GNG 5125

Data Science Application

Movie Recommendation System

Submitted to

Dr. Arya Rahgozar

Teaching Assistant, Migao Wu

Group 1

Femi Ajayi -8266716

Hossein Davarzanisani- 300102825

Chaitanya Garhwal- 7986325

Chinmay Garhwal-7983060

April 12th, 2021

University of Ottawa

Table of Contents

1.Introduction:

Significant dependencies exist between user and itemcentric activity. A successful recommendation system explores this activity and produces relevant suggestions. For example, a user who is interested in a historical documentary is more likely to be interested in another historical documentary or an educational program, rather than in an action movie. In many cases, various categories of items may show significant correlations, which can be leveraged to make more accurate recommendations. A movie recommendation system is important due to its strength in providing enhanced entertainment and personalized user experience.

1.1- Methodologies

There are inundated movies released every year and users inhabit varied choice of movies. So, it is important that movie recommendation engines keep users engaged by recommending the movie of his/her choice. In this project, we have implemented and evaluated content based, collaborative based (Item-Item, User-Item, SVD, SVD++) filtering on Movielens data. Moreover, using content-based and SVD based filtering, we used a hybrid model by stacking these models to increase the accuracy of movies recommended to the user. This project addresses the implementation and evaluation of models listed above.

1.2-Dataset

For this research project, we have used MovieLens 100K dataset. The GroupLens Research Project at the University of Minnesota collected MovieLens datasets.
This dataset consists of:

- 100,000 ratings (1-5) from 943 users on 1682 movies
- Each user has rated at least 20 movies

The data was collected through the MovieLens website (movielens.umn.edu)
We have used two csv files, the ratings.csv contains user id, movie id and the rating the user gave to that movie. Movie.csv contains

1.3-Python Libraries

The following libraries used for this project:

- Pandas
- Numpy
- Sklearn
- Seaborn
- SciPy

- Surprise
- Matplotlib

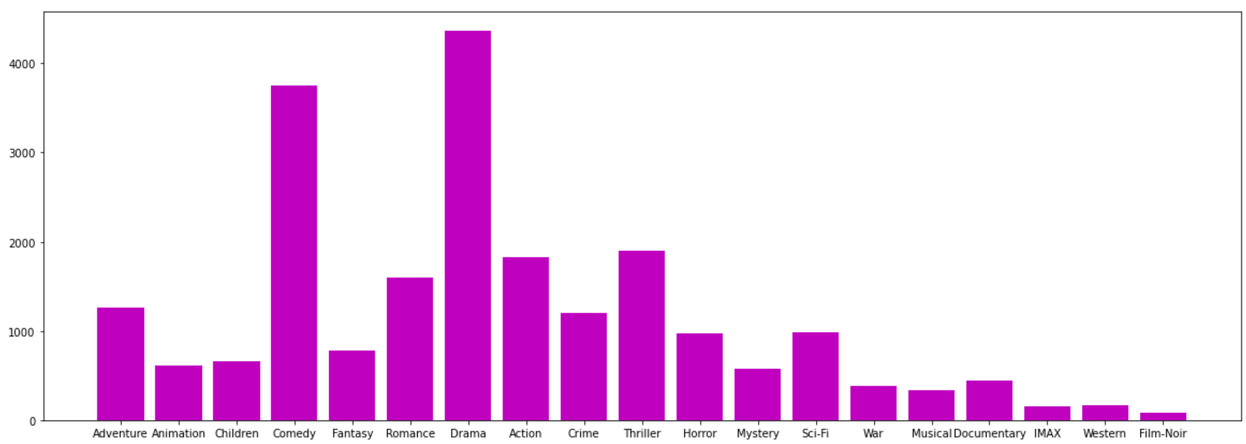
2-Data Analysis:

2-1-Exploratory Data Analysis (EDA):

2-1-1 Most popular genre

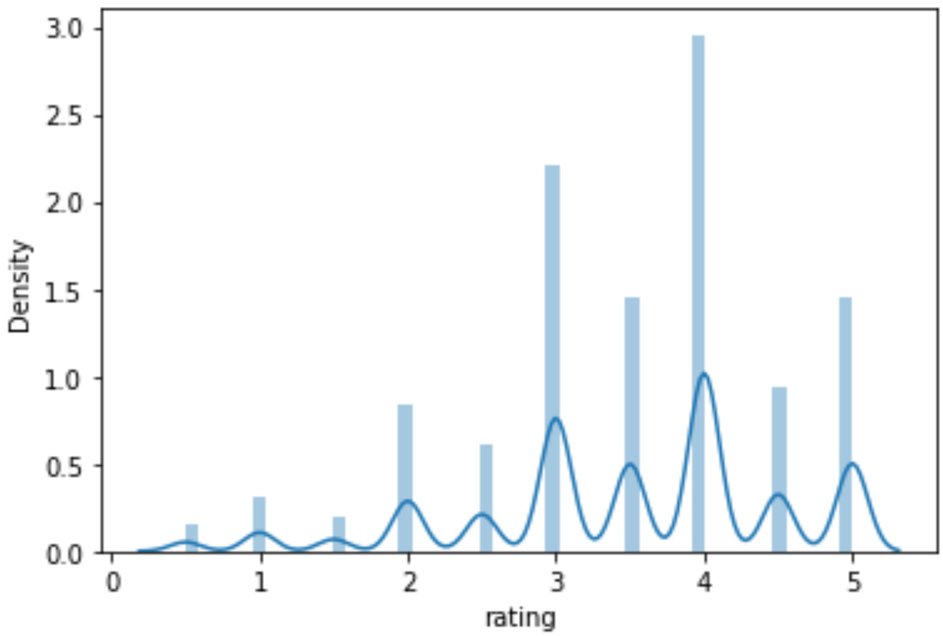
First, we have done some exploratory data analysis to have a better understanding of data.

The following charts shows the most popular genre:



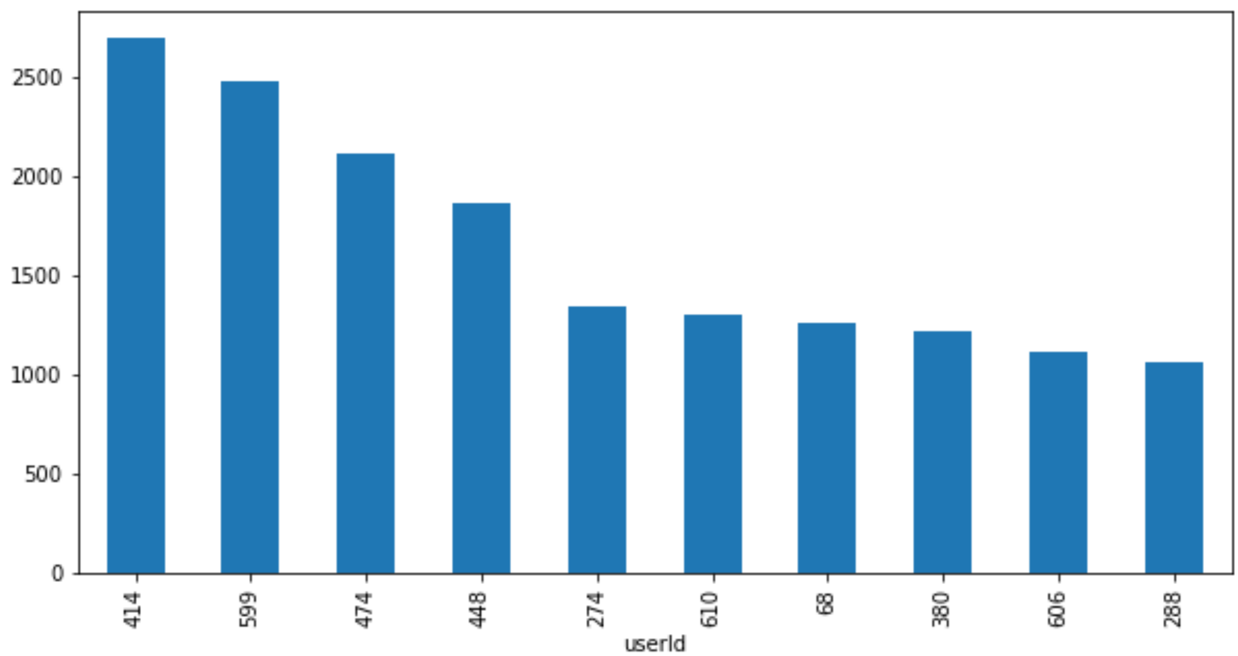
Here we can see that the most popular genre are Drama and Comedy and the least popular is IMAX and Film-Noir.

2-2-2 Distribution of user ratings



The plot shows most of the movies have been rated 4 (out of 5)

2-2-3 Top 10 users who have rated most of the movies



3-Methods

3-1-Content-Based Filtering:

Content-based filtering, also referred to as cognitive filtering, recommends items based on a comparison between the content of the items and a user profile. The content of each item is represented as a set of descriptors or terms, typically the words that occur in a document. The user profile is represented with the same terms and built up by analyzing the content of items which have been seen by the user. In the context of movie recommendation system, we will consider genre as term of movie to see similarity. The concepts of Term Frequency (TF) and Inverse Document Frequency (IDF) are used in information retrieval systems and content based filtering mechanisms (such as a content based recommender). They are used to determine the relative importance of a document / article / news item /movie etc.

We have considered genres as an important parameter to recommend user the movie he watches based on genres of movie user has already watched. To measure the distance or similarity between two movies, we can use various distance measures. Here, we have used Cosine Similarity.

In the codes, we have two functions, which recommend movies based on this method. *get_recommendations_based_on_genres* takes a movie name as input and provide two similar movies for the user. For example for the movie "*Father of the Bride Part II (1995)*" the function proposes "*Four Rooms (1995)* "and "*Ace Ventura: When Nature Calls (1995)* ".

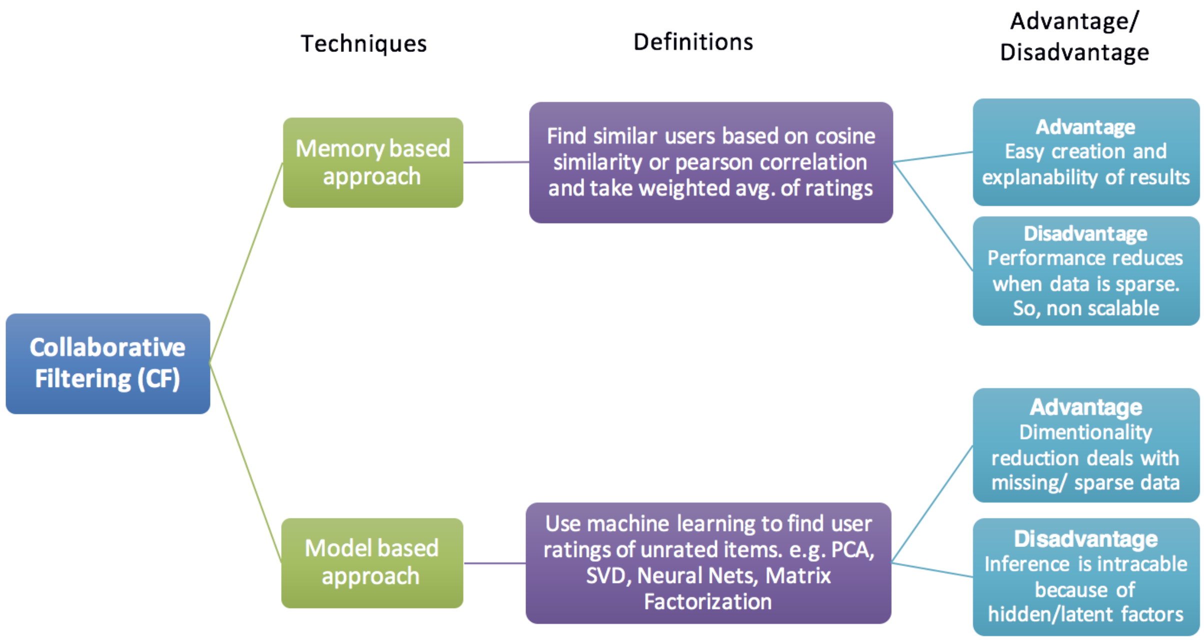
get_recommendation_content_model gets the user id as input and provide recommendations based on the movies the user have watched before. This function first finds the movies user have already rated and provide recommendations based on he genre of those movies by using the previous function.

Evaluation:

As the movie recommended by content-based filtering is based on genres, for evaluating model we have clustered movie based on groups of genres with the KNN classifier. The classifier label returned by KNN classifier to the movies recommended by the content based filtering will compare to the classifier label of the movie on which model recommends, the hit and error be calculated accordingly to measure accuracy. The following is the result of evaluation:

Hit:0.9325087251077807
Fault:0.06749127489221926

3-2- Collaborative Filtering:



The above chart shows different type of collaborative filtering.

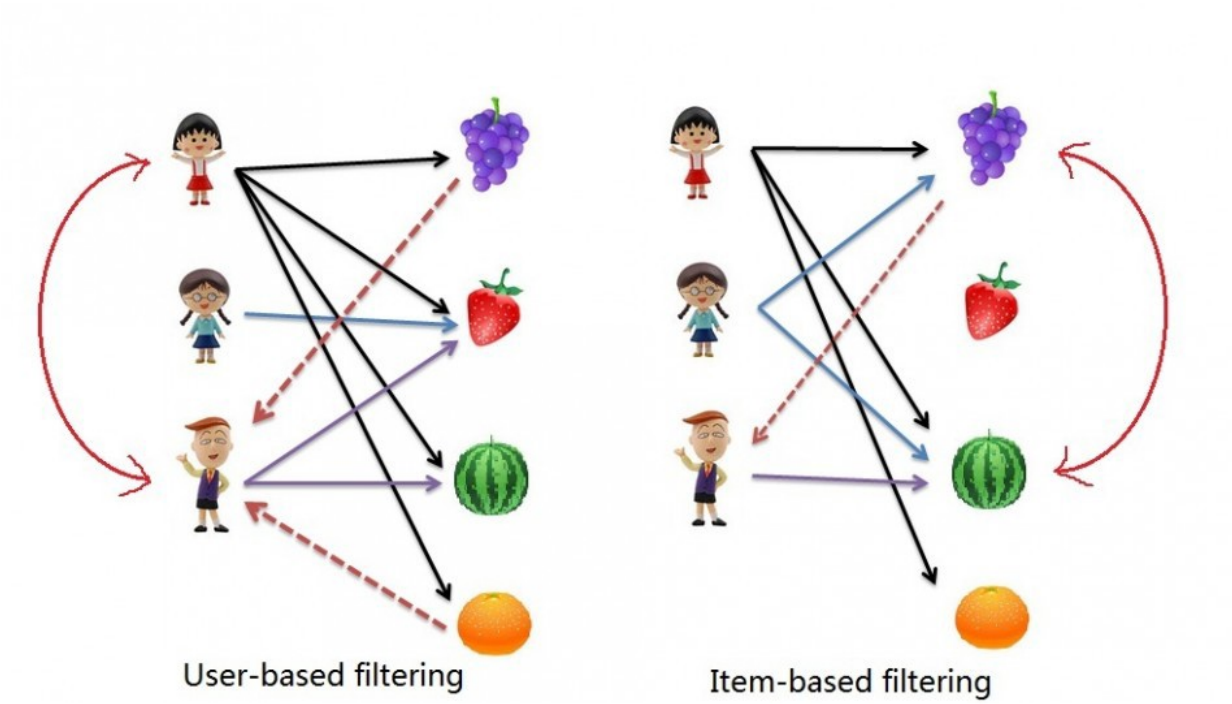
3-2-1 Memory-Based Collaborative Filtering

Memory-based algorithms approach the collaborative filtering problem by using the entire database. Here we draw the similarity between User-User or Item-Item by finding out the distance between them. Distance is calculated by referring to some numeric value. For use case of movie recommendation, rating can be considered as a factor to calculate the distance.

3-2-1-1-User-Item Collaborative filtering- In User-item filtering the distance between the items is calculated based on the users ratings (or likes, or whatever metric applies). When coming up with recommendations for a particular user, we look at the user that is closest to the chosen user and then suggest items, which liked by similar user but not watched by the chosen user. So, if you have watched and liked a certain number of movies we can look at other users who liked those same movies and recommend one that they also liked but which you might not have seen yet. Here the distance between users is calculated to infer the similarity between them. For a movie recommendation system, rating given by each user to the movie can be considered to create a vector for each user and then by using distance measures such as Euclidean Distance, Cosine distance, Pearson correlation or Jaccard Similarity we can find similarity between them.

3-2-1-2-Item-Item Collaborative Filtering- Item-item collaborative filtering was originally developed by Amazon and draws inferences about the relationship between different items based on which items are purchased together. Here the distance between items is calculated to infer the similarity between them. For a movie recommendation system, rating given by each user can be considered to create a vector for each movie and then by using distance measures such as Euclidean

Distance, Cosine distance, Pearson correlation or Jaccard Similarity we can find similarity between them. We have used just Cosine distance in this project.



In either scenario, we build a similarity matrix. First, we create a Pivot Table, rows are movies, the columns are users, and the content of table is the rating of the users to different movies.
To create the similarity matrix for user-item collaborative filtering, the distance between each two columns will be calculated using Cosine similarity so we will have a matrix of 610*610 (in our dataset we have 610 different users). For Item-Item similarity matrix, the distance between each two rows will be taken using Cosine similarity so we will have a matrix of 9724*9724. (The total number of movies in our dataset is 9724).

There are two functions in the codes, which do recommendation based on these methods. *recommendedMoviesAsperItemSimilarity* takes user id as input, finds the highest rated movie of that user and passes the movie name to *item_similarity* function. This function adds similarity scores column to movies data frame by using similarity matrix. The main function uses this data frame to provide recommendations where similarity score is more than 045 (out of 1). Here is the sample results for user id = 60:

Recommended movies,:
[510 Silence of the Lambs, The (1991)

Name: title, dtype: object, 659   Godfather, The (1972)
Name: title, dtype: object, 224   Star Wars: Episode IV - A New Hope (1977)
Name: title, dtype: object, 257   Pulp Fiction (1994)
Name: title, dtype: object, 2077   Iron Giant, The (1999)
Name: title, dtype: object, 43   Seven (a.k.a. Se7en) (1995)
Name: title, dtype: object, 507   Terminator 2: Judgment Day (1991)
Name: title, dtype: object, 315   Four Weddings and a Funeral (1994)
Name: title, dtype: object, 123   Apollo 13 (1995)
Name: title, dtype: object, 97   Braveheart (1995)
Name: title, dtype: object]

getRecommendedMoviesAsperUserSimilarity takes the user id as input (like previous function). Then finds similar users from the data frame of similar users. Then find the movies which have been watched by the similar user and have not been watched by user we are going recommend movies. Then movies will be sorted based on ratings and finally we propose top 10 high rated movies. Here is the result for user id = 60:
Movies you should watch are:

Movies you should watch are:
[138   Die Hard: With a Vengeance (1995)
Name: title, dtype: object, 398   Fugitive, The (1993)
Name: title, dtype: object, 31   Twelve Monkeys (a.k.a. 12 Monkeys) (1995)
Name: title, dtype: object, 249   Natural Born Killers (1994)
Name: title, dtype: object, 217   Interview with the Vampire: The Vampire Chroni...
Name: title, dtype: object, 314   Forrest Gump (1994)
Name: title, dtype: object, 510   Silence of the Lambs, The (1991)
Name: title, dtype: object, 461   Schindler's List (1993)
Name: title, dtype: object, 124   Rob Roy (1995)
Name: title, dtype: object, 97   Braveheart (1995)

Item-Item and User-Item collaborative filtering recommended different movies for same user. There is just two common movies in both recommendations.

3-2-1-3-Model Evaluation:

These two method are highly depend on the similarity matrix. To evaluate the model, we used the data frame of similar users (*df_similar_user* data frame in the codes) , which has been created, by using the similarity matrix. We create a function *get_user_similar_movies*, which takes a user id as input. Then finds the similar user from the similar users data frame (df_similar_user). Then finds the movies, which have been rated by both users and compare the ratings. Here is the output of this function for user id = 587:

title_x	userId_x	rating_x	userId_y	rating_y
Forrest Gump (1994)	587	4.0	511	4.5
Life Is Beautiful (1997)	578	5.0	511	4.5
Matrix, The (1999)	578	4.0	511	5.0

The most similar user to the user 578 is the user 511. We can see that these two users gave similar scores to same movies.

3-2-1-4 Cons of two methods

Challenges with User similarity

- The challenge with calculating user similarity is the user need to have some prior purchases and should have rated them.
- This recommendation technique does not work for new users.
- The system need to wait until the user make some purchases and rates them. Only then similar users can be found and recommendations can be made. This is called cold start problem.

Memory-based collaborative filtering approaches that compute distance relationships between items or users have these two major issues:

- It does not scale particularly well to massive datasets, especially for real-time recommendations based on user behavior similarities—which takes many computations.
- Ratings matrices may be overfitting to noisy representations of user tastes and preferences. When we use distance based "neighborhood" approaches on raw data, we match to sparse low-level details that we assume represent the user's preference vector instead of the vector itself.

3-2-2-Model-Based Collaborative Filtering:

Model-based Collaborative Filtering is based on matrix factorization (MF) which has received greater exposure, mainly as an unsupervised learning method for latent variable decomposition and dimensionality reduction. Matrix factorization is widely used for recommender systems where it can deal better with scalability and sparsity than Memory-based CF:

- The goal of MF is to learn the latent preferences of users and the latent attributes of items from known ratings (learn features that describe the characteristics of ratings) to then predict the unknown ratings through the dot product of the latent features of users and items.
- When you have a very sparse matrix, with a lot of dimensions, by doing matrix factorization, you can restructure the user-item matrix into low-rank structure, and you can represent the matrix by the multiplication of two low-rank matrices, where the rows contain the latent vector. This transformer performs linear dimensionality reduction by means of truncated singular value decomposition (SVD). Contrary to PCA, this estimator does not center the data before computing the singular value decomposition. This means it can work with sparse matrices efficiently.

- You fit this matrix to approximate your original matrix, as closely as possible, by multiplying the low-rank matrices together, which fills in the entries missing in the original matrix.

For example, let us check the sparsity of the ratings dataset:
The sparsity level of MovieLens100K dataset is **98.3%**

A well-known matrix factorization method is Singular Value Decomposition (SVD). At a high level, SVD is an algorithm that decomposes a matrix into the best lower rank (i.e. smaller/simpler) approximation of the original matrix. Mathematically, it decomposes A into a two unitary matrices and a diagonal matrix:

$$A = USV^T$$

Where:
A is an m × n matrix
U is an m × r orthogonal (Left Singular Matrix)
Matrix S is an r × r diagonal (Sigma is the diagonal matrix of singular values essentially weights/strengths of each concept)
Matrix V transpose is an r × n orthogonal matrix (is the right singluar vectors (movie "features" matrix)).

3-2-2-1-Setting Up SVD

Scipy and Numpy both have functions to do the singular value decomposition. We are going to use the Scipy function svds because it let's us choose how many latent factors we want to use to approximate the original ratings matrix (instead of having to truncate it after). The number latent factors in this project is 50. First, we create a Pivot Table, rows are movies, the columns are users, and the content of table is the rating of the users to different movies. Then decompose this matrix to three matrixes as explained above. Then we make predictions by using these three matrices. The final prediction matrix has 610 * 9724 dimensions without any zero value in this matrix that means we have ratings of every user for all movies.

The function *recommend_movies* in the codes use this matrix for prediction and provide 20 recommendations for the selected user.

Here is the result for user id = 150:

User 150 has already rated 26 movies.

```
In [204]: already_rated.head(20)
Out[204]:
    userId  movieId  rating  timestamp                                 title                                   genres
25     150     1356     5.0  854203229          Star Trek: First Contact (1996)      Action|Adventure|Sci-Fi|Thriller
5      150       32     5.0  854203071   Twelve Monkeys (a.k.a. 12 Monkeys) (1995)              Mystery|Sci-Fi|Thriller
12     150      141     5.0  854203072                    Birdcage, The (1996)                               Comedy
17     150      648     4.0  854203072              Mission: Impossible (1996)     Action|Adventure|Mystery|Thriller
2      150        6     4.0  854203123                             Heat (1995)                Action|Crime|Thriller
4      150       25     4.0  854203072                Leaving Las Vegas (1995)                        Drama|Romance
6      150       36     4.0  854203123                Dead Man Walking (1995)                          Crime|Drama
7      150       52     4.0  854203163                 Mighty Aphrodite (1995)                 Comedy|Drama|Romance
23     150      805     4.0  854203230                  Time to Kill, A (1996)                       Drama|Thriller
20     150      780     4.0  854203071       Independence Day (a.k.a. ID4) (1996)    Action|Adventure|Sci-Fi|Thriller
19     150      733     4.0  854203123                        Rock, The (1996)            Action|Adventure|Thriller
15     150      608     4.0  854203123                            Fargo (1996)          Comedy|Crime|Drama|Thriller
24     150     1073     3.0  854203163  Willy Wonka & the Chocolate Factory (1971)  Children|Comedy|Fantasy|Musical
22     150      786     3.0  854203163                           Eraser (1996)                Action|Drama|Thriller
21     150      784     3.0  854203163                   Cable Guy, The (1996)                      Comedy|Thriller
18     150      653     3.0  854203163                      Dragonheart (1996)             Action|Adventure|Fantasy
0      150        3     3.0  854203124                 Grumpier Old Men (1995)                       Comedy|Romance
16     150      628     3.0  854203229                      Primal Fear (1996)        Crime|Drama|Mystery|Thriller
14     150      494     3.0  854203124               Executive Decision (1996)            Action|Adventure|Thriller
1      150        5     3.0  854203124       Father of the Bride Part II (1995)                               Comedy
```

```
In [205]: predictions
Out[205]:
      movieId                                       title                                   genres
574       736                              Twister (1996)         Action|Adventure|Romance|Thriller
0           1                            Toy Story (1995)  Adventure|Animation|Children|Comedy|Fantasy
211       260          Star Wars: Episode IV - A New Hope (1977)              Action|Adventure|Sci-Fi
607       802                            Phenomenon (1996)                       Drama|Romance
12         17                  Sense and Sensibility (1995)                       Drama|Romance
87        112         Rumble in the Bronx (Hont faan kui) (1995)         Action|Adventure|Comedy|Crime
558       708                Truth About Cats & Dogs, The (1996)                      Comedy|Romance
599       788                     Nutty Professor, The (1996)         Comedy|Fantasy|Romance|Sci-Fi
886      1210  Star Wars: Episode VI - Return of the Jedi (1983)              Action|Adventure|Sci-Fi
634       852                                Tin Cup (1996)                Comedy|Drama|Romance
565       719                           Multiplicity (1996)                              Comedy
1047     1393                          Jerry Maguire (1996)                       Drama|Romance
80        104                          Happy Gilmore (1996)                              Comedy
9          14                                  Nixon (1995)                               Drama
532       661                 James and the Giant Peach (1996)  Adventure|Animation|Children|Fantasy|Musical
587       762                             Striptease (1996)                        Comedy|Crime
4           9                           Sudden Death (1995)                              Action
621       832                                 Ransom (1996)                      Crime|Thriller
523       637                             Sgt. Bilko (1996)                              Comedy
103       140                     Up Close and Personal (1996)                       Drama|Romance
```

These look like pretty good recommendations. It's good to see that, although we didn't actually use the genres of the movie as a feature, the truncated matrix factorization features "picked up" on the underlying tastes and preferences of the user. We have recommended some Action, Adventure, Romance, Thriller movies - all of which were genres of some of this user's top rated movies.

Model Evaluation

Instead of doing manually like the last time, we will use the Surprise library that provided various ready-to-use powerful prediction algorithms including (SVD) to evaluate its RMSE (Root Mean

Squared Error) on the MovieLens dataset. It is a Python scikit building and analyzing recommender systems.
The *svd.fit()* from Surprise tries to find best prediction matrix by continuously creating prediction matrix, doing predictions and comparing the prediction by actual ratings.

Here is the result of evaluating the svd model using cross validation, where the number of folds are 5:

{'test_rmse': array([0.873122 , 0.87879761, 0.87330569, 0.8749066 , 0.86522131]),

'test_mae': array([0.67374388, 0.67554549, 0.67154232, 0.6704354 , 0.66242437]),

We get a mean Root Mean Square Error of 0.87 which is pretty good.

After training our model, we can use it for predict the rating of a movie for a user which has not seen that movie before. Here is the result for user id = 150.

This user has already rated these movies:

Out[226]:

	userId	movieId	rating	timestamp
22277	150	3	3.0	854203124
22278	150	5	3.0	854203124
22279	150	6	4.0	854203123
22280	150	7	3.0	854203124
22281	150	25	4.0	854203072
22282	150	32	5.0	854203071
22283	150	36	4.0	854203123
22284	150	52	4.0	854203163
22285	150	58	3.0	854203163
22286	150	62	3.0	854203072
22287	150	79	3.0	854203229
22288	150	95	3.0	854203072
22289	150	141	5.0	854203072
22290	150	376	3.0	854203124
22291	150	494	3.0	854203124
22292	150	608	4.0	854203123
22293	150	628	3.0	854203229
22294	150	648	4.0	854203072
22295	150	653	3.0	854203163
22296	150	733	4.0	854203123
22297	150	780	4.0	854203071
22298	150	784	3.0	854203163
22299	150	786	3.0	854203163
22300	150	805	4.0	854203230
22301	150	1073	3.0	854203163
22302	150	1356	5.0	854203229

First, we can see how model predicts the rating for the movies that has already been rated. For example:

svd.predict(150, 6)

Out[233]: Prediction(uid=150, iid=6, r_ui=None, **est=4.047085359178342**, details={'was_impossible': False})

We can see that the user rating is 4 for this movie and or model also rated 4.04 which is very good. For other movies for example 1994 this user rating will be:

svd.predict(150, 1994)

Out[235]: Prediction(uid=150, iid=1994, r_ui=None, **est=3.5756551872567215**, details={'was_impossible': False})

3-2-2-2 SVD++:

To build a robust recommender system, we need to develop models which factor in both explicit and implicit user feedback. For our Movielens dataset, a less obvious kind of implicit data does exist. The dataset does not only tell us the rating values, but also which movies users rate, regardless of how they rated these movies. In other words, a user implicitly tells us about her preferences by choosing to voice her opinion and vote a (high or low) rating. This reduces the ratings matrix into a binary matrix, where “1” stands for “rated”, and “0” for “not rated”. Admittedly, this binary data is not as vast and independent as other sources of implicit feedback could be. Nonetheless, we have found that incorporating this kind of implicit data – which inherently exist in every rating based recommender system – significantly improves prediction accuracy. SVD++ factors in this implicit feedback and gives better accuracy as shown below.

SVDpp : Test Set

RMSE: 0.9342

Out[236]: 0.9342212416848242

3-2-3- Evaluating User-User and User-Item collaborative filtering by SVD:

In section 3-2-1-3, we have evaluated these models but we did not compare them because we did not have any tools to compare them. We just saw if our model could find similar users appropriately. Now, we can use svd to compare these models. The idea is this: first, we get recommendations based on these two models (We have increased the number of recommendations from 10 to 20 to have better comparison). Then, we predict how the user will rate those movies using svd. If the svd rating is greater than 3, then we consider that recommended movie as a success. Finally, we calculate the ration of number of movies rated 3 or more over the total number of predictions. This ratio was named HitRatio:

$$HitRatio = \frac{recommendedMoviesRating > 3}{totalNumberofRecommendedMovies}$$

For example for user id = 50 we have:

Hit ratio of User-user collaborative filtering **0.5**

Hit ratio of Item-Item collaborative filtering **0.7**

For user id = 78:

Hit ratio of User-user collaborative filtering **0.4**

Hit ratio of Item-Item collaborative filtering **1.0**

User id = 414 (the user who rated movies more than the others):

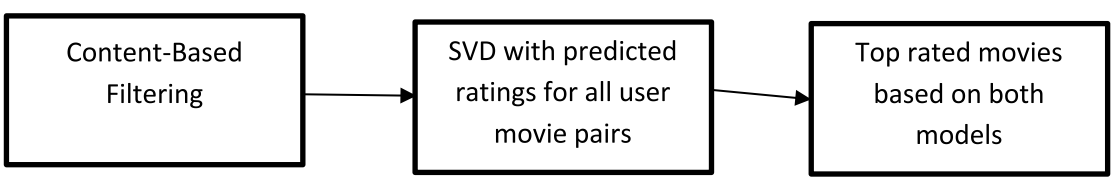
Hit ratio of User-user collaborative filtering **0.8**

Hit ratio of Item-Item collaborative filtering **1.0**

We can see that Item-Item does a better job compare to User-User collaborative filtering. We expected this result because the similarity matrix dimension for Item-Item is 9724*9724 while similarity matrix dimension for User-User is 610*610 which means for Item-Item collaborative filtering we have more data to make recommendations.

3-3- The Hybrid Model

To overcome shortcomings of an individual model, we have used a hybrid model wherein we stack two different models. The resultant hybrid model gives higher accuracy and more relevant results. The movie recommended by Content-Based Filtering is passed to SVD model which predicts the rating the user will give to the recommended movie. Finally, we return the movies in the descending order of SVD predicted ratings.



Here is the result for user id = 414:

Out[292]:

	movieId	title	genres	svd_rating
224	260	Star Wars: Episode IV - A New Hope (1977)	Action|Adventure|Sci-Fi	4.956404
3562	4878	Donnie Darko (2001)	Drama|Mystery|Sci-Fi|Thriller	4.905506
909	1208	Apocalypse Now (1979)	Action|Drama|War	4.873420
906	1204	Lawrence of Arabia (1962)	Adventure|Drama|War	4.817527
2259	2997	Being John Malkovich (1999)	Comedy|Drama|Fantasy	4.794175
520	608	Fargo (1996)	Comedy|Crime|Drama|Thriller	4.780281
585	720	Wallace & Gromit: The Best of Aardman Animatio...	Adventure|Animation|Comedy	4.735571
868	1148	Wallace & Gromit: The Wrong Trousers (1993)	Animation|Children|Comedy|Crime	4.644146
969	1270	Back to the Future (1985)	Adventure|Comedy|Sci-Fi	4.598936
1218	1617	L.A. Confidential (1997)	Crime|Film-Noir|Mystery|Thriller	4.551375
2355	3114	Toy Story 2 (1999)	Adventure|Animation|Children|Comedy|Fantasy	4.518675

3-4-Clustering with K-Means

3-4-1-Data Analysis

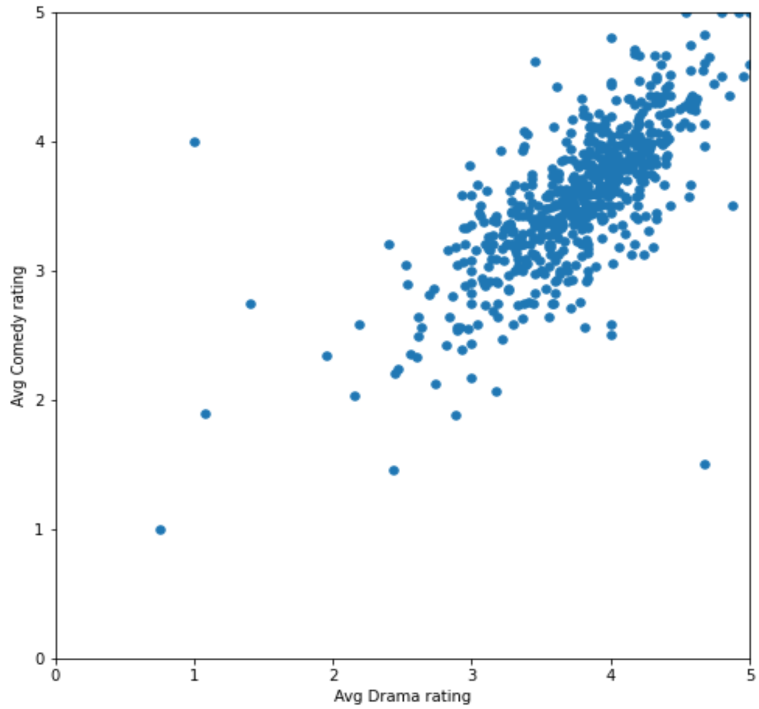
We also defined some definitions to Scatterplot, draw clusters including calculating cluster error. It also includes functions such as heatmaps in python, get functions to get user data and their movie ratings.

Next step is to implement average rating dataframe for our most common genres, “Drama” and ‘Comedy’. From this we can notice the number of records which are 610, which are too high to do clustering on them.

Number of records: 610

	avg_drama_rating	avg_comedy_rating
1	4.53	4.28
2	3.88	4.00
3	0.75	1.00
4	3.48	3.51
5	3.80	3.47

Scatterplot can be drawn to show the average value distribution. This data is not clean, and we clean it using by biasing our database.



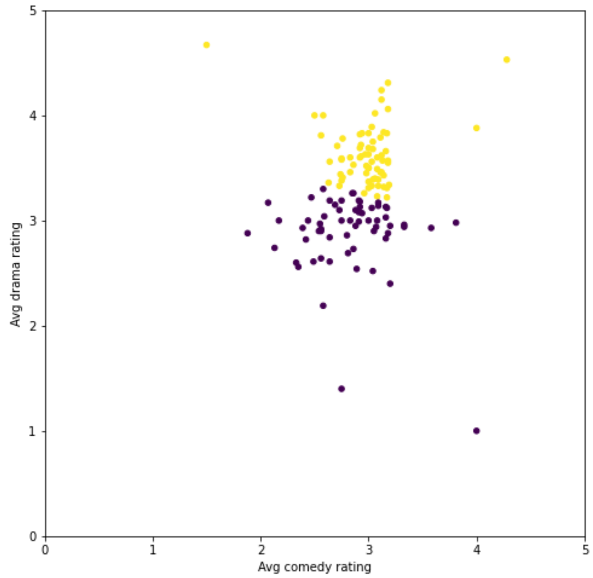
Now, we bias our dataset a little by removing people who like both comedy and drama, just so that our clusters tend to define them as liking one genre more than the other. Doing this also cleans our data and make it readable otherwise the dataset is too large. From this we can also notice that number of records have decreased significantly to 131. Which is more doable with our clustering methods.

```
Number of records:  131
```

	index	avg_drama_rating	avg_comedy_rating
0	7	3.13	3.16
1	14	3.71	2.71
2	19	2.61	2.64
3	21	2.95	3.20
4	22	2.61	2.49

Bias is created, we can break down the cluster into 2 groups using K-Means. We import KMeans library and choose cluster value to be 2. We can derive an error for cluster=2

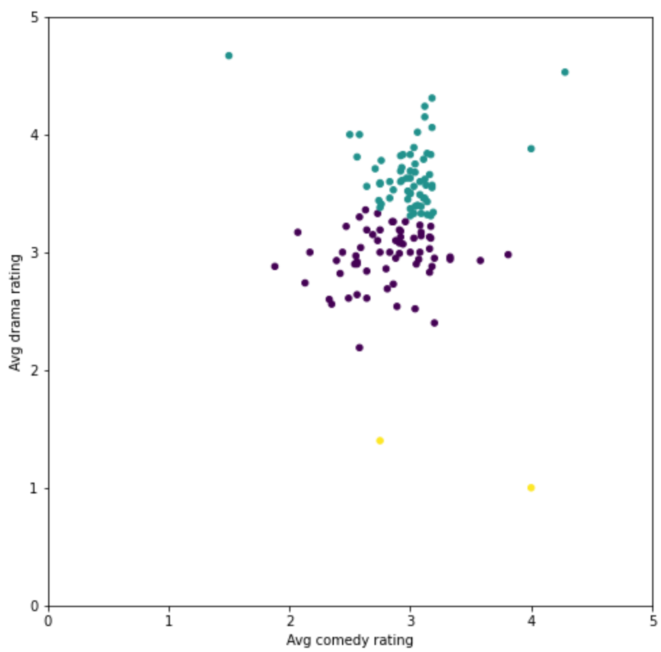
0.385771397400849



We can also increase the clusters to 3. So now we have 3 different groups. The groups are:

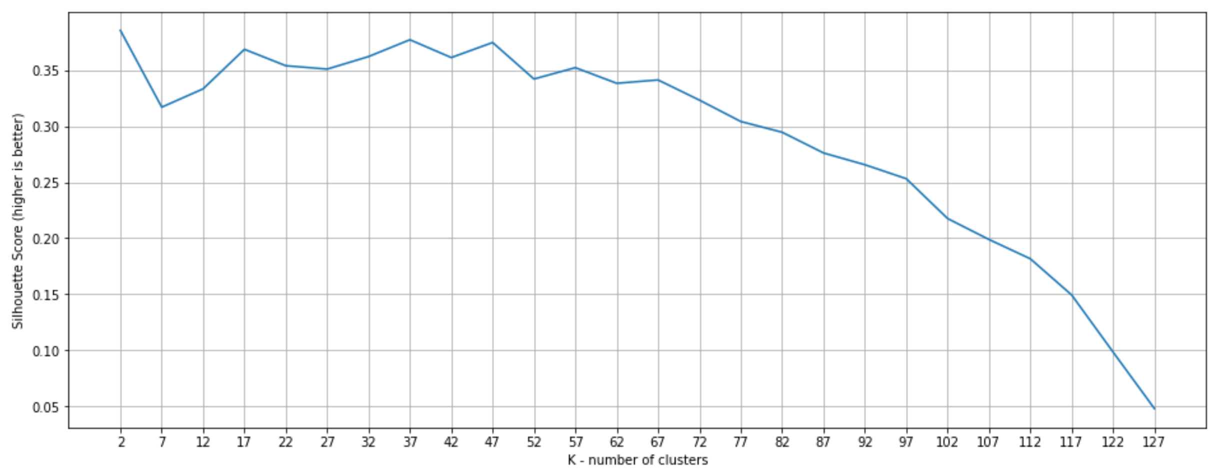
- People who like drama but not comedy
- people who like comedy but not drama
- people who like both comedy and drama

0.39436177541431655

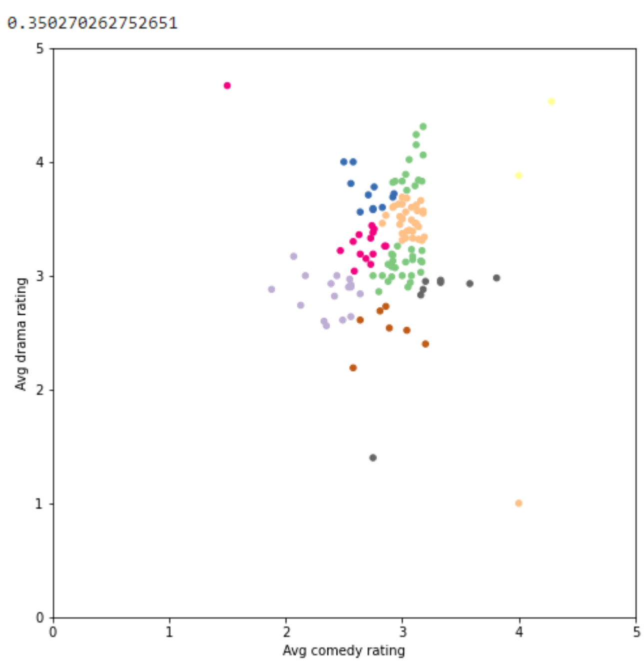


We will be using "the elbow method" to calculate K. The elbow method works by plotting the ascending values of k versus the total error calculated using that k. Error is calculated using squared error method. That gives us a list of all possible k-values and their respective error.

Also, by plotting the possible k values, we can notice the elbow curve occurs at clusters = 7. Furthermore at 12, 37 and so on. But it is hard to visualize at higher number of clusters and we also notice that after 47 clusters increasing the number of clusters (k) beyond that range starts to result in worse clusters.



We can plot cluster=7. With an error of 0.34 which is very reasonable for error margin. The way it is implemented is every like other cluster implementation just changing the number of clusters.



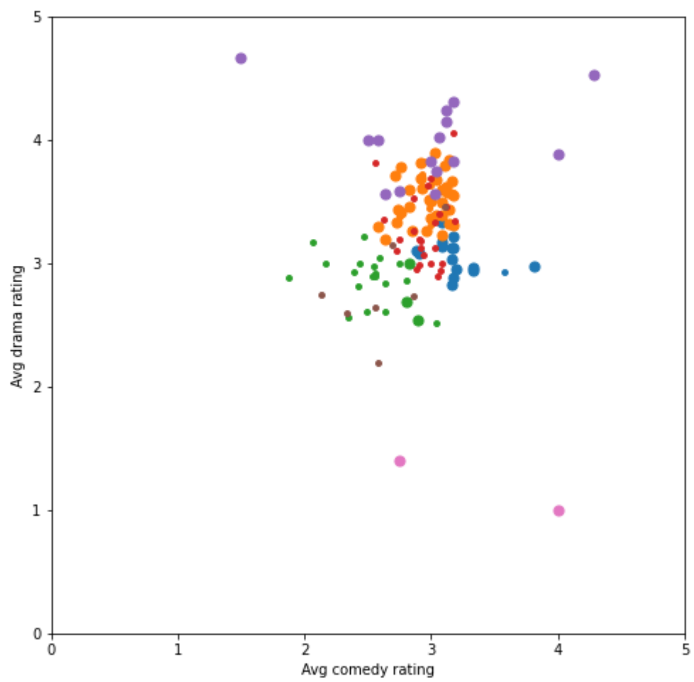
We now added another 2 genres, for us we chose Action and Thriller genre to make clustering more visual and bit more complex. We created a new DataFrame for those 4 genres, biased data frame and then converting that to list for easy transformation.

```
Number of records:  130
```

	index	avg_drama_rating	avg_comedy_rating	avg_action_rating	avg_thriller_rating
0	7	3.13	3.16	3.26	3.43
1	14	3.71	2.71	3.33	3.46
2	19	2.61	2.64	2.73	2.55
3	21	2.95	3.20	3.46	3.55
4	22	2.61	2.49	2.78	2.78

Plotting those 4 genres with clusters=7. We can see a lot of overlapping occurring. the size of the dot to roughly code the 'action' rating (large dot for avg ratings over than 3, small dot otherwise).

We can start seeing the added genre is changing how the users are clustered. The more data we give to k-means, the more similar the tastes of the people in each group would be. Unfortunately, we cannot visualize more than this using clustering technique as it limits to 2D.



3-4-2-Prediction Based on User ratings

To do that, we will shape the dataset in the form of userId vs user rating for each movie.

dataset dimensions: (610, 9719)

example:

title	'71 (2014)	'Hellboy': The Seeds of Creation (2004)	'Round Midnight (1986)	'Salem's Lot (2004)	'Til There Was You (1997)	'Tis the Season for Love (2015)	'burbs, The (1989)	'night Mother (1986)	(500) Days of Summer (2009)
userId									
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

We can notice the presence of NaN. This occurs due to the fact that most users have not rated and watched most movies. Datasets like this are called "sparse" because only a small number of cells have values.

We can overcome this by sorting our data by most rated movies and users who have rated the most movies. If we do that, our data will look like,

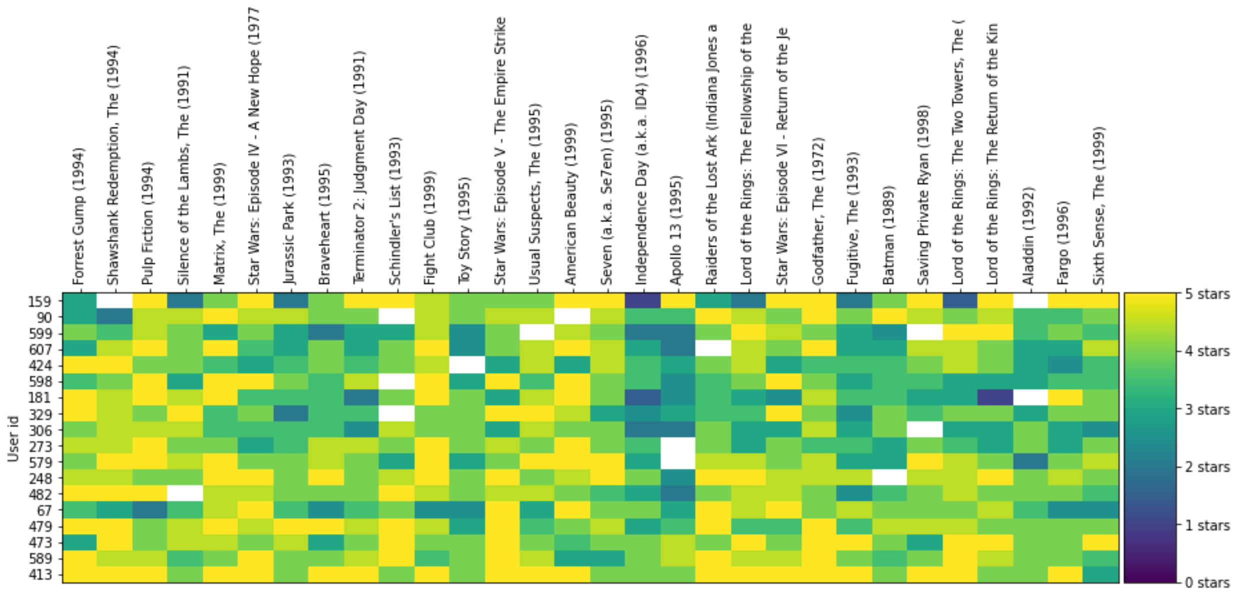
dataset dimensions: (18, 30)

title	Forrest Gump (1994)	Shawshank Redemption, The (1994)	Pulp Fiction (1994)	Silence of the Lambs, The (1991)	Matrix, The (1999)	Star Wars: Episode IV - A New Hope (1977)	Jurassic Park (1993)	Braveheart (1995)	Terminator 2: Judgment Day (1991)	Schindler's List (1993)	...	Star Wars: Episode VI - Return of the Jedi (1983)	Godfather, The (1972)	Fugitive, The (1993)	Batman (1989)	Saving Private Ryan (1998)	Lord of the Rings: The Two Towers, The (2002)	Lord of the Rings: The Return of the King, The (2003)	Aladdin (1992)
413	5.0	5.0	5.0	4.0	5.0	5.0	4.0	5.0	5.0	4.0	...	5.0	5.0	5.0	4.0	5.0	5.0	4.0	4.0
589	5.0	4.5	4.5	3.5	4.0	5.0	4.0	4.0	4.5	5.0	...	4.5	5.0	4.0	3.5	4.0	5.0	4.5	4.0
473	3.0	5.0	4.0	4.5	4.5	4.0	4.5	3.0	4.0	5.0	...	4.0	5.0	5.0	4.0	3.0	5.0	5.0	4.0
479	5.0	5.0	4.0	4.5	5.0	4.5	5.0	5.0	4.5	5.0	...	3.5	5.0	3.5	4.5	4.5	4.5	4.0	4.0
67	3.5	3.0	2.0	3.5	4.5	5.0	3.5	2.5	3.5	4.0	...	5.0	4.0	4.5	4.0	4.0	4.0	4.5	3.5

5 rows × 30 columns

We visualized our data using a heatmap function, which made visualizing a lot easier to understand.

```
movies_heatmap(most_rated_movies_users_selection)
```



In heat map, each column is a movie, each row is a user. The color gradient of the cell shows user ratings as shown on right.

White cells show respective user did not rate that movie. This is an issue you will come across when clustering in real life. Our datasets can often be sparse and not have a value in each cell of the dataset. This makes it less straightforward to cluster users directly by their movie ratings as k-means is based on all the values in the cells.

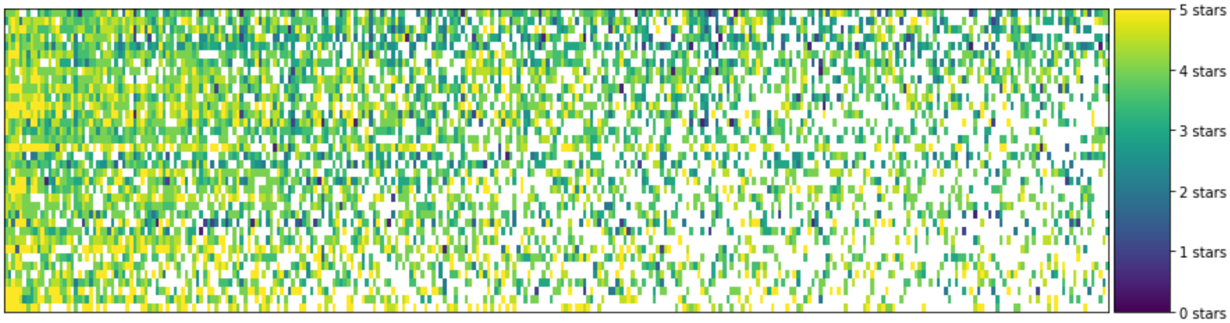
We will only use ratings for 1000 movies (out of the 9000+ available in the dataset).

We will be converting our dataframe to sparse matrix to SparseDataFrame using SciPi library. We must do this to overcome datasets with missing values like our dataset.

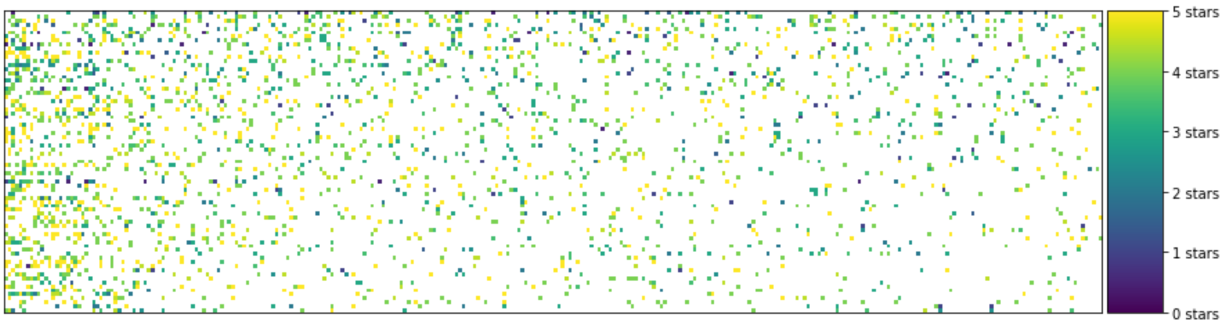
To use sparse matrix, we had to downgrade our pandas version on Google CoLab. It was done in starting of the code.

We can specify k, the number of clusters. We chose k up to 15 due to elbow curve. We can plot each value for the clusters, but we don't. We only use some examples to show difference in heatmaps for different numbers of clusters.

Below we have clusters=3



Now we can visualize for clusters=7



We can notice that,

- Some clusters are sparser than others, containing people who probably watch and rate less movies than in other clusters.
- Some clusters are mostly yellow and bring together people who really love a certain group of movies. Other clusters are mostly green or navy blue.

Now predicting how users will rate a certain they have not rated in our cluster. Since in our clusters users have similar taste, we can predict how the user will rate the movie using the average of votes in that cluster. In our case we choose 'Fight Club (1999)'

```python
cluster.fillna('').head()
```

	Matrix, The (1999)	Shawshank Redemption, The (1994)	Forrest Gump (1994)	Silence of the Lambs, The (1991)	Star Wars: Episode IV - A New Hope (1977)	American Beauty (1999)	Schindler's List (1993)	Fight Club (1999)
28			0.5					
32			4					
155		4		4	4	5		
39			4	2			2	
77						5		

5 rows × 300 columns

```python
# Pick a movie from the table above since we're looking at a subset of dataset
movie_name = "Fight Club (1999)"

print('Predicted rating:',cluster[movie_name].mean())
```

```
Predicted rating: 3.9318181818181817
```

We have used k-means to cluster users according to their ratings that lead us to clusters of users with similar ratings and thus generally a similar taste in movies. Now basing on this, when one user did not have a rating for a certain movie, we can average the ratings of all the other users in the cluster, and that was our guess to how this one user would like the movie.

Using this logic, we can calculate the average score in this cluster for every movie.

We can also calculate average ratings of the movies in that cluster by all those users.

```python
cluster.mean().head(20)
```

```
Batman (1989)                          3.257576
Pulp Fiction (1994)                    3.846154
True Lies (1994)                       3.552239
Apollo 13 (1995)                       3.942623
Dances with Wolves (1990)              3.836066
Fugitive, The (1993)                   4.204918
Forrest Gump (1994)                    4.377193
Braveheart (1995)                      4.327586
Clear and Present Danger (1994)        3.655172
Die Hard: With a Vengeance (1995)      3.561404
Batman Forever (1995)                  3.140351
Shawshank Redemption, The (1994)       4.413793
Jurassic Park (1993)                   3.903509
Crimson Tide (1995)                    3.807018
Ace Ventura: Pet Detective (1994)      2.964912
Aladdin (1992)                         3.767857
```

We can also show them recommendations that are appropriate to their taste. The formula for these recommendations is to select the cluster's highest-rated movies that the user **"did not"** rate yet.

For this case, we can derive total number of movies recommended, 240 and we display top 30 of those to the user =32 from the table above.

```
Total recommended movies are:  240


Top 30 movies are:

Producers, The (1968)                                                       5.000000
L.A. Confidential (1997)                                                    4.800000
Jaws (1975)                                                                 4.777778
Dark Knight Rises, The (2012)                                               4.750000
Shutter Island (2010)                                                       4.750000
Philadelphia Story, The (1940)                                              4.700000
Departed, The (2006)                                                        4.666667
Hunt for Red October, The (1990)                                            4.666667
One Flew Over the Cuckoo's Nest (1975)                                      4.653846
Godfather: Part II, The (1974)                                              4.625000
Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1964)    4.611111
Dark Knight, The (2008)                                                     4.611111
Good, the Bad and the Ugly, The (Buono, il brutto, il cattivo, Il) (1966)     4.600000
The Imitation Game (2014)                                                   4.600000
Cool Hand Luke (1967)                                                       4.562500
Batman Begins (2005)                                                        4.500000
Rear Window (1954)                                                          4.500000
Patton (1970)                                                               4.500000
Snatch (2000)                                                               4.437500
E.T. the Extra-Terrestrial (1982)                                           4.416667
Star Trek: First Contact (1996)                                             4.416667
Chinatown (1974)                                                            4.400000
It's a Wonderful Life (1946)                                                4.400000
Mask, The (1994)                                                            4.375000
Avengers, The (2012)                                                        4.375000
Rock, The (1996)                                                            4.333333
North by Northwest (1959)                                                   4.333333
Shawshank Redemption, The (1994)                                            4.321429
Apocalypse Now (1979)                                                       4.312500
Star Wars: Episode IV - A New Hope (1977)                                   4.293103
Name: 0, dtype: float64
```

4- Comparing the Models

To compare the models, we relied on the predicted ratings by the svd model. To be fair, we did not use the trained svd (i.e *svd.fit(trainset)*) to recommend movies. (i.e the svd we have used here for recommendation is a manual implementation of svd no an svd trained by machine).

The main idea is this: for a user, we get recommendations by different methods. Then by svd, we predict the ratings that user will give to proposed movies. Then we compare the ratings. The model with highest rating given by svd is the winner.

The following results are for user id = 6

First, we got recommendation from Clustering method and then used svd to predict the ratings:

```
In [340]: recomForsix[['title','genres','movieId','rating_clustering','svd_rating']].iloc[0:30].sort_values("rating_clustering", ascending
Out[340]:
```

	title	genres	movieId	rating_clustering	svd_rating
22	Casablanca (1942)	Drama|Romance	912	4.857143	4.206700
25	Citizen Kane (1941)	Drama|Mystery	923	4.857143	3.901282
100	Bowling for Columbine (2002)	Documentary	5669	4.833333	3.816139
93	Memento (2000)	Mystery|Thriller	4226	4.833333	3.843367
32	Brazil (1985)	Fantasy|Sci-Fi	1199	4.800000	3.812348
20	North by Northwest (1959)	Action|Adventure|Mystery|Romance|Thriller	908	4.714286	4.135932
66	Being John Malkovich (1999)	Comedy|Drama|Fantasy	2997	4.666667	3.730381
27	African Queen, The (1951)	Adventure|Comedy|Romance|War	969	4.666667	3.799223
6	Pulp Fiction (1994)	Comedy|Crime|Drama|Thriller	296	4.642857	3.448927
16	Fargo (1996)	Comedy|Crime|Drama|Thriller	608	4.642857	3.953347
54	Seven Samurai (Shichinin no samurai) (1954)	Action|Adventure|Drama	2019	4.625000	4.061333
5	Taxi Driver (1976)	Crime|Drama|Thriller	111	4.571429	3.917000
77	Erin Brockovich (2000)	Drama	3408	4.500000	3.897179
71	Galaxy Quest (1999)	Adventure|Comedy|Sci-Fi	3175	4.500000	3.613087
34	Psycho (1960)	Crime|Horror	1219	4.500000	4.239450
63	Monty Python's And Now for Something Completel...	Comedy	2788	4.500000	4.054787
19	Philadelphia Story, The (1940)	Comedy|Drama|Romance	898	4.500000	4.119046
92	Traffic (2000)	Crime|Drama|Thriller	4034	4.500000	4.252491
89	Best in Show (2000)	Comedy	3911	4.500000	3.884008
17	Dr. Strangelove or: How I Learned to Stop Worr...	Comedy|War	750	4.500000	4.130243
80	Do the Right Thing (1989)	Drama	3424	4.500000	4.028445
33	Goodfellas (1990)	Crime|Drama	1213	4.428571	4.232545
76	Dog Day Afternoon (1975)	Crime|Drama	3362	4.416667	4.046767
8	Shawshank Redemption, The (1994)	Crime|Drama	318	4.416667	4.544533
68	Fisher King, The (1991)	Comedy|Drama|Fantasy|Romance	3108	4.333333	3.513629
38	Great Escape, The (1963)	Action|Adventure|Drama|War	1262	4.333333	4.461743
53	Last Emperor, The (1987)	Drama	1960	4.333333	3.797396

This is sorted by clustering method. ratings. And the next one sorted based on ratings given by the svd.

```
In [339]: recomForsix[['title','genres','movieId','rating_clustering','svd_rating']].iloc[0:30].sort_values("svd_rating", ascending=False)
Out[339]:
```

	title	genres	movieId	rating_clustering	svd_rating
8	Shawshank Redemption, The (1994)	Crime|Drama	318	4.416667	4.544533
38	Great Escape, The (1963)	Action|Adventure|Drama|War	1262	4.333333	4.461743
92	Traffic (2000)	Crime|Drama|Thriller	4034	4.500000	4.252491
34	Psycho (1960)	Crime|Horror	1219	4.500000	4.239450
33	Goodfellas (1990)	Crime|Drama	1213	4.428571	4.232545
41	Cool Hand Luke (1967)	Drama	1276	4.300000	4.223183
22	Casablanca (1942)	Drama|Romance	912	4.857143	4.206700
20	North by Northwest (1959)	Action|Adventure|Mystery|Romance|Thriller	908	4.714286	4.135932
17	Dr. Strangelove or: How I Learned to Stop Worr...	Comedy|War	750	4.500000	4.130243
19	Philadelphia Story, The (1940)	Comedy|Drama|Romance	898	4.500000	4.119046
54	Seven Samurai (Shichinin no samurai) (1954)	Action|Adventure|Drama	2019	4.625000	4.061333
63	Monty Python's And Now for Something Completel...	Comedy	2788	4.500000	4.054787
76	Dog Day Afternoon (1975)	Crime|Drama	3362	4.416667	4.046767
80	Do the Right Thing (1989)	Drama	3424	4.500000	4.028445
101	Adaptation (2002)	Comedy|Drama|Romance	5902	4.250000	3.979570
16	Fargo (1996)	Comedy|Crime|Drama|Thriller	608	4.642857	3.953347
5	Taxi Driver (1976)	Crime|Drama|Thriller	111	4.571429	3.917000
25	Citizen Kane (1941)	Drama|Mystery	923	4.857143	3.901282
77	Erin Brockovich (2000)	Drama	3408	4.500000	3.897179
89	Best in Show (2000)	Comedy	3911	4.500000	3.884008
93	Memento (2000)	Mystery|Thriller	4226	4.833333	3.843367
100	Bowling for Columbine (2002)	Documentary	5669	4.833333	3.816139
32	Brazil (1985)	Fantasy|Sci-Fi	1199	4.800000	3.812348
27	African Queen, The (1951)	Adventure|Comedy|Romance|War	969	4.666667	3.799223

Before starting comparing the methods it is of the note to say that if we compare the rating given by clustering which a simple average the ratings given by the users belong to the cluster user id =6 with svd ratings (which is a machine learned model), we can see major differences between them. For some movies they are close.

First, we compare the clustering method with hybrid model:

Here is the results for user 6 by hybrid model:

```
In [336]: hybriRecoms
Out[336]:
```

	movieId	title	genres	svd_rating
906	1204	Lawrence of Arabia (1962)	Adventure|Drama|War	4.360359
828	1089	Reservoir Dogs (1992)	Crime|Mystery|Thriller	4.331112
615	780	Independence Day (a.k.a. ID4) (1996)	Action|Adventure|Sci-Fi|Thriller	4.282797
7467	81847	Tangled (2010)	Animation|Children|Comedy|Fantasy|Musical|Roma...	4.270350
704	922	Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)	Drama|Film-Noir|Romance	4.224275
720	940	Adventures of Robin Hood, The (1938)	Action|Adventure|Romance	4.214726
3608	4956	Stunt Man, The (1980)	Action|Adventure|Comedy|Drama|Romance|Thriller	4.202083
224	260	Star Wars: Episode IV - A New Hope (1977)	Action|Adventure|Sci-Fi	4.201388
10	11	American President, The (1995)	Comedy|Drama|Romance	4.179217
1730	2324	Life Is Beautiful (La Vita Ã¨ bella) (1997)	Comedy|Drama|Romance|War	4.170836
307	349	Clear and Present Danger (1994)	Action|Crime|Drama|Thriller	4.152468

Interesting results, if we compare first tow movies resulted from clustering method, they received better ratings from svd and the hybrid model could not find those two movies for this user. But after the first two, the Hybrid model recommended higher rated movies for this user.

Now we compare the winner with recommendation based on user similarity:

Here is the recommendations and svd predictions for user 6:

```
In [350]: userSimRecom
Out[350]:
    movieId    rating
1    1272.0  4.325810
5    4499.0  4.145883
12   4855.0  4.008402
19   3836.0  4.006612
0    2067.0  3.901977
11   4803.0  3.884240
10   4848.0  3.756637
8    4329.0  3.694753
13   4901.0  3.630097
16   4946.0  3.624043
2    2023.0  3.546120
17   4947.0  3.537325
4     163.0  3.486828
3    3682.0  3.477559
9    4498.0  3.451201
18   4238.0  3.447651
15   4945.0  3.408161
14   4917.0  3.391850
7    4310.0  2.933997
6    4254.0  2.710794
```

As we see, the highest rated movie here is 4.32. Still the winner is clustering method.

Comparing with Item similarity and here id the champion model:

```
In [382]: itemsimRecomsfinal
Out[382]:
    movieId                                     title                                        genres  svd_rating
17     1207                To Kill a Mockingbird (1962)                                        Drama    4.631622
9       587                               Ghost (1990)       Comedy|Drama|Fantasy|Romance|Thriller    4.468107
7       480                       Jurassic Park (1993)            Action|Adventure|Sci-Fi|Thriller    4.328830
14      780         Independence Day (a.k.a. ID4) (1996)            Action|Adventure|Sci-Fi|Thriller    4.282797
18     1213                          Goodfellas (1990)                                  Crime|Drama    4.232545
4       260  Star Wars: Episode IV - A New Hope (1977)                     Action|Adventure|Sci-Fi    4.201388
13      648                 Mission: Impossible (1996)           Action|Adventure|Mystery|Thriller    4.116748
15     1096                      Sophie's Choice (1982)                                        Drama    4.110280
8       500                       Mrs. Doubtfire (1993)                                 Comedy|Drama    4.087421
3       150                            Apollo 13 (1995)                        Adventure|Drama|IMAX    4.058974
19     1269                 Arsenic and Old Lace (1944)                     Comedy|Mystery|Thriller    4.001095
16     1197                 Princess Bride, The (1987)    Action|Adventure|Comedy|Fantasy|Romance    3.976770
10      589            Terminator 2: Judgment Day (1991)                               Action|Sci-Fi    3.973253
12      608                                Fargo (1996)                  Comedy|Crime|Drama|Thriller    3.953347
2       111                          Taxi Driver (1976)                        Crime|Drama|Thriller    3.917000
11      597                         Pretty Woman (1990)                               Comedy|Romance    3.884050
0         1                            Toy Story (1995)  Adventure|Animation|Children|Comedy|Fantasy    3.771992
5       296                         Pulp Fiction (1994)                  Comedy|Crime|Drama|Thriller    3.448927
1        50                  Usual Suspects, The (1995)                      Crime|Mystery|Thriller    3.393945
6       357           Four Weddings and a Funeral (1994)                               Comedy|Romance    3.245133

In [383]:
```

We can see that the movie "To kill a Mockingbird" received the highest rating from svd and other methods could not hunt that movie for the user id = 6.

5-Conclusion

In this project we have tried 7 different methods for recommending a movie to a user. For svd and svd++ we used RMSE and MSE for evaluation and for other methods we used different approach for different methods. We have seen that although the idea behind clustering approach is not very complicated but it did a very good job on recommending movies. The idea behind the clustering is similar to user-user approach but user-user was weakest model and clustering was semi champion.

We have used SVD, which is very powerful feature engineering based linear algebra, especially when we sparse matrices that is the case in many data science fields especially recommender systems. By SVD, we could have a perfect math supported tool to predict the rating for other users who have never seen the movies.

The results comes from the Hybrid Model and clustering shows that may be we should use one method to recommend top 2 movies and other methods for the rest. It was also possible to create a model by stacking all other models by considering the svd as a judge to compare the ratings and find to rated movies.