Muhammad Usman Abbasi 2014262

Raheel Jameel 2014294

Syed Saadat Hussain Rizvi 2014370

Data Warehousing and Data Mining

CS437

Semester Project

Contents

[Simple IMDB Recommender 2](#_Toc513581322)

[Results 3](#_Toc513581323)

[Content-Based Recommender 4](#_Toc513581324)

[Plot Description Based Recommender 4](#_Toc513581325)

[Results 5](#_Toc513581326)

[Credits, Genres and Keywords Based Recommender 6](#_Toc513581327)

[Results 7](#_Toc513581328)

[Movie Recommendation based on User ratings using K-Means 9](#_Toc513581329)

[Processing 10](#_Toc513581330)

[Results 12](#_Toc513581331)

[Collaborative Filtering 14](#_Toc513581332)

[User-based Collaborative filtering: 14](#_Toc513581333)

[Item-based Collaborative filtering: 14](#_Toc513581334)

[Conclusion 22](#_Toc513581335)

# Simple IMDB Recommender

It is a basic system to recommend to recommend top movies based on the calculated scores using a weighted rating formula. We could have simply used movie rating as our recommendation metric, however this has a few caveats.

1. It doesn’t take the popularity of a movie into consideration. For example, a recently released local movie with a relatively small size (e.g. 30) of audience and voters (who voted this movie 9.5) would be considered better than a movie with a rating of 8.5 with voters over 100,000.
2. It will in general always prefer movies with extremely high ratings and will not wait for the ratings to regularize after a sufficient viewership(votes) have been achieved.

Taking this into consideration, a weighted rating system that takes into account both the number of voters as well as the respective ratings of a movie would prove to be better. An example of this would be IMDB’s Top 250 Chart.

The formula that we used is as follows:

*Weighted Rating (WR) = (v/(v+m) \* R) + (m/(m+v) \* C)*

Where,

* v is the number of votes for the movie;
* m is the minimum votes required to be listed in the chart;
* R is the average rating of the movie;
* C is the mean vote across the whole report

**Parameters**: For the value of ‘m’, I have used a cutoff of 90%. This means that for a movie to be in the top 10 recommended movies list, it must have more voters than 90% of the movies.

### Results

The following two graphs show the change in results when quantile(m) is changed from 0.80 to 0.90:

# Content-Based Recommender

The main idea behind such systems is that if a person liked a movie then he/she will also like a movie that is very similar to the previously liked movie. It uses the metadata of a movie such as plot description, top actors, directors, genre etc. to suggest other movies

## Plot Description Based Recommender

In this recommender, we computed pairwise similarity scores for all movies in the dataset based on their plot descriptions and recommended movies accordingly.

We computed Term Frequency-Inverse Document Frequency (TF-IDF) vectors for each movie plot.

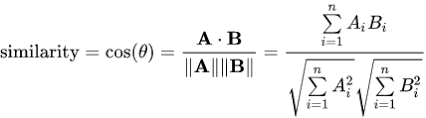
• TF: Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:

*TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).*

• IDF: Inverse Document Frequency, which measures how important a term is. While computing TF, all terms are considered equally important. However, it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

*IDF(t) = log(Total number of documents / Number of documents with term t in it).*

After constructing the TF-IDF matrix, I learned that in my dataset there were **75827** words used to describe **45466** movies. I then used the cosine similarity to calculate the numeric quantity that denotes the similarity between two movies.



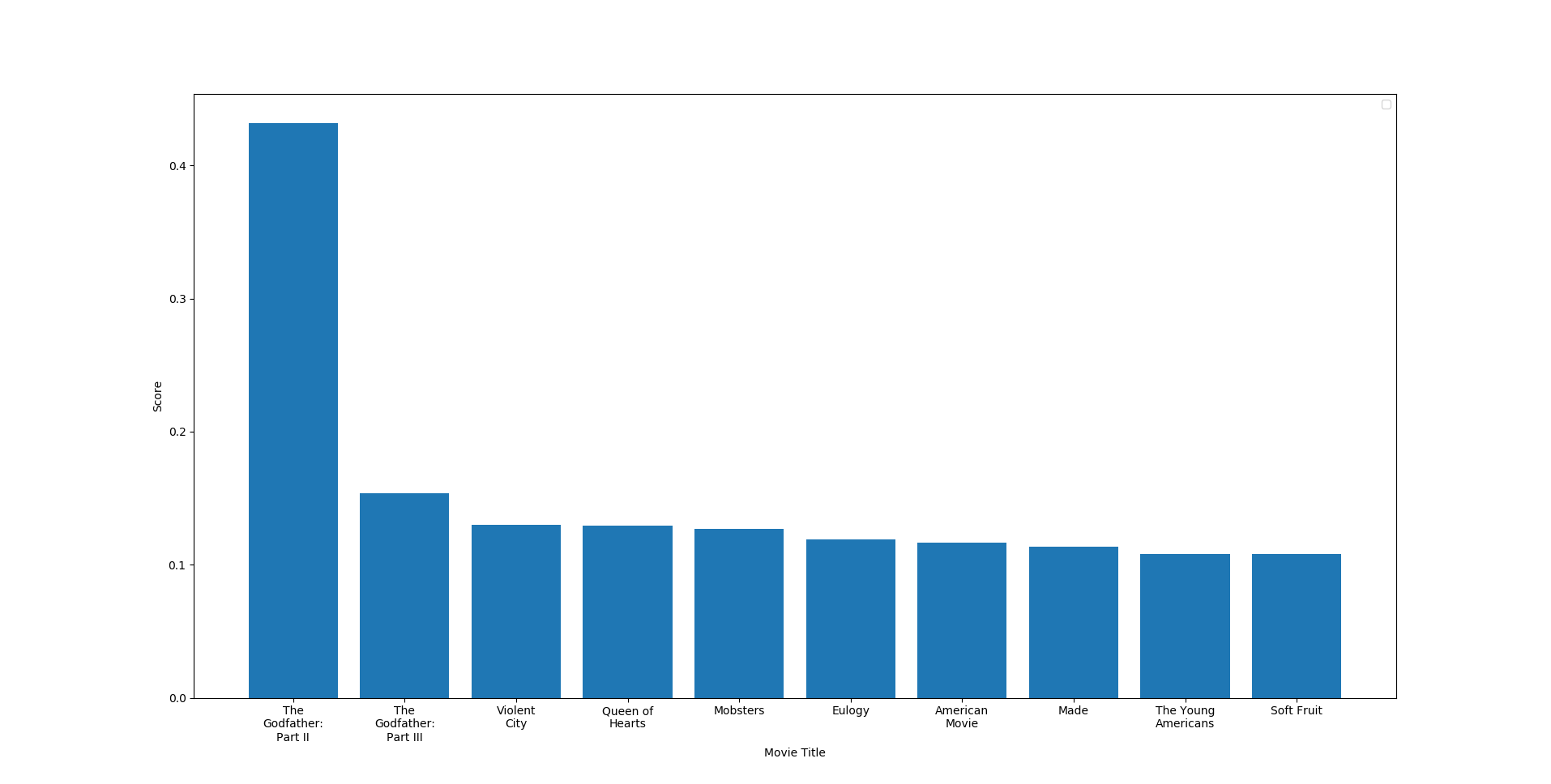
Where

Ai = Term Frequency\*Inverse Document Frequency for word at index i in A,

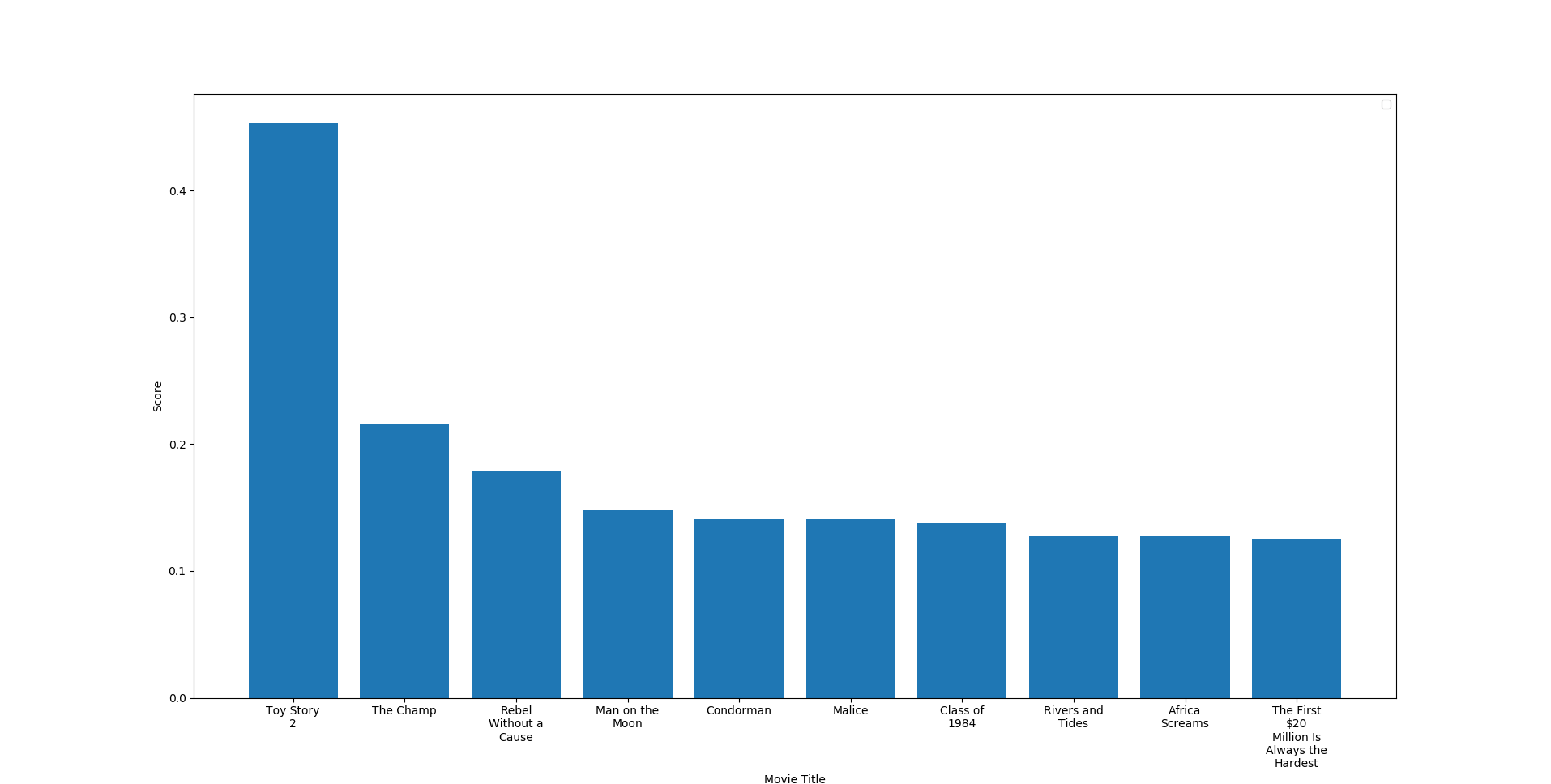
Bi= Term Frequency\*Inverse Document Frequency for word at index i in B

### Results

#### Input Movie: 'The Godfather'



#### Input Movie: ' Toy Story’



## Credits, Genres and Keywords Based Recommender

The basic idea behind this system was that if a user enjoys a particular actor or director or genre, then this recommender would recommend movies that contain those entities.

In this recommender, I used the following metadata:

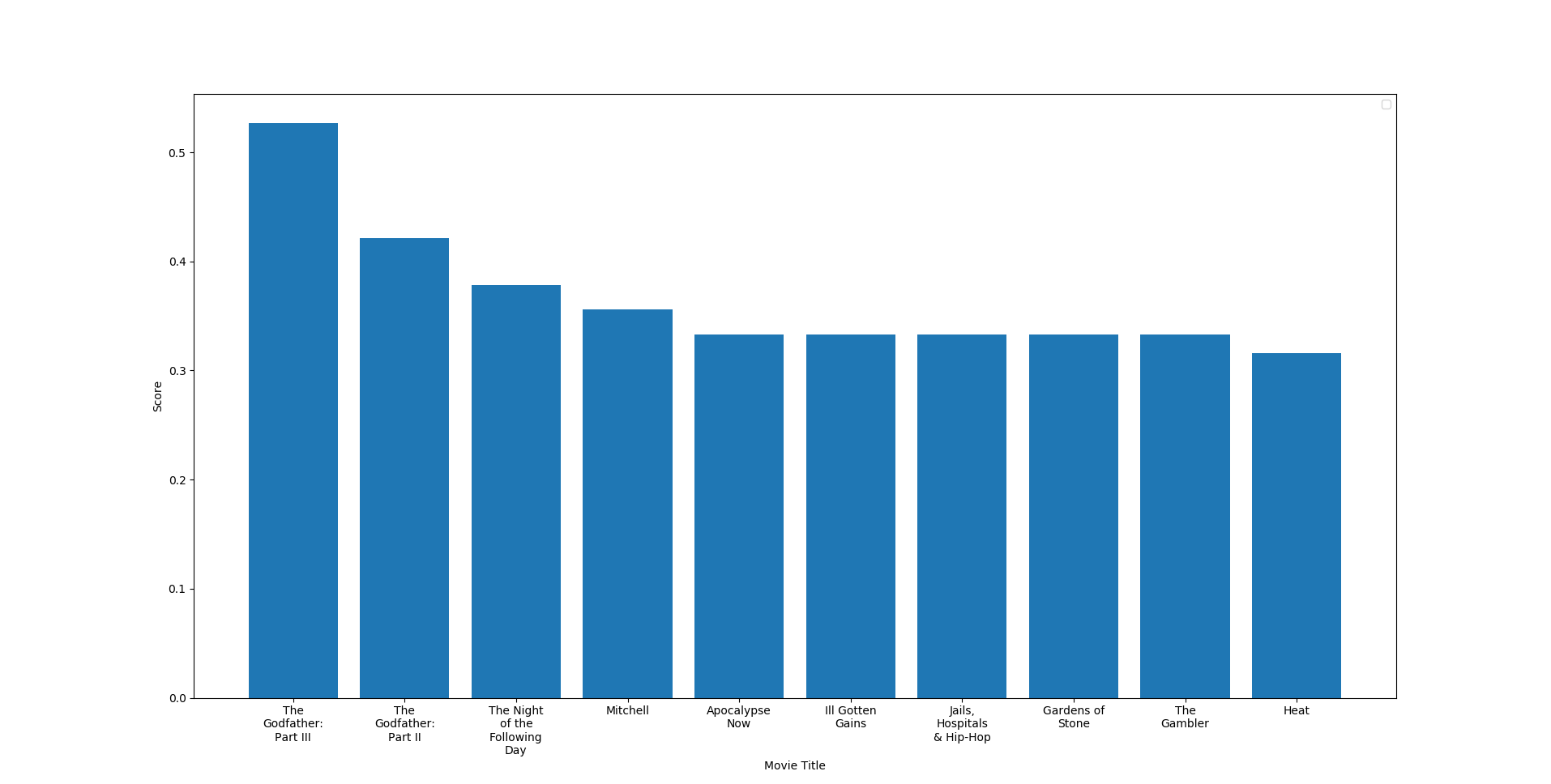
* 3 top actors
* Movie Director
* Related Genre
* Movie Plot Keywords

The main challenge in this recommender was to merge three different data sets and then convert their combined data into useful form. One of the most important steps was to convert all the names and keywords into lowercase and to remove all spaces in between. This would ensure that Johnny of "Johnny Depp" and "Johnny Galecki" are not counted as the same actor.

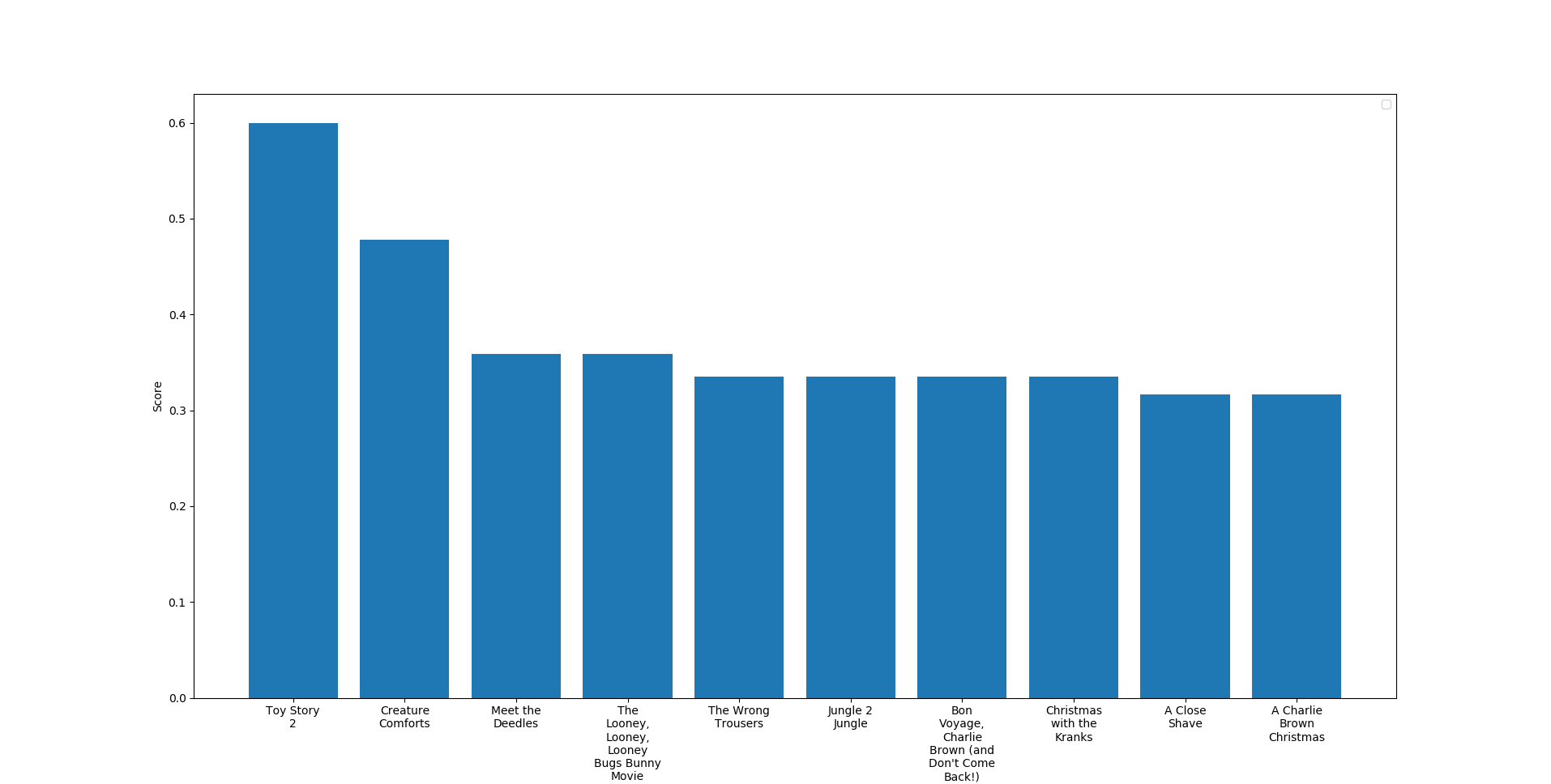
This time I didn’t use TF\_IDF to check for similarity because I didn’t want to down-weight the any actor/director if he/she has acted/directed more movies.

### Results

#### Input Movie: 'The Godfather'



#### Input Movie: ' Toy Story’



# Movie Recommendation based on User ratings using K-Means

This recommendation system takes into account the ratings given to different movies by many people and uses that information to recommend movies based on the ratings of movies given by the target user.

In a way it examines the target user based on his/her previously given reviews to movies, finds out the users who gave similar reviews to the commonly watched movies and then recommends movies which it predicts if watched by the target user, he/she would give a high rating to them also.

This system uses K-Means Clustering algorithm to group together people with similar reviews.

K-Means Clustering is one of the most popular Machine Learning algorithms for cluster analysis in data mining. K-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster.

K-Means algorithm is an unsupervised learning algorithm, i.e. it needs no training data, it performs the computation on the actual dataset. This should be apparent from the fact that in K Means, we are just trying to group similar data points into clusters, there is no prediction involved.

The K Means algorithm is easy to understand and to implement. It works well in a large number of cases and is a powerful tool to have in the closet.

The K Means algorithm is iterative based, it repeatedly calculates the cluster centroids, refining the values until they do not change much.

The k-means algorithm takes a dataset of ‘n’ points as input, together with an integer parameter ‘k’ specifying how many clusters to create (supplied by the programmer). The output is a set of ‘k’ cluster centroids and a labelling of the dataset that maps each of the data points to a unique cluster.

**Step 1:**

We randomly pick K cluster centers (centroids). Let’s assume these are c1,c2,…,ck, and we can say that;



C is the set of all centroids

**Step 2:**

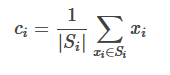
In this step we assign each input value to closest center. This is done by calculating Euclidean (L2) distance between the point and the each centroid.



Where dist(.) is the Euclidean distance.

**Step 3:**

In this step, we find the new centroid by taking the average of all the points assigned to that cluster.



Si is the set of all points assigned to the i’th cluster.

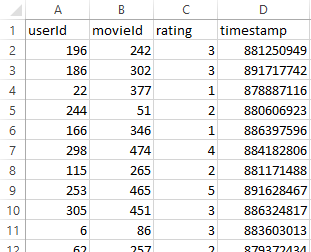
**Step 4:**

In this step, we repeat step 2 and 3 until none of the cluster assignments change or until a maximum number of iterations is reached. That means until our clusters remain stable, we repeat the algorithm.

## Processing

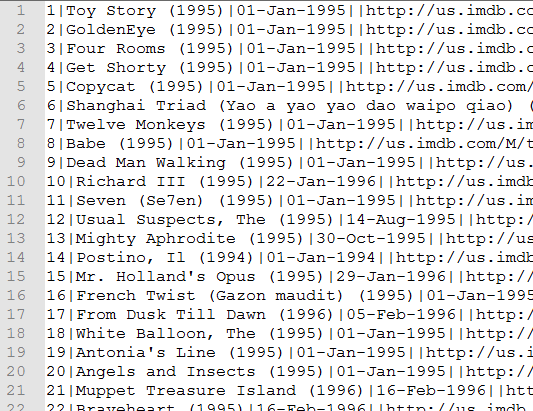
A list of user and their ratings were used. The full u data set consists of 100000 ratings by 943 users on 1682 movies. Each user has rated at least 20 movies.

A subset of that data is shown below:

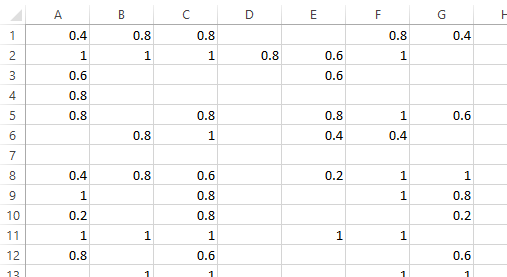


This contains the main data, the users and the ratings on a scale of 1 to 5 which they gave to various movies.

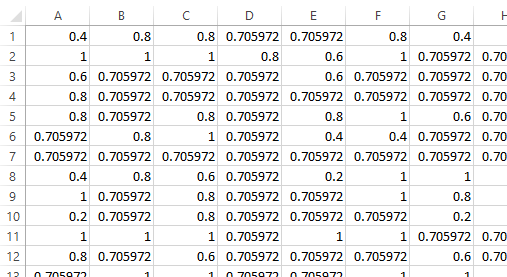
The subset of movie information is shown:



The users and their ratings data are then rearranged. The rows represent the users and the columns represent the movies and the rating the user gave:



These are then normalised and the movies for which the user did not give any rating is set to the average of all the ratings given by all users:

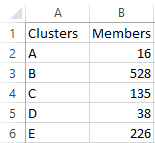


After this the K-Means algorithm is run on this data many times with different values of K, the number of clusters required.

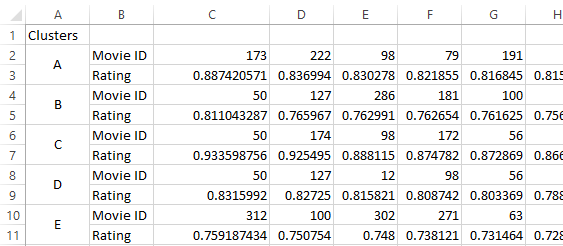
## Results

An interesting pattern was observed in the data that for a big number of K the clusters formed were skewed. Majority of the clusters only got a few members, less than 5, and the remaining few clusters (around 4) contained all of the members. This shows that the data has a natural tendency to form large clusters. After running the system multiple time on many different values of K showed 5 number of clusters to be able to effectively partition the data.

The results are as below:



And these are the centroids of these clusters, sorted on the highest ratings. Which show the top movies which the members of these clusters would like.



Now this will act as the recommendation data set. Any new user will be required to rate at least 10 movies of his choosing. And then on the bases of his data the similarity will be calculated to find which cluster this user belongs to and then the top rated movies of that cluster will be recommended movies for this user.

# Collaborative Filtering

Collaborative Filtering techniques make recommendations for a user based on ratings and preferences data of many users. The main underlying idea is that if two users have both liked certain common items, then the items that one user has liked that the other user has not yet tried can be recommended to him. We see collaborative filtering techniques in action on various Internet platforms such as Amazon.com, Netflix and Facebook.

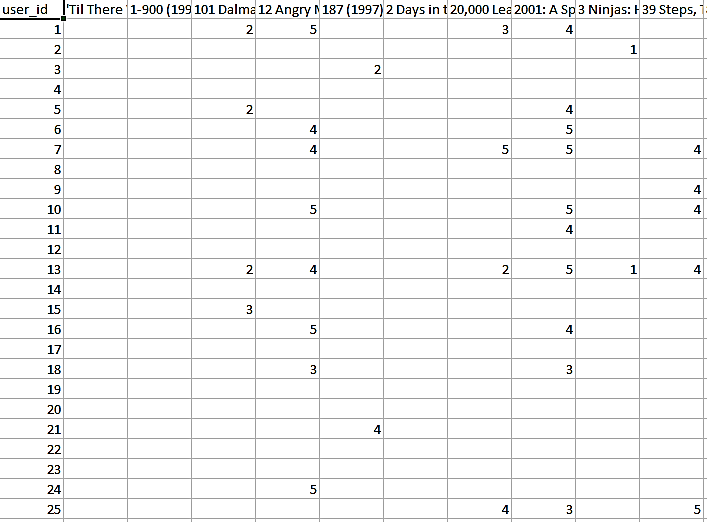
## User-based Collaborative filtering:

In user-based collaborative filtering, we see how similar two users are with respect to the items that both the users have rated in past. However, the drawbacks of this technique are:

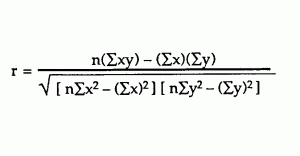
1. Data Scarcity: When the number of items is large, number of items a user has rated reduces to a very small percentage. This makes the correlation coefficient less reliable to be used for recommendation purposes.
2. Users’ preferences change depending on their mood and hence they rate things differently. This results in re-computation of the whole model which takes extra time and resources.

## Item-based Collaborative filtering:

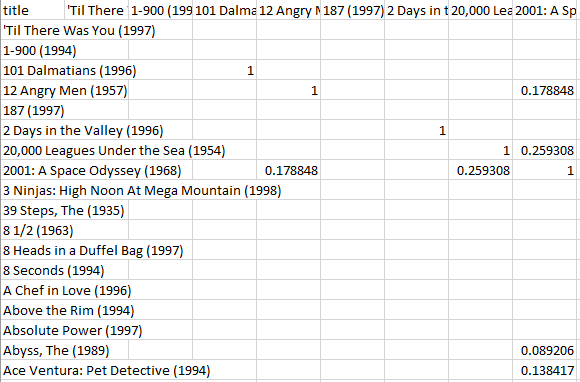
To cater and resolve the drawbacks of user-based collaborative filtering, we will use item-based collaborative filtering. In this technique, instead of finding out similar users, we find out items that are similar to each other. Here, item similarity means how people treat two items in terms of like and dislike and not by their attributes. This technique is also considered better than user-based collaborative filtering because an average item has a lot more ratings than the average user. In other words, the number of ratings per item/object are much greater.

We converted the data from dataset into the following form:

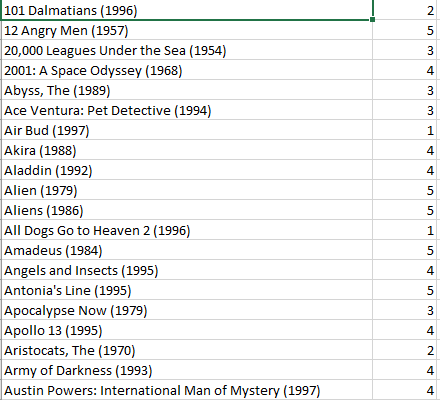
After that, the similarity between pairs of items is calculated by using Pearson co-relation between rating vectors. The formula for Pearson co-relation is:



We used built-in functions of panda data structure in Python to speed up our processing and reducing the calculations errors to minimum. We set the minimum number of observations required per pair of items to be 50. After applying Pearson co-relation on our data, we got the following data:

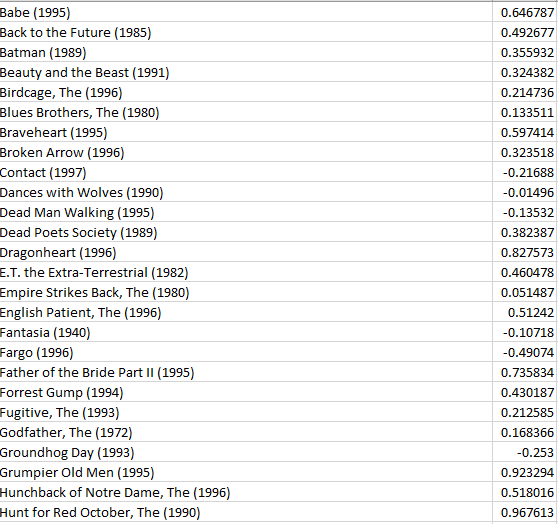


Then we found out all the movies that the user 1 in our dataset had rated. This gave us the following output:

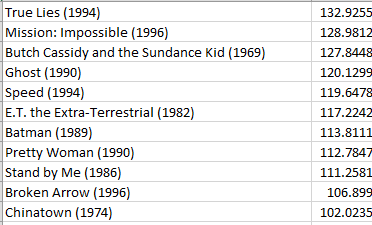


Some of the movies rated by User 1:

After that, we found movies that were similar to each other (had some co-relation between them):

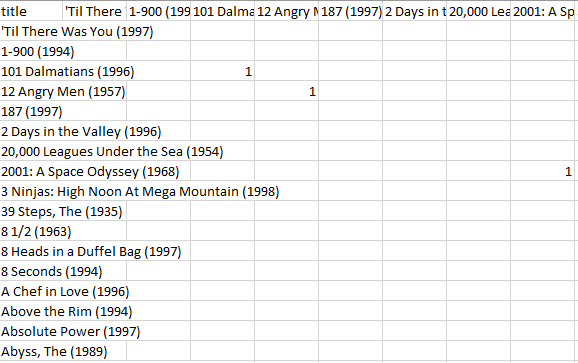


This data was then sorted in descending order with respect to the co-relation value. Finally sum of co-relations was calculated and data was filtered for duplicate values. The final result is given below:

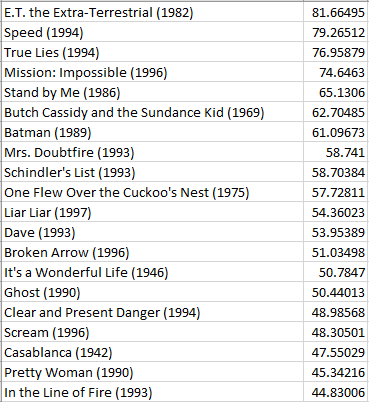


If we change the minimum number of observations required per pair of items from 50 to 100, the changes would be:

Co-Relation matrix:



Final result:



Similarly, running the same program for User 2 with minimum observation value of 100 would give us the following results:



# Conclusion

Our Simple IMDB Movie Recommender only calculates weighted ratings of movies and then recommends them to users. This can be useful if the user is completely new to the system. And the system already has several movies along with their ratings.

Content-based filtering with respect to movie plot does not give personalized recommendations and does not require movie ratings. It recommends movies which have the plot closest to a certain movie.

Content-based filtering based on top actors, director and genre also does not give personalized recommendations and finds movies closest to the top actors, director and genre parameters.

K-Means algorithm requires a user to have rated a certain number of movies. These ratings will be used by our system to put the user in a cluster of users who have similarly rated movies. For this technique, a large set of movies, movie ratings, users and user ratings are required.

Item-based collaborative filtering gives personalized recommendations. This technique is useful where we have a large number of ratings of a certain item and the number of users is less than that.