MOVIE RECOMMENDATION

DSC 680 -PROJECT MILESTONE 2

Abhijit Mandal

04/02/2022

Introduction

Recommendation Systems are a type of **information filtering systems** as they improve the quality of search results and provides items that are more relevant to the search item or are related to the search history of the user.

They are used to predict the **rating** or **preference** that a user would give to an item. Almost every major tech company has applied them in some form or the other. 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. 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.

The goal of this project is to provide a recommendation system for video content providers to predict whether someone will enjoy a movie based on how much they liked or disliked other movies.

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 similar to it.

- **Collaborative Filtering**- T 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.

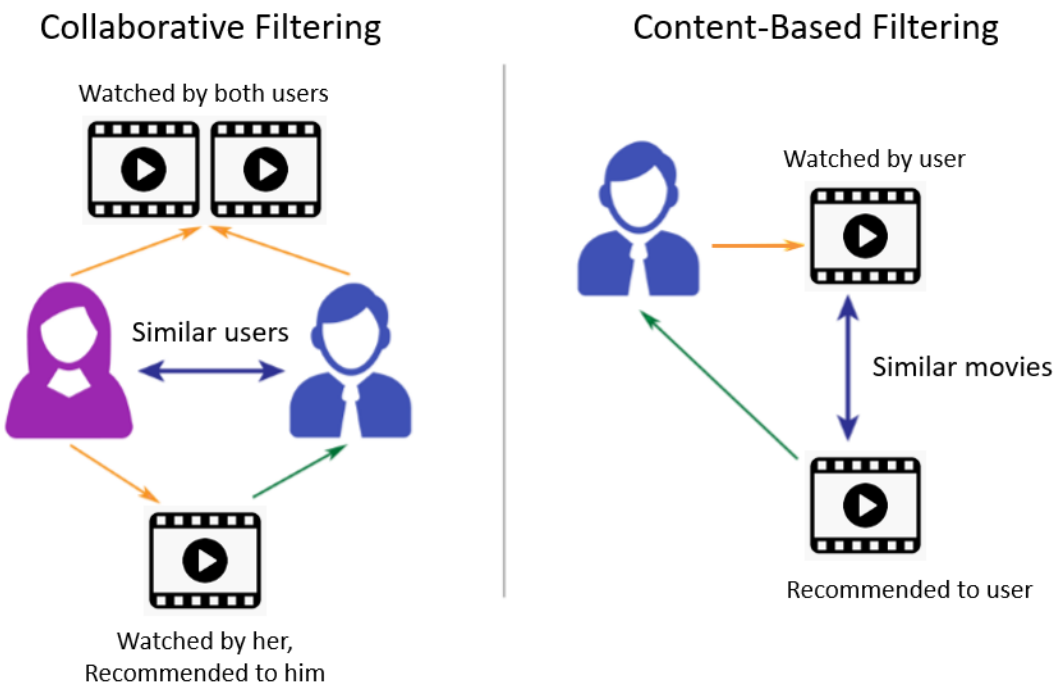


Figure 1 – Differentiating Collaborative with Content-Based Fitering

Data description

In this project we are applying item based collaborative filtering. The reason behind this is because user taste may change with respect to time but item doesn't change it remains same. There are certain stages to make our recommendation system efficiently to respond.

- **Data Loading** – To load the data and display accordingly we have to perform some operation like merging the two file in the dataset.
- **Data Slicing** – Here we are removing unnecessary column and data.
- **Data Cleaning** – In the real world data if we make a table of ratings in the recommendation system we find that most of the user are not rating the movies and are mostly inactive.

The same cases are with movies either users don't watch or it's get too old. To make our computation more accurate we will remove such users and movies from our research. Now to predict similarity we have two methods either we can use correlation method or cosine method. In our research we had used correlation method.

Approach:

Demographic Filtering

- 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 40 votes. So, we will be using IMDB's weighted rating (wr) which is given as :-

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

- 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:

```
C= dfMovies['vote_average'].mean()
```

Output: 6.092171559442011

So, the mean rating for all the movies is approx 6 on a scale of 10. The next step is to determine an appropriate value for m, the minimum votes required to be listed in the chart. We will use 95th percentile as our cutoff.

```
m= dfMovies['vote_count'].quantile(0.95)
```

Output: 1838.4000000000015

Now, we can filter out the movies that qualify for the chart

```
filter_movies = dfMovies.copy().loc[dfMovies['vote_count'] >= m]
```

Code Snippet 1

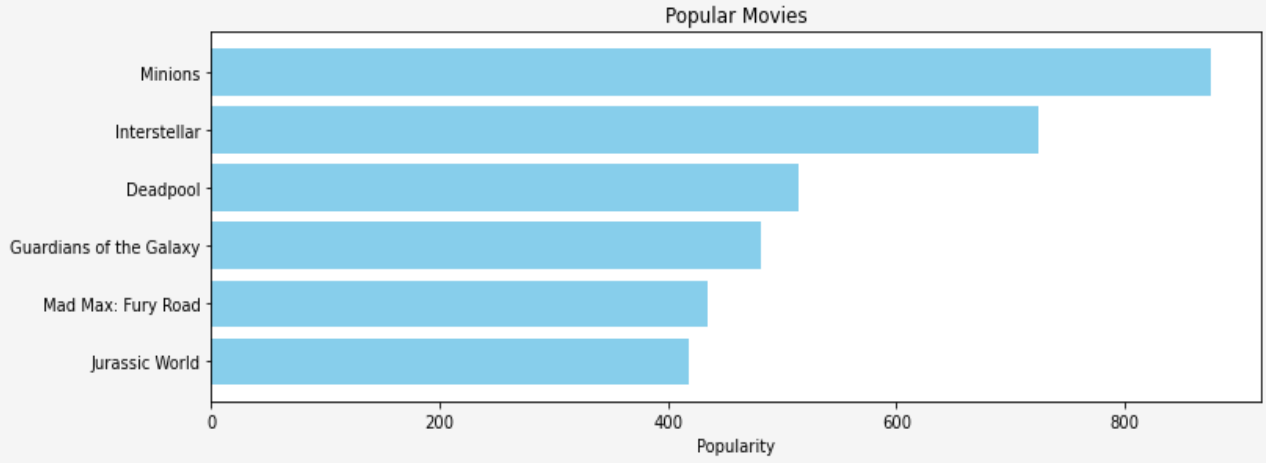
We see that there are 481 movies which qualify to be in this list. Now, we need to calculate our metric for each qualified movie. To do this, we will define a function, weighted_rating() and define a new feature score, of which we'll calculate the value by applying this function to our DataFrame of qualified movies:

feature 'score' and calculate its value with `weighted_rating()`

```
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)

filter_movies['score'] = q_movies.apply(weighted_rating, axis=1)
```

Under the Trending Now tab of these systems we find movies that are very popular and they can just be obtained by sorting the dataset by the popularity column.



The above demographic recommender provides a general chart of recommended movies to all the users. They are not sensitive to the interests and tastes of a particular user. In the following section we will do

Code Snippet 2

Content Based Filtering

In this recommender system the content of the movie (overview, cast, crew, keyword, tagline) is used to find its similarity with other movies. Then the movies that are most likely to be similar are recommended.

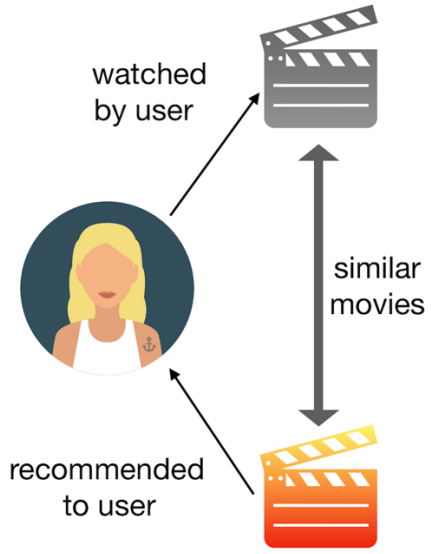


Figure 2: Content based Filtering

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.

```
#Import TfIdfVectorizer from scikit-learn
from sklearn.feature_extraction.text import TfidfVectorizer
#Define a TF-IDF Vectorizer Object. Remove all english stop words such as 'the', 'a'
tfidf = TfidfVectorizer(stop_words='english')
#Replace NaN with an empty string
df2['overview'] = df2['overview'].fillna('')
#Construct the required TF-IDF matrix by fitting and transforming the data
tfidf_matrix = tfidf.fit_transform(df2['overview'])

#Output the shape of tfidf_matrix
tfidf_matrix.shape
```

Output: (4803, 20978)

We see that over 20,000 different words were used to describe the 4800 movies in our dataset.

We will now compute a similarity score. There are several candidates for this; such as the euclidean, the Pearson and the cosine similarity scores. Different scores work well in different scenarios and it is often a good idea to experiment with different metrics.

We will be using the cosine similarity to calculate a numeric quantity that denotes the similarity between two movies. We use the cosine similarity score since it is independent

Code Snippet 3

of magnitude and is relatively easy and fast to calculate. Mathematically, it is defined as follows:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

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 particular 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.

Code Snippet 4

```
# 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 df2['original_title'].iloc[movie_indices]

get_recommendations('The Dark Knight Rises')
```

```
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: Dark of Justice
1181                                       JFK
210                              Batman & Robin
Name: original_title, dtype: object
```

As we see, the quality of recommendations is not that great. "The Dark Knight Rises" returns all Batman movies while it is more likely that the people who liked that movie are more inclined to enjoy other Christopher Nolan movies. This is something that cannot be captured by the present system.

Improving Recommender System

The quality of our recommender can 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.

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.

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.

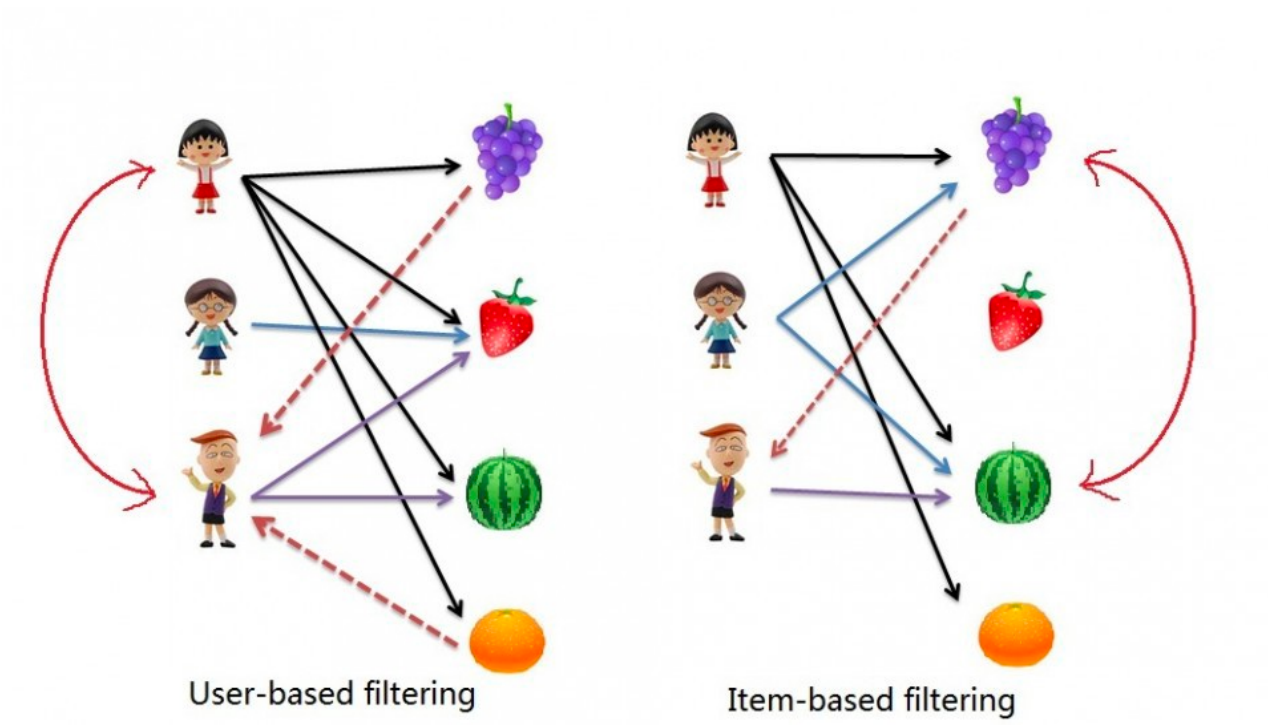


Figure 3 – User based vs Item based Filtering

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

Code Snippet 5

```
## We'll be using the Surprise library to implement SVD
from surprise import Reader, Dataset, SVD
reader = Reader()
ratings = pd.read_csv('MovieData/ratings_small.csv')
ratings.head()
```

\

	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.

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

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)    0.8731  0.8822  0.8692  0.8706  0.8766  0.8743
0.0046
MAE (testset)     0.6690  0.6760  0.6693  0.6690  0.6745  0.6716
0.0030
```

```
Fit time          4.17    4.14    4.17    4.15    4.11    4.15
0.02
Test time         0.08    0.08    0.13    0.08    0.08    0.09
0.02

{'test_rmse': array([0.87307556, 0.88215171, 0.86922027, 0.8705823
8, 0.87658646]),
 'test_mae': array([0.66902936, 0.67599411, 0.66932824, 0.66903745
, 0.67453348]),
 'fit_time': (4.1684417724609375,
  4.143563985824585,
  4.169765949249268,
  4.146688938140869,
  4.112934350967407),
 'test_time': (0.07999300956726074,
  0.07716584205627441,
  0.13251495361328125,
  0.07776594161987305,
  0.07889509201049805)}
```

We get a mean Root Mean Sqaure Error of 0.89 approx which is more than good enough for our case. Let us now train on our dataset and arrive at predictions.

```
trainset = data.build_full_trainset()
svd.fit(trainset)
```

```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x7fdd
22143af0>
```

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

```
ratings[ratings['userId'] == 2]
```

	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

	userId	movieId	rating	timestamp
253	2	99114	3.5	1445714874
254	2	106782	5.0	1445714966
255	2	109487	3.0	1445715145
256	2	112552	4.0	1445714882
257	2	114060	2.0	1445715276
258	2	115713	3.5	1445714854
259	2	122882	5.0	1445715272
260	2	131724	5.0	1445714851

```
svd.predict(2, 258, 3)
```

```
Prediction(uid=2, iid=258, r_ui=3, est=2.9957210112574106, details
={'was_impossible': False})
```

For movie with ID 258, we get an estimated prediction of **2.99**. This recommender system works on the basis of 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

The Recommendation system is a very powerful technology which helps people to find what they like. These systems have certain limitations not recommending efficiently to the users. If we take Collaborative Filtering as an example, then we find it is most successful and powerful algorithm, even though it is best but this algorithm has some high runtime and faces some issues like data Sparsity which can be removed by using a

Hybrid movie recommendation system. **Hybrid Systems** can take advantage of content-based and collaborative filtering as the two approaches are proved to be almost. complimentary. But each algorithms have its strength and weakness. Our proposed approach can handle quite a big amount of data effectively. In future, we will work on its weakness and on its user interface.

Questions:

1. **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. How to handle this scenario?
2. **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. How can we address this issue?
3. Scalability of the model ?
4. Drawbacks and limitations of Collaborative Filtering?
5. Drawbacks and limitations of Content Based Filtering?
6. Drawbacks and limitations of Demographic Filtering?
7. Which is the best Algorithm for Recommendation?
8. What are the factors that are taken into consideration while recommending a movie to the user, for eg. Age, demography, ethnicity, interests, language etc.
9. How much data is enough to predict recommendations for a user or does this changes with the amount of data we process?
10. How soon this recommender system needs to be re-trained ?

References

[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-toMachine Learning/tree/master/CollaborativeFiltering/dataset

Appendix