head is a function that gets the first N rows of a dataframe. N's default is 5.
print(movies_data.head(10))
print(ratings_data.head(10))
```

```
   movieId                               title  \
0        1                    Toy Story (1995)
1        2                      Jumanji (1995)
2        3             Grumpier Old Men (1995)
3        4            Waiting to Exhale (1995)
4        5  Father of the Bride Part II (1995)
5        6                         Heat (1995)
6        7                      Sabrina (1995)
7        8                 Tom and Huck (1995)
8        9                 Sudden Death (1995)
9       10                    GoldenEye (1995)

                                        genres
0  Adventure|Animation|Children|Comedy|Fantasy
1                   Adventure|Children|Fantasy
2                               Comedy|Romance
3                         Comedy|Drama|Romance
4                                       Comedy
5                        Action|Crime|Thriller
6                               Comedy|Romance
7                          Adventure|Children
8                                       Action
9                   Action|Adventure|Thriller
   userId  movieId  rating   timestamp
0       1      296     5.0  1147880044
1       1      306     3.5  1147868817
2       1      307     5.0  1147868828
3       1      665     5.0  1147878820
4       1      899     3.5  1147868510
5       1     1088     4.0  1147868495
6       1     1175     3.5  1147868826
7       1     1217     3.5  1147878326
8       1     1237     5.0  1147868839
9       1     1250     4.0  1147868414
```

In [5]:

```
#Using regular expressions to find a year stored between parentheses
#We specify the parantheses so we don't conflict with movies that have years in
 their titles
movies_data['year'] = movies_data.title.str.extract('(\(\d\d\d\d\))',expand=Fals
e)
#Removing the parentheses
movies_data['year'] = movies_data.year.str.extract('(\d\d\d\d)',expand=False)
#Removing the years from the 'title' column
movies_data['title'] = movies_data.title.str.replace('(\(\d\d\d\d\))', '')
#Applying the strip function to get rid of any ending whitespace characters that
may have appeared
movies_data['title'] = movies_data['title'].apply(lambda x: x.strip())
movies_data.head(10)
```

Out[5]:

	movieId	title	genres	year
0	1	Toy Story	Adventure|Animation|Children|Comedy|Fantasy	1995
1	2	Jumanji	Adventure|Children|Fantasy	1995
2	3	Grumpier Old Men	Comedy|Romance	1995
3	4	Waiting to Exhale	Comedy|Drama|Romance	1995
4	5	Father of the Bride Part II	Comedy	1995
5	6	Heat	Action|Crime|Thriller	1995
6	7	Sabrina	Comedy|Romance	1995
7	8	Tom and Huck	Adventure|Children	1995
8	9	Sudden Death	Action	1995
9	10	GoldenEye	Action|Adventure|Thriller	1995

In [6]:

```
#Every genre is separated by a | so we simply have to call the split function on
|
movies_data['genres'] = movies_data.genres.str.split('|')
movies_data.head(10)
```

Out[6]:

	movieId	title	genres	year
0	1	Toy Story	[Adventure, Animation, Children, Comedy, Fantasy]	1995
1	2	Jumanji	[Adventure, Children, Fantasy]	1995
2	3	Grumpier Old Men	[Comedy, Romance]	1995
3	4	Waiting to Exhale	[Comedy, Drama, Romance]	1995
4	5	Father of the Bride Part II	[Comedy]	1995
5	6	Heat	[Action, Crime, Thriller]	1995
6	7	Sabrina	[Comedy, Romance]	1995
7	8	Tom and Huck	[Adventure, Children]	1995
8	9	Sudden Death	[Action]	1995
9	10	GoldenEye	[Action, Adventure, Thriller]	1995

In [7]:

```
#Copying the movie dataframe into a new one since we won't need to use the genre
information in our first case.
moviesWithGenres_data = movies_data.copy()

#For every row in the dataframe, iterate through the list of genres and place a
 1 into the corresponding column
for index, row in movies_data.iterrows():
    for genre in row['genres']:
        moviesWithGenres_data.at[index, genre] = 1
#Filling in the NaN values with 0 to show that a movie doesn't have that colum
n's genre
moviesWithGenres_data = moviesWithGenres_data.fillna(0)
moviesWithGenres_data.head(10)
```

Out[7]:

	movieId	title	genres	year	Adventure	Animation	Children	Comedy	Fantasy
0	1	Toy Story	[Adventure, Animation, Children, Comedy, Fantasy]	1995	1.0	1.0	1.0	1.0	1.0
1	2	Jumanji	[Adventure, Children, Fantasy]	1995	1.0	0.0	1.0	0.0	1.0
2	3	Grumpier Old Men	[Comedy, Romance]	1995	0.0	0.0	0.0	1.0	0.0
3	4	Waiting to Exhale	[Comedy, Drama, Romance]	1995	0.0	0.0	0.0	1.0	0.0
4	5	Father of the Bride Part II	[Comedy]	1995	0.0	0.0	0.0	1.0	0.0
5	6	Heat	[Action, Crime, Thriller]	1995	0.0	0.0	0.0	0.0	0.0
6	7	Sabrina	[Comedy, Romance]	1995	0.0	0.0	0.0	1.0	0.0
7	8	Tom and Huck	[Adventure, Children]	1995	1.0	0.0	1.0	0.0	0.0
8	9	Sudden Death	[Action]	1995	0.0	0.0	0.0	0.0	0.0
9	10	GoldenEye	[Action, Adventure, Thriller]	1995	1.0	0.0	0.0	0.0	0.0

10 rows × 24 columns

In [8]:

```
#Drop removes a specified row or column from a dataframe
ratings_data = ratings_data.drop('timestamp', 1)
ratings_data.head()
```

Out[8]:

	userId	movieId	rating
0	1	296	5.0
1	1	306	3.5
2	1	307	5.0
3	1	665	5.0
4	1	899	3.5

In [9]:

```
#Merge is a function that combines all the rows of the dataset with the specifie
d dataframe.
data = ratings_data.merge(movies_data,on='movieId', how='left')
data.head(10)
```

Out[9]:

	userId	movieId	rating	title	genres	year
0	1	296	5.0	Pulp Fiction	[Comedy, Crime, Drama, Thriller]	1994
1	1	306	3.5	Three Colors: Red (Trois couleurs: Rouge)	[Drama]	1994
2	1	307	5.0	Three Colors: Blue (Trois couleurs: Bleu)	[Drama]	1993
3	1	665	5.0	Underground	[Comedy, Drama, War]	1995
4	1	899	3.5	Singin' in the Rain	[Comedy, Musical, Romance]	1952
5	1	1088	4.0	Dirty Dancing	[Drama, Musical, Romance]	1987
6	1	1175	3.5	Delicatessen	[Comedy, Drama, Romance]	1991
7	1	1217	3.5	Ran	[Drama, War]	1985
8	1	1237	5.0	Seventh Seal, The (Sjunde inseglet, Det)	[Drama]	1957
9	1	1250	4.0	Bridge on the River Kwai, The	[Adventure, Drama, War]	1957

Content-Based Recommendation system

In [10]:

```
userInput = [
            {'title':'Fast & Furious (Fast and the Furious 4, The)', 'rating':4
},
            {'title':'Jumanji: Welcome to the Jungle', 'rating':4.5},
            {'title':'Secret Superstar', 'rating':3},
            {'title':"Captain Marvel", 'rating':3.5},
            {'title':'Intern, The', 'rating':5}
         ]
inputMovies = pd.DataFrame(userInput)
inputMovies
```

Out[10]:

	rating	title
0	4.0	Fast & Furious (Fast and the Furious 4, The)
1	4.5	Jumanji: Welcome to the Jungle
2	3.0	Secret Superstar
3	3.5	Captain Marvel
4	5.0	Intern, The

In [11]:

```
#Filtering out the movies by title
inputId = movies_data[movies_data['title'].isin(inputMovies['title'].tolist())]
#Then merging it so we can get the movieId. It's implicitly merging it by title.
inputMovies = pd.merge(inputId, inputMovies)
#Dropping information we won't use from the input dataframe
inputMovies = inputMovies.drop('genres', 1).drop('year', 1)
#Final input dataframe
#If a movie you added in above isn't here, then it might not be in the original
#dataframe or it might spelled differently, please check capitalisation.
inputMovies
```

Out[11]:

	movieId	title	rating
0	67923	Fast & Furious (Fast and the Furious 4, The)	4.0
1	79022	Intern, The	5.0
2	122910	Captain Marvel	3.5
3	179397	Secret Superstar	3.0
4	179401	Jumanji: Welcome to the Jungle	4.5

In [12]:

```
#Filtering out the movies from the input
userMovies = moviesWithGenres_data[moviesWithGenres_data['movieId'].isin(inputMo
vies['movieId'].tolist())]
userMovies
```

Out[12]:

	movieId	title	genres	year	Adventure	Animation	Children	Comedy	Fantas
13226	67923	Fast & Furious (Fast and the Furious 4, The)	[Action, Crime, Drama, Thriller]	2009	0.0	0.0	0.0	0.0	0.(
14914	79022	Intern, The	[Comedy]	2000	0.0	0.0	0.0	1.0	0.(
25066	122910	Captain Marvel	[Action, Adventure, Sci-Fi]	2018	1.0	0.0	0.0	0.0	0.(
49685	179397	Secret Superstar	[Drama]	2017	0.0	0.0	0.0	0.0	0.(
49687	179401	Jumanji: Welcome to the Jungle	[Action, Adventure, Children]	2017	1.0	0.0	1.0	0.0	0.(

5 rows × 24 columns

In [13]:

```
#Resetting the index to avoid future issues
userMovies = userMovies.reset_index(drop=True)
#Dropping unnecessary issues due to save memory and to avoid issues
userGenreTable = userMovies.drop('movieId', 1).drop('title', 1).drop('genres', 1
).drop('year', 1)
userGenreTable
```

Out[13]:

	Adventure	Animation	Children	Comedy	Fantasy	Romance	Drama	Action	Crime	Thrill
0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1
1	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0
2	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0
3	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0
4	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0

In [14]:

```
inputMovies['rating']
```

Out[14]:

```
0    4.0
1    5.0
2    3.5
3    3.0
4    4.5
Name: rating, dtype: float64
```

In [15]:

```
#Dot produt to get weights
userProfile = userGenreTable.transpose().dot(inputMovies['rating'])
#The user profile
userProfile
```

Out[15]:

```
Adventure              8.0
Animation              0.0
Children               4.5
Comedy                 5.0
Fantasy                0.0
Romance                0.0
Drama                  7.0
Action                12.0
Crime                  4.0
Thriller               4.0
Horror                 0.0
Mystery                0.0
Sci-Fi                 3.5
IMAX                   0.0
Documentary            0.0
War                    0.0
Musical                0.0
Western                0.0
Film-Noir              0.0
(no genres listed)     0.0
dtype: float64
```

In [16]:

```
#Now let's get the genres of every movie in our original dataframe
genreTable = moviesWithGenres_data.set_index(moviesWithGenres_data['movieId'])
#And drop the unnecessary information
genreTable = genreTable.drop('movieId', 1).drop('title', 1).drop('genres', 1).dr
op('year', 1)
genreTable.head()
```

Out[16]:

movieId	Adventure	Animation	Children	Comedy	Fantasy	Romance	Drama	Action	Crime
1	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0
2	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0
4	0.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0
5	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0

In [17]:

```
genreTable.shape
```

Out[17]:

```
(62423, 20)
```

In [18]:

```
#Multiply the genres by the weights and then take the weighted average
recommendationTable_data = ((genreTable*userProfile).sum(axis=1))/(userProfile.s
um())
recommendationTable_data.head()
```

Out[18]:

```
movieId
1    0.364583
2    0.260417
3    0.104167
4    0.250000
5    0.104167
dtype: float64
```

In [19]:

```
#Sort our recommendations in descending order
recommendationTable_data = recommendationTable_data.sort_values(ascending=False)
#Just a peek at the values
recommendationTable_data.head()
```

Out[19]:

```
movieId
144324    0.833333
122787    0.833333
81132     0.833333
64645     0.833333
115479    0.802083
dtype: float64
```

In [20]:

```
#The final recommendation table
movies_data.loc[movies_data['movieId'].isin(recommendationTable_data.head(20).keys())]
```

Out[20]:

	movieId	title	genres	year
4850	4956	Stunt Man, The	[Action, Adventure, Comedy, Drama, Romance, Th...	1980
5546	5657	Flashback	[Action, Adventure, Comedy, Crime, Drama]	1990
6865	6990	The Great Train Robbery	[Action, Adventure, Comedy, Crime, Drama]	1978
9315	27735	Unstoppable	[Action, Adventure, Comedy, Drama, Thriller]	2004
11829	55116	Hunting Party, The	[Action, Adventure, Comedy, Drama, Thriller]	2007
12879	64645	The Wrecking Crew	[Action, Adventure, Comedy, Crime, Drama, Thri...	1968
13238	68033	Vigilante Force	[Action, Adventure, Crime, Drama, Thriller]	1976
13380	69095	Graduation	[Action, Adventure, Comedy, Crime, Drama]	2007
15389	81132	Rubber	[Action, Adventure, Comedy, Crime, Drama, Film...	2010
15793	83266	Kaho Naa... Pyaar Hai	[Action, Adventure, Comedy, Drama, Mystery, Ro...	2000
22553	115479	Whip Hand, The	[Action, Adventure, Crime, Drama, Sci-Fi, Thri...	1951
23306	117646	Dragonheart 2: A New Beginning	[Action, Adventure, Comedy, Drama, Fantasy, Th...	2000
25008	122787	The 39 Steps	[Action, Adventure, Comedy, Crime, Drama, Thri...	1959
31401	138522	Bail Out	[Action, Adventure, Comedy, Drama, Thriller]	1990
33107	142418	Joseph Andrews	[Action, Adventure, Comedy, Drama, Romance, Th...	1977
33940	144324	Once Upon a Time	[Action, Adventure, Comedy, Crime, Drama, Roma...	2008
33953	144350	Under the Mountain	[Action, Adventure, Children, Drama, Fantasy, ...	2009
39119	156479	Follow the Leader	[Action, Adventure, Comedy, Crime, Drama]	1944
40057	158585	Bal-Can-Can	[Action, Adventure, Comedy, Crime, Drama, War]	2005
51529	183347	Dragnet	[Action, Adventure, Crime, Drama, Thriller]	1954

Collaborative Filtering based Recommendation System

In [21]:

```
#Filtering out users that have watched movies that the input has watched and sto
ring it
userSubset = ratings_data[ratings_data['movieId'].isin(inputMovies['movieId'].to
list())]
userSubset.head()
```

Out[21]:

	userId	movieId	rating
683	3	67923	3.5
1140	4	179401	3.5
6289	44	67923	4.5
6754	51	179401	4.0
14751	114	179401	3.5

In [22]:

```
#Groupby creates several sub dataframes where they all have the same value in th
e column specified as the parameter
userSubsetGroup = userSubset.groupby(['userId'])
```

In [23]:

```
userSubsetGroup.get_group(114)
```

Out[23]:

	userId	movieId	rating
14751	114	179401	3.5

In [24]:

```
#Sorting it so users with movie most in common with the input will have priority
userSubsetGroup = sorted(userSubsetGroup,  key=lambda x: len(x[1]), reverse=True
)
```

In [25]:

```
userSubsetGroup[0:3]
```

Out[25]:

```
[(6115,         userId  movieId  rating
  909126    6115    67923     3.5
  909480    6115   122910     4.5
  909633    6115   179397     3.5
  909634    6115   179401     3.5), (123832,           userId  movie
Id  rating
  19097191  123832    67923     3.0
  19097451  123832   122910     4.0
  19097752  123832   179397     4.0
  19097753  123832   179401     3.5), (130333,
           userId  movieId  rating
  20049601  130333    67923     4.5
  20050313  130333   122910     2.0
  20050995  130333   179397     0.5
  20050996  130333   179401     3.5)]
```

In [26]:

```
userSubsetGroup = userSubsetGroup[0:100]
```

In [27]:

```
#Store the Pearson Correlation in a dictionary, where the key is the user Id and
the value is the coefficient
pearsonCorrelationDict = {}

#For every user group in our subset
for name, group in userSubsetGroup:
    #Let's start by sorting the input and current user group so the values are
n't mixed up later on
    group = group.sort_values(by='movieId')
    inputMovies = inputMovies.sort_values(by='movieId')
    #Get the N for the formula
    nRatings = len(group)
    #Get the review scores for the movies that they both have in common
    temp_df = inputMovies[inputMovies['movieId'].isin(group['movieId'].tolist
())]
    #And then store them in a temporary buffer variable in a list format to faci
litate future calculations
    tempRatingList = temp_df['rating'].tolist()
    #Let's also put the current user group reviews in a list format
    tempGroupList = group['rating'].tolist()
    #Now let's calculate the pearson correlation between two users, so called, x
and y
    Sxx = sum([i**2 for i in tempRatingList]) - pow(sum(tempRatingList),2)/float
(nRatings)
    Syy = sum([i**2 for i in tempGroupList]) - pow(sum(tempGroupList),2)/float(n
Ratings)
    Sxy = sum( i*j for i, j in zip(tempRatingList, tempGroupList)) - sum(tempRat
ingList)*sum(tempGroupList)/float(nRatings)

    #If the denominator is different than zero, then divide, else, 0 correlatio
n.
    if Sxx != 0 and Syy != 0:
        val = (Sxx*Syy)**(1/2)
        pearsonCorrelationDict[name] = Sxy/(val)
    else:
        pearsonCorrelationDict[name] = 0
```

In [28]:

```
pearsonCorrelationDict.items()
```

Out[28]:

```
dict_items([(6115, -0.2581988897471611), (123832, -0.67419986246324
2), (130333, 0.8483677805978151), (997, 0), (1977, -0.95382096647653
25), (4019, -0.9819805060619666), (4429, -0.1889822365046136), (451
5, 0), (4642, 0.5), (5694, 0), (6184, 0), (6729, 0.0), (8476, 0.0),
(9770, 0.0), (10502, 0.8660254037844448), (11005, 0.866025403784438
7), (12552, 0.0), (13134, 0.0), (16774, 0.8660254037844355), (17669,
0), (18265, 0.8660254037844448), (18434, 0.8660254037844448), (1850
0, 1.0), (18580, -0.5), (19379, 0.0), (19475, -0.9933992677987827),
(20068, 0), (20302, -0.8660254037844402), (20917, 0.5), (22241, 0.
0), (22349, 0.8660254037844402), (22840, 0.0), (24523, 0.8660254037844402), (24906, 0.5), (25716, 0.8660254037844448), (28812, 0.8660254037844355), (30643, -0.32732683535398843), (32922, -0.18898223650461
36), (33069, 0.8660254037844355), (35734, 0.8660254037844448), (3724
6, 0), (37438, 1.0), (37635, -0.8660254037844448), (38588, -1.0), (3
8672, 0.0), (38806, 0.0), (40006, 0.8660254037844448), (41657, 0),
(45948, 1.0), (46342, 0.0), (46633, -0.5), (47153, -0.7559289460184544), (49004, 0.5), (49403, 0.2401922307076306), (49638, 0.3273268353539889), (51278, -0.8660254037844402), (51392, -0.5), (51799, -1.0),
(53289, 0.0), (53837, -0.5), (54071, 0.32732683535398843), (54137,
0.866025403784439), (55189, 0.8660254037844448), (59398, 0.0), (5959
1, -0.8660254037844387), (60567, 0.944911182523068), (60632, -0.8660254037844387), (63873, 0.0), (64018, 0.9607689228305233), (64107, 0.
5), (66268, 0.0), (66315, 0.9819805060619666), (66353, 0.8660254037844355), (66401, 0.0), (69928, -0.59603956067927), (71293, 0), (7231
5, -0.8660254037844355), (72601, 0.5), (73824, 0.0), (74608, -0.5),
(77237, 0.8660254037844355), (78438, -0.8660254037844386), (79382,
0), (79910, 0), (81694, 1.0), (81867, 0.3273268353539889), (82463,
0.0), (82788, 0.0), (82994, -0.8660254037844402), (83446, -0.8660254037844402), (84440, 0.0), (84801, -0.755928946018466), (86339, 0.155
54275420956376), (86462, 0.3273268353539889), (90785, 0.277350098112
6146), (91400, 0.8660254037844448), (92091, 0.0), (94234, 0.9819805060619652), (94334, 0.32732683535398843), (95075, 0.8660254037844355)])
```

In [29]:

```
pearsonData = pd.DataFrame.from_dict(pearsonCorrelationDict, orient='index')
pearsonData.columns = ['similarityIndex']
pearsonData['userId'] = pearsonData.index
pearsonData.index = range(len(pearsonData))
pearsonData.head()
```

Out[29]:

	similarityIndex	userId
0	-0.258199	6115
1	-0.674200	123832
2	0.848368	130333
3	0.000000	997
4	-0.953821	1977

In [30]:

```
topUsers=pearsonData.sort_values(by='similarityIndex', ascending=False)[0:50]
topUsers.head()
```

Out[30]:

	similarityIndex	userId
22	1.000000	18500
84	1.000000	81694
48	1.000000	45948
41	1.000000	37438
71	0.981981	66315

In [31]:

```
topUsersRating=topUsers.merge(ratings_data, left_on='userId', right_on='userId',
how='inner')
topUsersRating.head()
```

Out[31]:

	similarityIndex	userId	movieId	rating
0	1.0	18500	1	4.5
1	1.0	18500	2	3.0
2	1.0	18500	6	4.5
3	1.0	18500	9	3.0
4	1.0	18500	16	4.0

In [32]:

```
#Multiplies the similarity by the user's ratings
topUsersRating['weightedRating'] = topUsersRating['similarityIndex']*topUsersRat
ing['rating']
topUsersRating.head()
```

Out[32]:

	similarityIndex	userId	movieId	rating	weightedRating
0	1.0	18500	1	4.5	4.5
1	1.0	18500	2	3.0	3.0
2	1.0	18500	6	4.5	4.5
3	1.0	18500	9	3.0	3.0
4	1.0	18500	16	4.0	4.0

In [33]:

```
#Applies a sum to the topUsers after grouping it up by userId
tempTopUsersRating = topUsersRating.groupby('movieId').sum()[['similarityIndex',
'weightedRating']]
tempTopUsersRating.columns = ['sum_similarityIndex','sum_weightedRating']
tempTopUsersRating.head()
```

Out[33]:

movieId	sum_similarityIndex	sum_weightedRating
1	27.056808	108.273770
2	19.817809	65.820656
3	3.155066	8.917317
4	0.240192	0.120096
5	3.387478	8.199806

In [34]:

```
#Creates an empty dataframe
recommendation_data = pd.DataFrame()
#Now we take the weighted average
recommendation_data['weighted average recommendation score'] = tempTopUsersRatin
g['sum_weightedRating']/tempTopUsersRating['sum_similarityIndex']
recommendation_data['movieId'] = tempTopUsersRating.index
recommendation_data.head()
```

Out[34]:

movieId	weighted average recommendation score	movieId
1	4.001720	1
2	3.321288	2
3	2.826349	3
4	0.500000	4
5	2.420622	5

In [35]:

```
recommendation_data = recommendation_data.sort_values(by='weighted average recommendation score', ascending=False)
recommendation_data.head(10)
```

Out[35]:

movieId	weighted average recommendation score	movieId
82037	5.0	82037
2677	5.0	2677
96563	5.0	96563
101971	5.0	101971
101648	5.0	101648
100540	5.0	100540
99549	5.0	99549
168846	5.0	168846
99045	5.0	99045
196115	5.0	196115

In [36]:

```
movies_data.loc[movies_data['movieId'].isin(recommendation_data.head(10)['movieId'].tolist())]
```

Out[36]:

	movieId	title	genres	year
2585	2677	Buena Vista Social Club	[Documentary, Musical]	1999
15575	82037	Tillman Story, The	[Documentary]	2010
18436	96563	Paradise Lost 3: Purgatory	[Documentary]	2011
19004	99045	Aftershock (Tangshan dadizhen)	[Drama, IMAX]	2010
19106	99549	Mansome	[Documentary]	2012
19361	100540	Bronies: The Extremely Unexpected Adult Fans o...	[Documentary]	2012
19557	101648	Flat, The	[Documentary]	2011
19643	101971	Never Sleep Again: The Elm Street Legacy	[Documentary]	2010
44728	168846	Neal Brennan: 3 Mics	[Comedy]	2017
57325	196115	Prosecuting Evil: The Extraordinary World of B...	[Documentary]	2018

CHAPTER : 7

CONCLUSION

8.1 SUMMARY

In this project, both content based and collaborative filtering based movie recommendation system are made and executed on Jupyter Notebook. These tests and results were processed on IBM Watson Studio and copied here. This project demonstrates recommendations to a user based on his rating to 5 random movies. Two methods were applied based on which different results were presented.

8.2 ADVANTAGES OF THE SYSTEM

1) The System would benefit those users who have to use search engines to locate their favorite movie titles. They have to scroll through pages of results to find relevant content.

2) Rather than searching for quality movies, the users of this system would be directly taken to quality movies matching their personal interests and preferences.

3) The system would deliver quality movies as it is not just dependent on the rating given by other users which could be deceiving at times.

8.3 FUTURE SCOPE

- The development of this project surely prompts many new areas of investigations. This project has wide scope to implement it in any ecommerce websites.
- This project covers all functionalities related to Book Selling. Hence it can be implemented anywhere else after minute in a click
- The sharing facility will be available only to the logged in users.

- We are also planning to provide the user with the facility to define different themes of the apps and provide online chat system.

REFERENCES

Netflix Recommendations: Beyond the 5 stars. Netflix Technology Blog | 2012

Robin Burke1 , Alexander Felfernig2 , Mehmet H. Göker3, “Recommender Systems: An Overview ”, Association for the Advancement of Artificial Intelligence (www.aaai.org) ,Research 2011

files.grouplens.org/datasets/movielens

www.google.com

pandas.pydata.org

www.cognitiveclass.ai

www.matplotlib.org/

www.numpy.org

docs.python.org/3

dataplatform.cloud.ibm.com/

www.towardsdatascience.com/