Movie Recommendation

DSC 680 – Project

Mehtre Vijaykumar

06/28/2024

Introduction

This project focus on recommendation system for video content providers to predict whether someone will enjoy a movie based on how much they liked or disliked other movies. Recommendation systems are a type of **information filtering system** designed to enhance search results by providing items that are more relevant to the search query or related to the user's search history.

We will do so by following a sequence of steps needed to implement recommendation system. We will start with preprocessing and cleaning of the raw text of the movies, author and ratings. Then we will explore the cleaned text and try to get some intuition about the context of the movies and ratings. After that, we will extract numerical features from the data and finally use these feature sets to train models and identify the movie which someone will enjoy.

Business Problem

Major companies like YouTube, Amazon, Netflix use recommendation systems in social and e-commerce sites use recommendation system for its users to suggest for an individual according to their requirement more precise and accurate item.
These online content and service providers have a huge amount of content so the problem which arises is which data is required for whom so the problem of providing apposite content frequently. This project represents the overview and approaches of techniques generated in a recommendation system.

There are basically three types of recommender systems: -

- **Demographic Filtering**- offers users with similar demographic background the similar movies that are popular and well-rated regardless of the genre or any other factors. Therefore, since it does not consider the individual taste of each person, it provides a simple result but easy to be implemented. The System recommends the same movies to users with similar demographic features. Since each user is different, this approach is too simple. The basic idea behind this system is that movies that are more popular and critically acclaimed will have a higher probability of being liked by the average audience.
- **Content Based Filtering**- consider the object's contents, this system uses item metadata, such as genre, director, description, actors, etc. for movies, to make these recommendations, it will give users the movie recommendation more closely to the individual's preference. They suggest similar items based on a particular item.

The general idea behind these recommender systems is that if a person liked a particular item, he or she will also like an item that is like it.

- **Collaborative Filtering**- focuses on user's preference data and recommend movies based on it through matching with other users' historical movies that have a similar preference as well and does not require movies' metadata. This system matches persons with similar interests and provides recommendations based on this matching. Collaborative filters do not require item metadata like its content-based counterparts.

Content based systems works based on the label or genre of an item. If a user watched a movie so it recommends similar movies based on director, a genre, and many more aspects. The theory behind the collaborative filtering is that if user's 'A' and 'B' have rated correspondingly in the past, then there will be an assumption that they will rate correspondingly in the future.

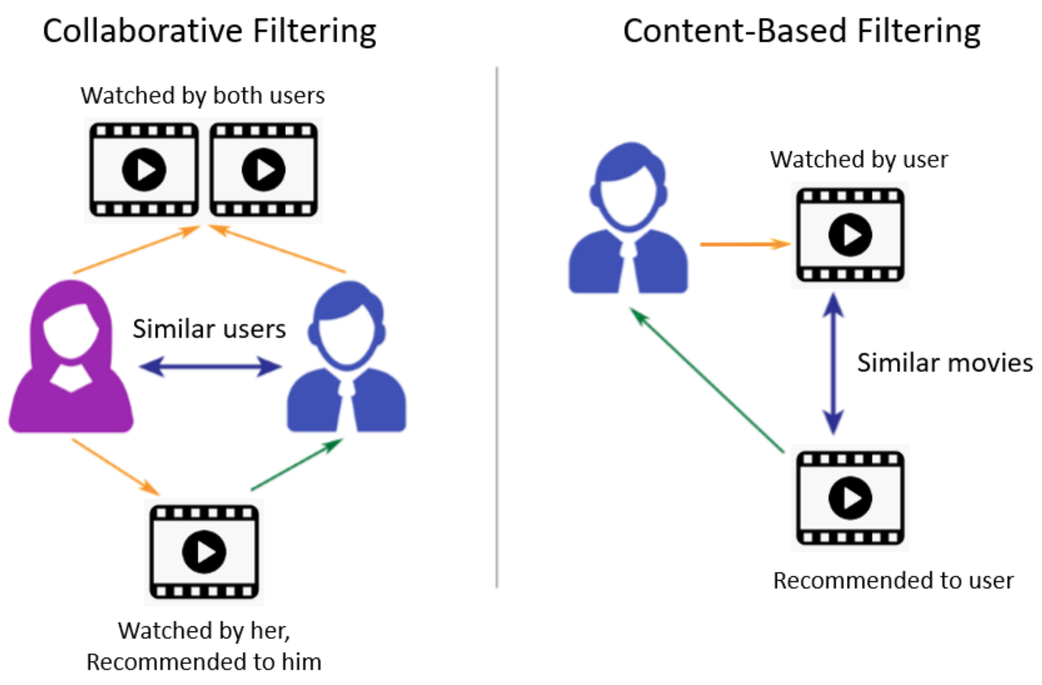


Figure 1 – Recommendation System

Data Explanation

I have used movie-lens dataset from: https://grouplens.org/datasets/movielens/latest/.

Data description

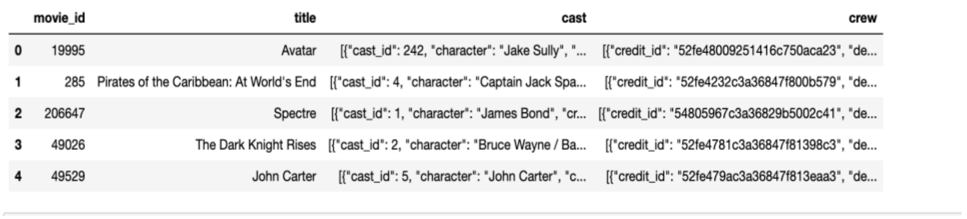
The dataset contains 100k+ ratings and 3k+ tag applications across 9k+ movies. The data was captured for 600+ users between 1996 and 2018. This dataset was generated in September 2018.

We are going to use 3 dataset , movie , credit and rating to implement movie recommendation system .

The **credits** dataset contains the following features: -

- movie_id - A unique identifier for each movie.
- cast - The name of lead and supporting actors.

- crew - The name of Director, Editor, Composer, Writer etc.



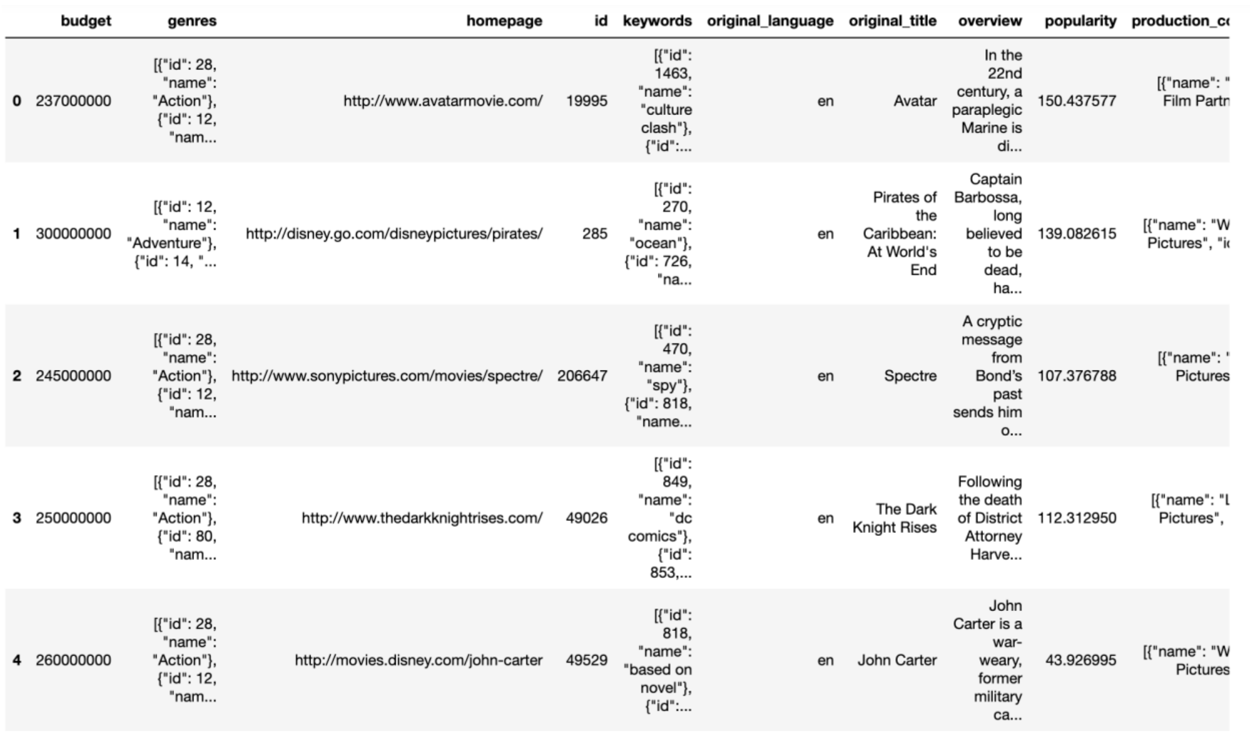
	movie_id	title	cast	crew
0	19995	Avatar	[{"cast_id": 242, "character": "Jake Sully", "...	[{"credit_id": "52fe48009251416c750aca23", "de...
1	285	Pirates of the Caribbean: At World's End	[{"cast_id": 4, "character": "Captain Jack Spa...	[{"credit_id": "52fe4232c3a36847f800b579", "de...
2	206647	Spectre	[{"cast_id": 1, "character": "James Bond", "cr...	[{"credit_id": "54805967c3a36829b5002c41", "de...
3	49026	The Dark Knight Rises	[{"cast_id": 2, "character": "Bruce Wayne / Ba...	[{"credit_id": "52fe4781c3a36847f81398c3", "de...
4	49529	John Carter	[{"cast_id": 5, "character": "John Carter", "c...	[{"credit_id": "52fe479ac3a36847f813eaa3", "de...

Figure 2 – Credit Data snapshot

The **Movie** dataset has the following features: -

- budget - The budget in which the movie was made.
- genre - The genre of the movie, Action, Comedy ,Thriller etc.
- homepage - A link to the homepage of the movie.
- id - This is infact the movie_id as in the first dataset.
- keywords - The keywords or tags related to the movie.
- original_language - The language in which the movie was made.
- original_title - The title of the movie before translation or adaptation.
- overview - A brief description of the movie.
- popularity - A numeric quantity specifying the movie popularity.
- production_companies - The production house of the movie.
- production_countries - The country in which it was produced.
- release_date - The date on which it was released.
- revenue - The worldwide revenue generated by the movie.
- runtime - The running time of the movie in minutes.

- status - "Released" or "Rumored".
- tagline - Movie's tagline.
- title - Title of the movie.
- vote_average - average ratings the movie recieved.
- vote_count - the count of votes receive

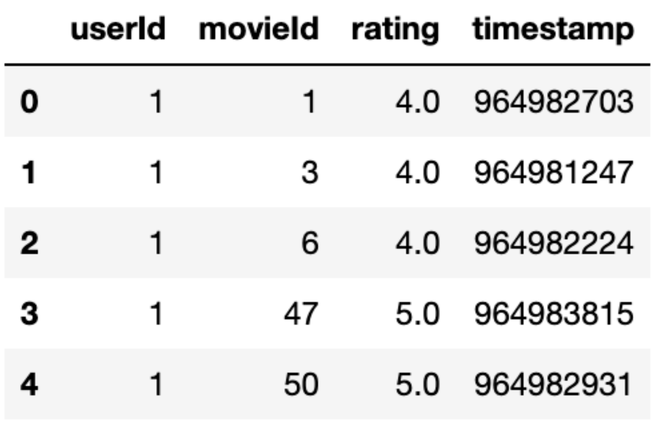


	budget	genres	homepage	id	keywords	original_language	original_title	overview	popularity	production_c
0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.avatarmovie.com/	19995	[{"id": 1463, "name": "culture clash"}, {"id":...	en	Avatar	In the 22nd century, a paraplegic Marine is di...	150.437577	[{"name": " Film Partn
1	300000000	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	http://disney.go.com/disneypictures/pirates/	285	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	en	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, ha...	139.082615	[{"name": "W Pictures", "i
2	245000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.sonypictures.com/movies/spectre/	206647	[{"id": 470, "name": "spy"}, {"id": 818, "name...	en	Spectre	A cryptic message from Bond's past sends him o...	107.376788	[{"name": " Pictures
3	250000000	[{"id": 28, "name": "Action"}, {"id": 80, "nam...	http://www.thedarkknightrises.com/	49026	[{"id": 849, "name": "dc comics"}, {"id": 853,...	en	The Dark Knight Rises	Following the death of District Attorney Harve...	112.312950	[{"name": "L Pictures",
4	260000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://movies.disney.com/john-carter	49529	[{"id": 818, "name": "based on novel"}, {"id":...	en	John Carter	John Carter is a war-weary, former military ca...	43.926995	[{"name": "W Pictures

Figure 3 – Movie Data snapshot

The **Rating** dataset has the following features: -

- userId - A unique identifier for user who submitted the rating
- movieId - A unique identifier for each movie.
- rating - rating for movie from 0 to 5
- timestamp – rating timestamp



	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

Figure 4 – Rating Data snapshot

Data Analysis and Implementation

1. Demographic Filtering

Before getting started with this -

- we need a metric to score or rate movie.
- Calculate the score for every movie.
- Sort the scores and recommend the best rated movie to the users.

We can use the average ratings of the movie as the score but using this won't be fair enough since a movie with 9 average rating and only 10 votes cannot be considered better than the movie with 8 as as average rating but 60 votes. So, we will be using IMDB's weighted rating (wr) which is given as :-

$$\text{Weighted Rating (WR)} = \left(\frac{v}{v+m} . R\right) + \left(\frac{m}{v+m} . C\right)$$

were,

- 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; And
- C is the mean vote across the whole report

We already have v(**vote count**) and R (**vote average**) and C can be calculated as

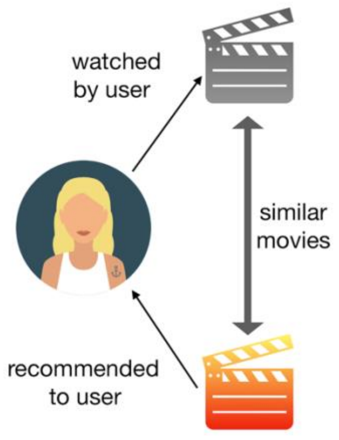
Weighted Rating

We need to calculate our metric for each qualified movie. To do this, we will define a function, **weighted_rating()** and define a new feature **score**, of which we'll calculate the value by applying this function to our DataFrame of qualified movies:

```
def weighted_rating(x, m=m, C=C):
  v = x['vote_count']
  R = x['vote_average']
  # Calculation based on the IMDB formula
  return (v/(v+m) * R) + (m/(m+v) * C)
```

2. Content Based Filtering

We will compute pairwise similarity scores for all movies based on their plot descriptions and recommend movies based on that similarity score.



We will compute Term Frequency-Inverse Document Frequency (TF-IDF) vectors for each overview. Term Frequency: **(term instances/total instances)**. Inverse Document Frequency: **log(number of documents/documents with term)** The overall importance of each word to the documents in which they appear is equal to **TF * IDF**

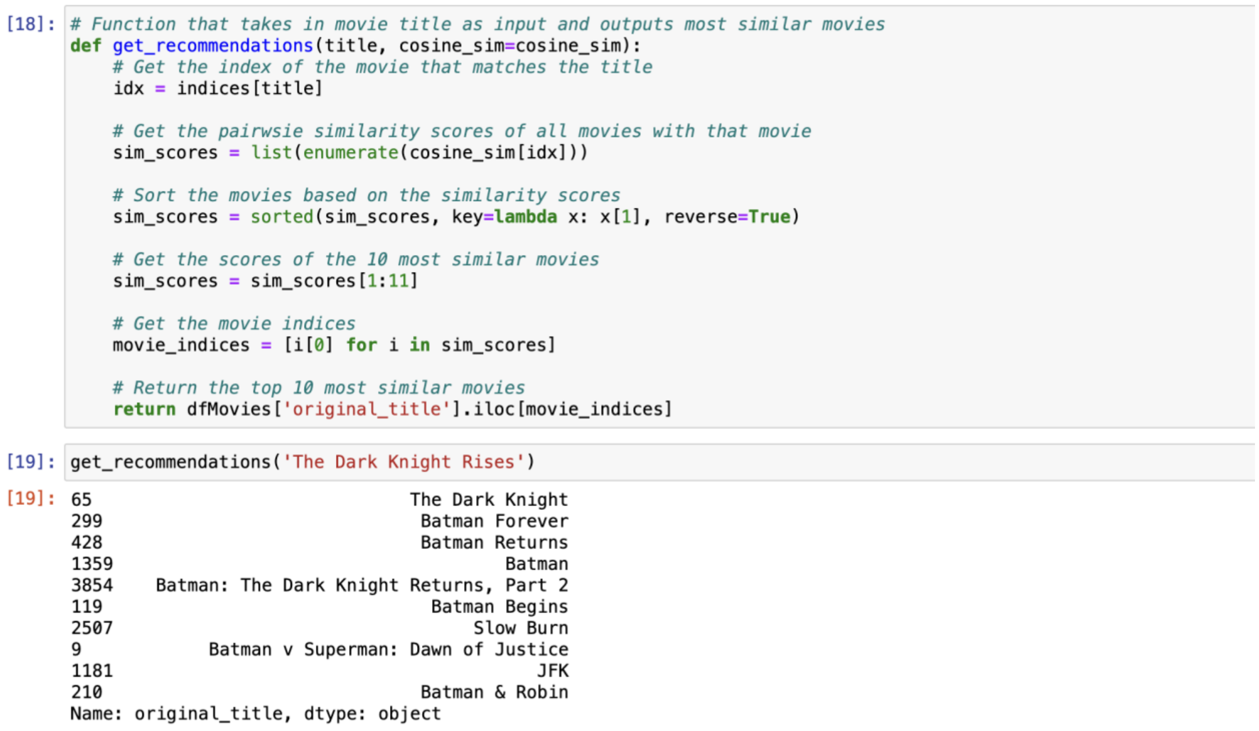
This will give us a matrix where each column represents a word in the overview vocabulary and each row represents a movie. This is done to reduce the importance of words that occur frequently in plot overviews and therefore, their significance in computing the final similarity score.

We will use the library scikit-learn which gives us a built-in TfIdfVectorizer class that produces the TF-IDF matrix.

These are the following steps we'll follow to define our recommender system:-

- Get the index of the movie given its title.

- Get the list of cosine similarity scores for that movie with all movies. Convert it into a list of tuples where the first element is its position and the second is the similarity score.
- Sort the mentioned list of tuples based on the similarity scores; that is, the second element.
- Get the top 10 elements of this list. Ignore the first element as it refers to self (the movie most similar to a particular movie is the movie itself).
- Return the titles corresponding to the indices of the top elements.



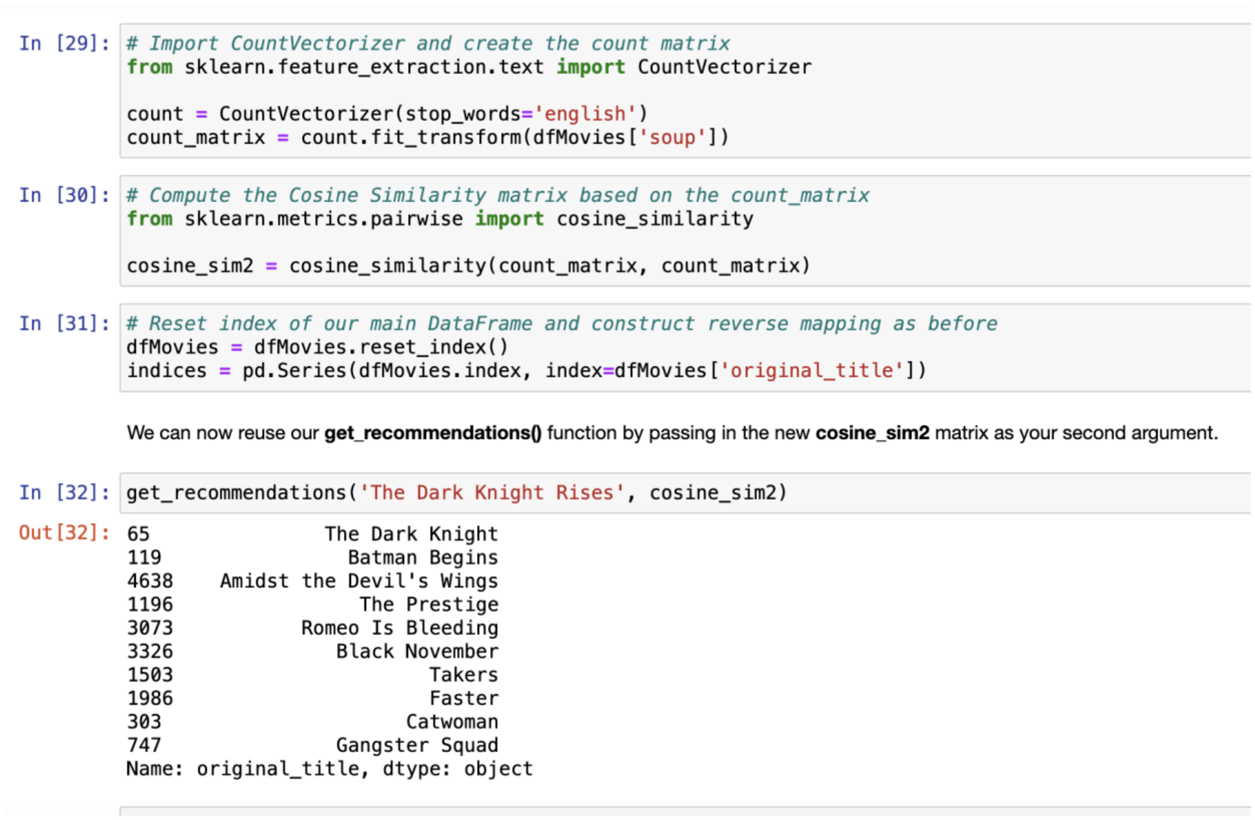
```
[18]: # Function that takes in movie title as input and outputs most similar movies
      def get_recommendations(title, cosine_sim=cosine_sim):
          # Get the index of the movie that matches the title
          idx = indices[title]

          # Get the pairwsie similarity scores of all movies with that movie
          sim_scores = list(enumerate(cosine_sim[idx]))

          # Sort the movies based on the similarity scores
          sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

          # Get the scores of the 10 most similar movies
          sim_scores = sim_scores[1:11]

          # Get the movie indices
          movie_indices = [i[0] for i in sim_scores]

          # Return the top 10 most similar movies
          return dfMovies['original_title'].iloc[movie_indices]
```

```
[19]: get_recommendations('The Dark Knight Rises')
```

```
[19]: 65                                 The Dark Knight
      299                                 Batman Forever
      428                                 Batman Returns
      1359                                        Batman
      3854    Batman: The Dark Knight Returns, Part 2
      119                                  Batman Begins
      2507                                     Slow Burn
      9            Batman v Superman: Dawn of Justice
      1181                                           JFK
      210                                 Batman & Robin
      Name: original_title, dtype: object
```

Improving Recommender System

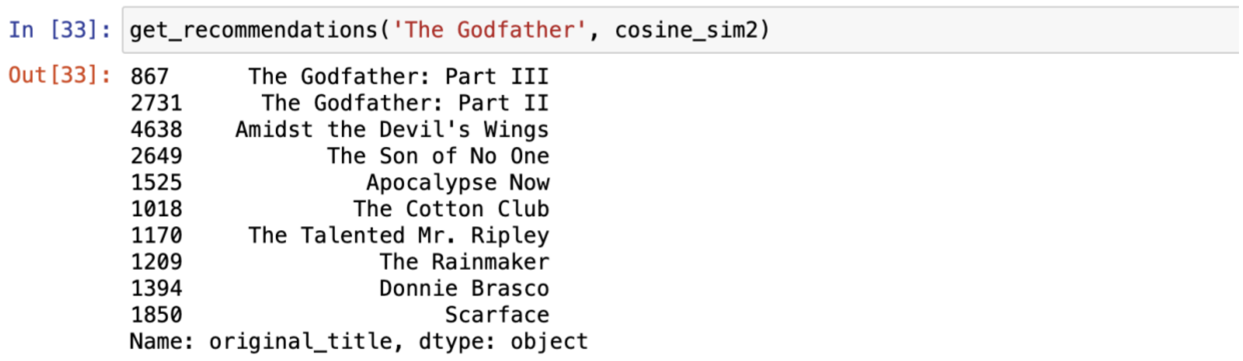
The quality of our recommender would be increased with the usage of better metadata. For this we are going to build a recommender based on the following metadata: the 3 top actors, the director, related genres and the movie plot keywords. From the cast, crew and keywords features, we need to extract the three most important actors, the director and the keywords associated with that movie.



```
In [29]: # Import CountVectorizer and create the count matrix
         from sklearn.feature_extraction.text import CountVectorizer

         count = CountVectorizer(stop_words='english')
         count_matrix = count.fit_transform(dfMovies['soup'])

In [30]: # Compute the Cosine Similarity matrix based on the count_matrix
         from sklearn.metrics.pairwise import cosine_similarity

         cosine_sim2 = cosine_similarity(count_matrix, count_matrix)

In [31]: # Reset index of our main DataFrame and construct reverse mapping as before
         dfMovies = dfMovies.reset_index()
         indices = pd.Series(dfMovies.index, index=dfMovies['original_title'])
```

We can now reuse our **get_recommendations()** function by passing in the new **cosine_sim2** matrix as your second argument.

```
In [32]: get_recommendations('The Dark Knight Rises', cosine_sim2)

Out[32]: 65                The Dark Knight
         119                 Batman Begins
         4638    Amidst the Devil's Wings
         1196                 The Prestige
         3073            Romeo Is Bleeding
         3326               Black November
         1503                       Takers
         1986                       Faster
         303                      Catwoman
         747                Gangster Squad
         Name: original_title, dtype: object
```



```
In [33]: get_recommendations('The Godfather', cosine_sim2)

Out[33]: 867      The Godfather: Part III
         2731      The Godfather: Part II
         4638    Amidst the Devil's Wings
         2649           The Son of No One
         1525              Apocalypse Now
         1018             The Cotton Club
         1170     The Talented Mr. Ripley
         1209               The Rainmaker
         1394               Donnie Brasco
         1850                    Scarface
         Name: original_title, dtype: object
```

After Applying the cosine similarity, we see that our recommender has been successful in capturing more information due to more metadata and has given us better recommendations.

We can also increase the weight of the director, by adding the feature multiple times in the metadata.

3. Collaborative Filtering

The content based engine is only capable of suggesting movies which are close to a certain movie. It is not capable of capturing tastes and providing recommendations across genres.

Also, the engine that we built 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.

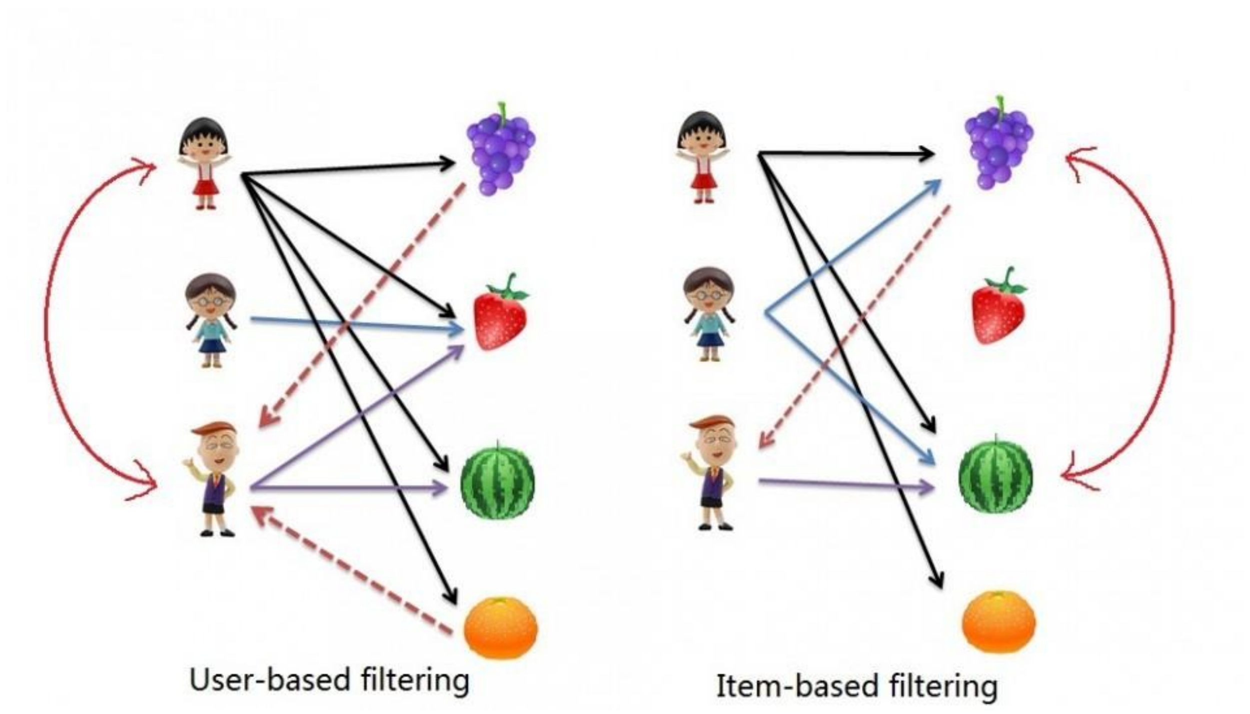
Therefore, in this section, we will use Collaborative Filtering to make recommendations to Movie Watchers. It is basically of two types:-

- **User based filtering**- User-Based Collaborative Filtering is a technique used to predict the items that a user might like on the basis of ratings given to that item by the other users who have similar taste with that of the target user. Many websites use collaborative filtering for building their recommendation system. Imagine that we want to recommend a movie to our friend Stanley. We could assume that similar people will have similar taste. Suppose that me and Stanley have seen the same movies, and we rated them all almost identically. But Stanley hasn't seen 'The Godfather: Part II' and I did. If I love that movie, it sounds logical to think that he will too. With that, we have created an artificial rating based on our similarity. Well, UB-CF uses that logic and recommends items by finding similar users to the active user (to whom we are trying to recommend a movie). A specific application of this is the user-based Nearest Neighbor algorithm. This algorithm needs two tasks:

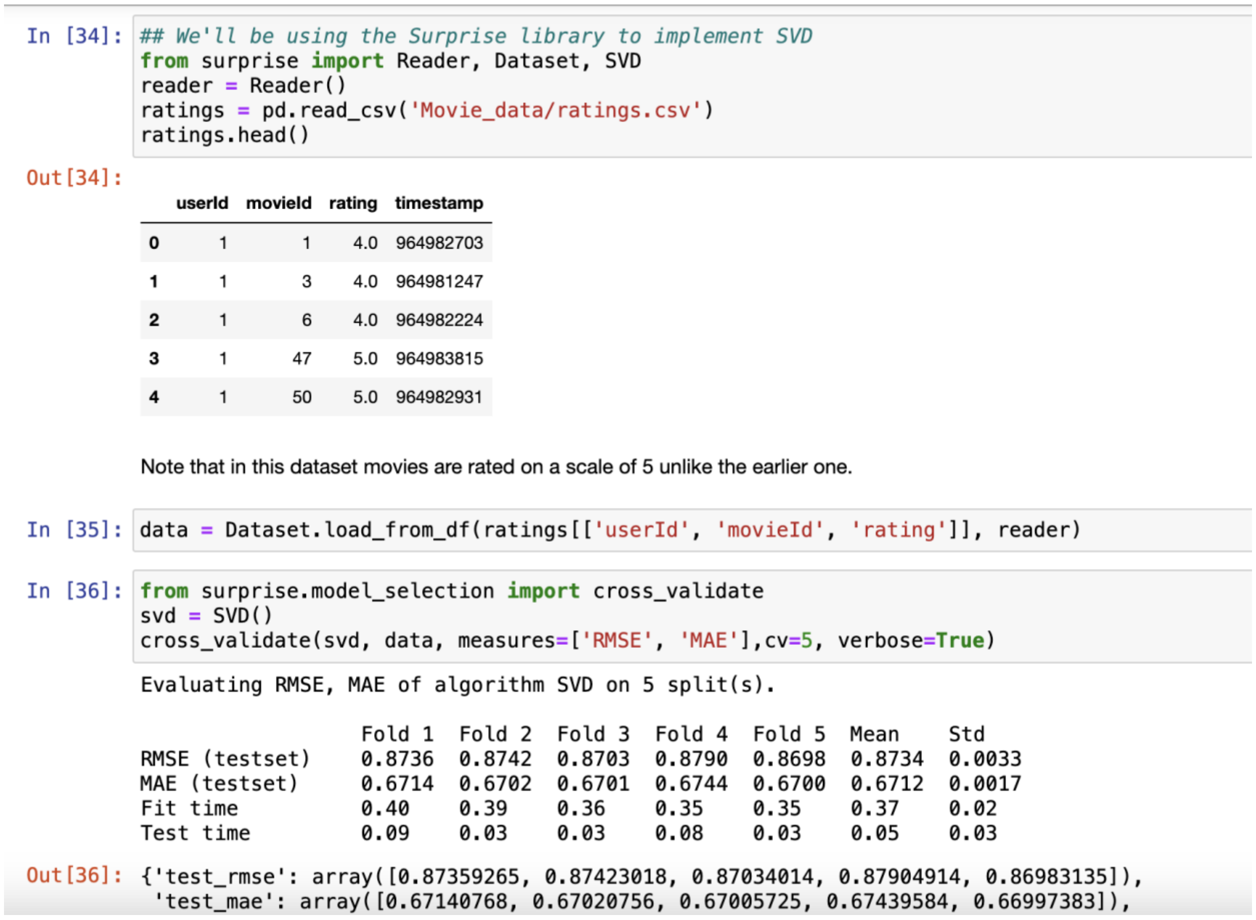
- Find the K-nearest neighbors (KNN) to the user a, using a similarity function w to measure the distance between each pair of users:
- Predict the rating that user a will give to all items the k neighbors have consumed but a has not. We Look for the item j with the best predicted rating. In other words, we are creating a User-Item Matrix, predicting the ratings on items the active user has not see, based on the other similar users. This technique is memory-based.

Although computing user-based CF is very simple, it suffers from several problems. One main issue is that users' preference can change over time. It indicates that precomputing the matrix based on their neighboring users may lead to bad performance. To tackle this problem, we can apply item-based CF.

- **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 opposed to the horizontal manner that user-based CF does.



It avoids the problem posed by dynamic user preference as item-based CF is more static.



In [34]:
```python
## We'll be using the Surprise library to implement SVD
from surprise import Reader, Dataset, SVD
reader = Reader()
ratings = pd.read_csv('Movie_data/ratings.csv')
ratings.head()
```

Out[34]:

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

Note that in this dataset movies are rated on a scale of 5 unlike the earlier one.

In [35]:
```python
data = Dataset.load_from_df(ratings[['userId', 'movieId', 'rating']], reader)
```

In [36]:
```python
from surprise.model_selection import cross_validate
svd = SVD()
cross_validate(svd, data, measures=['RMSE', 'MAE'],cv=5, verbose=True)
```

```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8736  0.8742  0.8703  0.8790  0.8698  0.8734  0.0033
MAE (testset)     0.6714  0.6702  0.6701  0.6744  0.6700  0.6712  0.0017
Fit time          0.40    0.39    0.36    0.35    0.35    0.37    0.02
Test time         0.09    0.03    0.03    0.08    0.03    0.05    0.03
```

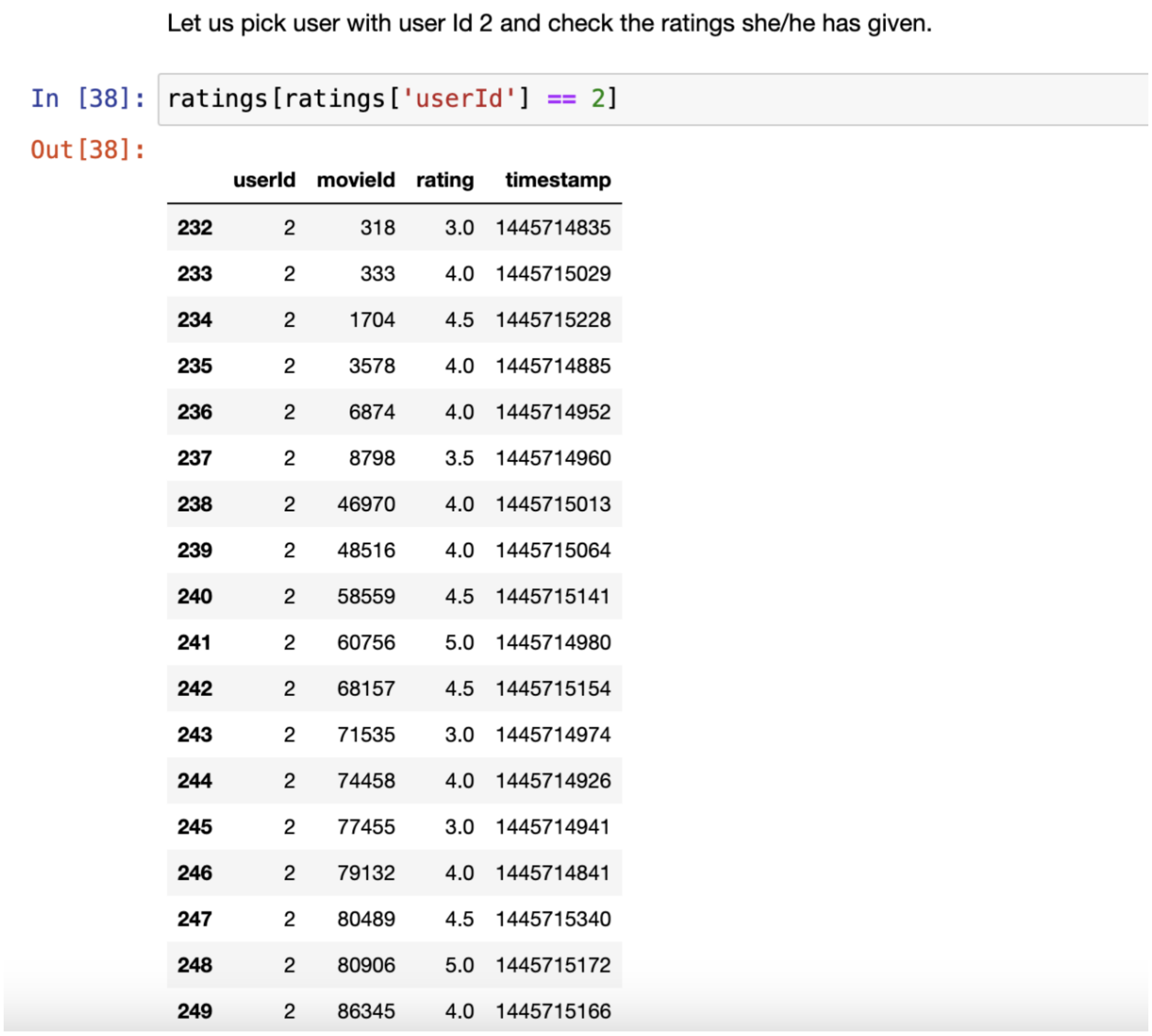
Out[36]:
```
{'test_rmse': array([0.87359265, 0.87423018, 0.87034014, 0.87904914, 0.86983135]),
 'test_mae': array([0.67140768, 0.67020756, 0.67005725, 0.67439584, 0.66997383]),
```

Out[36]:
```
{'test_rmse': array([0.87359265, 0.87423018, 0.87034014, 0.87904914, 0.86983135]),
 'test_mae': array([0.67140768, 0.67020756, 0.67005725, 0.67439584, 0.66997383]),
 'fit_time': (0.4006781578063965,
  0.38509106636047363,
  0.3554058074951172,
  0.3536229133605957,
  0.3520359992980957),
 'test_time': (0.08996081352233887,
  0.030429840087890625,
  0.03096604347229004,
  0.08120965957641602,
  0.030778169631958008)}
```

We get a mean Root Mean Sqaure Error of 0.87 approx which is more than good enough for our case. Let us now train on our dataset and arrive at predictions.

In [37]:
```python
trainset = data.build_full_trainset()
svd.fit(trainset)
```

Out[37]:
```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x32220edd0>
```

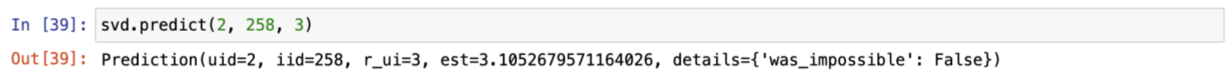


Let us pick user with user Id 2 and check the ratings she/he has given.

```
In [38]: ratings[ratings['userId'] == 2]
```

Out[38]:

	userId	movieId	rating	timestamp
232	2	318	3.0	1445714835
233	2	333	4.0	1445715029
234	2	1704	4.5	1445715228
235	2	3578	4.0	1445714885
236	2	6874	4.0	1445714952
237	2	8798	3.5	1445714960
238	2	46970	4.0	1445715013
239	2	48516	4.0	1445715064
240	2	58559	4.5	1445715141
241	2	60756	5.0	1445714980
242	2	68157	4.5	1445715154
243	2	71535	3.0	1445714974
244	2	74458	4.0	1445714926
245	2	77455	3.0	1445714941
246	2	79132	4.0	1445714841
247	2	80489	4.5	1445715340
248	2	80906	5.0	1445715172
249	2	86345	4.0	1445715166



```
In [39]: svd.predict(2, 258, 3)
Out[39]: Prediction(uid=2, iid=258, r_ui=3, est=3.1052679571164026, details={'was_impossible': False})
```

For movie with ID 258, we get an estimated prediction of **2.99**. This recommender system works based on an assigned movie ID and tries to predict ratings based on how the other users have predicted the movie, it doesn't care what the movie is.

Conclusion

We create recommenders using demographic, content- based and collaborative filtering. While demographic filtering is very elementary and cannot be used practically, **Hybrid Systems** can take advantage of content-based and collaborative filtering as the two approaches are proved to be almost complimentary.

Challenges

New User: A newly released movie cannot be recommended to the user until it gets some ratings. A new user or item added based problem is difficult to handle as it is impossible to obtain a similar user without knowing previous interest or preferences.

Synonymy: arises when a single item is represented with two or more different names or listings of items having similar meanings, in such condition, the recommendation system can’t recognize whether the terms show various items or the same item.

Ethical Consideration

Users for all the reviews in the dataset were selected at random for inclusion. All selected users had rated at least 20 movies. No demographic information is included. Each user is represented by an id, and no other information is provided.

Reference

[1]Herlocker, J, Konstan, J., Terveen, L., and Riedl, J. Evaluating Collaborative Filtering Recommender Systems. ACM Transactions on Information Systems 22 (2004), ACM Press, 5-53.

[2] Koren, Yehuda. “Factorization meets the neighborhood: a multifaceted collaborative filtering model.” In Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, 426–434. ACM, 2008.

[3] https://www.mygreatlearning.com/blog/masterclass-on-movie-recommendation-system/

[4] https://docs.microsoft.com/en-us/dotnet/machine-learning/tutorials/movie-recommendation

[5] MovieLens 2018 Introduction-to-Machine-Learning

https://github.com/codeheroku/Introduction-toMachineLearning/tree/master/CollaborativeFiltering/dataset

Appendix