Recommendation System

Matej Petrinović

1 Introduction and data preparation

The rapid growth of data collection has led to a new era of information. Data is being used to create more efficient systems and this is where Recommendation Systems come into play. 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 realted to the search history of the user.

Figure 1: TMDB logo

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: Amazon uses it to suggest products to customers, YouTube uses it to decide which video to play next on autoplay, and Facebook uses it to recommend pages to like and people to follow. Moreover, companies like Netflix and Spotify depend highly on the effectiveness of their recommendation engines for their business and sucees. In this kernel we'll be building a baseline Movie Recommendation System using TMDB 5000 Movie Data.

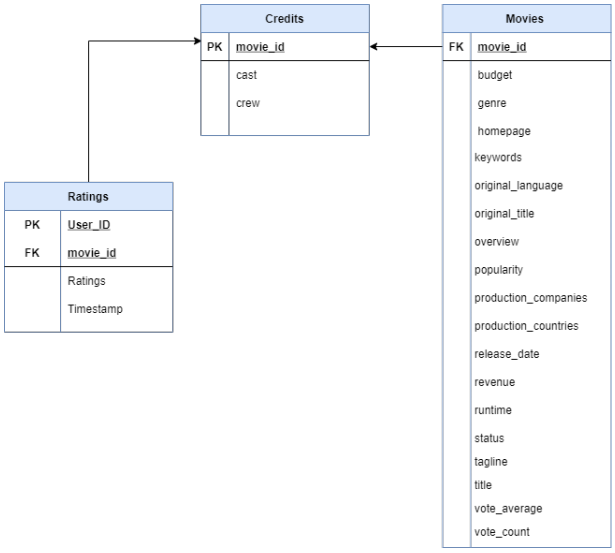

Figure 2: Entity Relation of TMDB 5000 Movie Data

For further modeling we will join Credits and Movies tables. There are basically three types of recommender systems which we will use in this work:

- Demographic Filtering
- Content Based Filtering
- Collaborative Filtering

Let's load the data and start working on it.

2 Building a Recommendation System

2.1 Demographic Filtering

They offer generalized recommendations to every user, based on movie popularity and/or genre. The System recommends the same movies to users with similar demographic features. Since each user is different , this approach is considered to be 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. 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 8.9 average rating and only 3 votes cannot be considered better than the movie with 7.8 as as average rating but 40 votes. So, I'll be using IMDB's weighted rating (wr) which is given as

$$WR = \left(\frac{v}{v+m} \cdot R\right) + \left(\frac{m}{m+v} \cdot C\right)$$

where,

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

We already have v(vote_count) and R (vote_average) and C can be calculated as $C = \frac{1}{N}\sum_{i=1}^{N} R_i$. So $C \approx 6$. The next step is to determine an appropriate value for m, the minimum votes required to be listed in the chart. We will use 85th percentile as our cutoff. In other words, for a movie to feature in the charts, it must have more votes than at least 85% of the movies in the list.

Finally, let's sort the DataFrame based on the score feature and output the title, vote count, vote average and weighted rating or score of the top 5 movies.

title	vote_count	vote_average	score
The Shawshank Redemption	8205	8.5	8.170528
Fight Club	9413	8.3	8.031958
Pulp Fiction	8428	8.3	8.004820
The Dark Knight	12002	8.2	7.993903
The Godfather	5893	8.4	7.982719

Table 1: Top 5 movies

We can see that most scored movie is The Shawshank Redemption, which is expected considering it's popularity. Now let's see plot that shows us most popular movies.

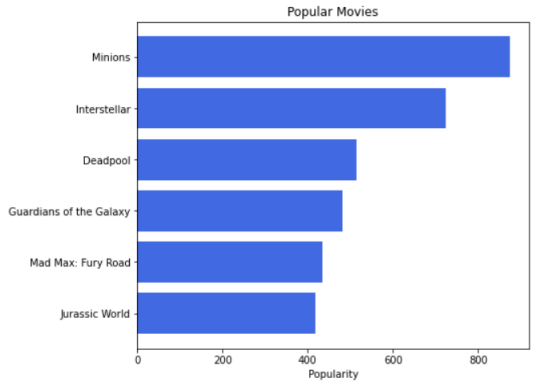

Figure 3: Most popular movies

As we can see that most popular movie is Minions, which is not top scored. Now something to keep in mind is that these demographic recommender provide a general chart of recommended movies to all the users. They are not sensitive to the interests and tastes of a particular user. This is when we move on to a more refined system- Content Basesd Filtering.

2.2 Content Based Filtering

Content Based Filtering suggest similar items based on a particular item. This system uses item metadata, such as genre, director, description, actors, etc. for movies, to make these recommendations. 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

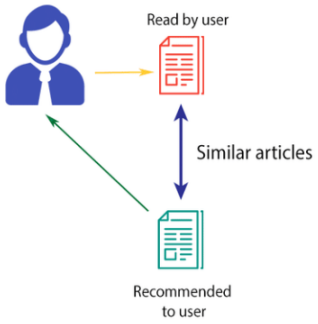

Figure 4: Content Based Filtering

We will compute pairwise similarity scores for all movies based on their plot descriptions and recommend movies based on that similarity score. The plot description is given in the overview feature of our dataset. Let's take a look at the data.

title	overview
Avatar	In the 22nd century, a paraplegic Marine is di...
Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, ha...
Spectre	A cryptic message from Bond's past sends him o...
The Dark Knight Rises	Following the death of District Attorney Harve...
John Carter	John Carter is a war-weary, former military ca...

Table 2: Movie Description

Now we'll compute Term Frequency-Inverse Document Frequency (TF-IDF) vectors for each overview. It is the relative frequency of a word in a document and is given as (term instances/total instances). Inverse Document Frequency is the relative count of documents containing the term is given as 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. It can be calculate by following formula:

$$\text{tf-idf}(t, d, D) = \left(0.5 + 0.5 \cdot \frac{f_{t,d}}{\max(f_{t,d} : t \in d)}\right) \cdot \log \frac{N}{|\{d \in D, t \in d\}|}$$

This will give you a matrix where each column represents a word in the overview vocabulary (all the words that appear in at least one document) and each row represents a movie, as before.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.

With this matrix in hand, we can now compute a similarity score. There are several candidates for this; such as the euclidean, the Pearson and the cosine similarity scores. There is no right answer to which score is the best. 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 of magnitude and is relatively easy and fast to calculate. Mathematically, it is defined as follows:

$$\cos(\theta) = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i \sum_{i=1}^{n} b_i}}$$

We are going to define a function that takes in a movie title as an input and outputs a list of the 10 most similar movies. Firstly, for this, we need a reverse mapping of movie titles and DataFrame indices. In other words, we need a mechanism to identify the index of a movie in our metadata DataFrame, given its title. So let's say we have watched The Lord of the Rings: The Return of the King, the system give us 10 most similar movies:

title
The Lord of the Rings: The Two Towers
The Lord of the Rings: The Fellowship of the Ring
Transformers
The Hobbit: The Desolation of Smaug
Night Watch
Once Upon a Time in the West
Sparkler
I Am Sam
Prince of Persia: The Sands of Time
Ouija

Table 3: Similiar Movies

While our system has done a decent job of finding movies with similar plot descriptions, the quality of recommendations is not that great. "TLord of the Rings: The Return of the King" returns all LOTR movies while it is more likely that the people who liked that movie are more inclined to enjoy other Peter Jackson movies. This is something that cannot be captured by the present system.

2.3 Collaborative Filtering

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

Also, the engine that we built is not really personal in that it doesn't capture the personal tastes and biases of a user. Anyone querying our engine for recommendations based on a movie will receive the same recommendations for that movie, regardless of who she/he is.Therefore, in this section, we will use a technique called Collaborative Filtering to make recommendations to Movie Watchers. It has two types:

- User based filtering
- Item Based Collaborative Filtering

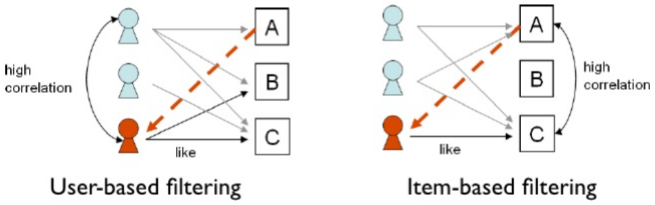

Figure 5: Collaborative Filtering

No matter which way we choose, they both work on the same principle. They're both based on Singular Value Decomposition.

SVD has a great property that it has the minimal reconstruction Sum of Square Error (SSE); therefore, it is also commonly used in dimensionality reduction. The below formula is SVD statemeant.

$$\min_{U,V,\Sigma} \sum_{i,j \in A} \left(A_{i,j} - [U\Sigma V^T]_{i,j}\right)^2$$

One way to handle the scalability and sparsity issue created by CF is to leverage a latent factor model to capture the similarity between users and items. Essentially, we want to turn the recommendation problem into an optimization problem. We can view it as how good we are in predicting the rating for items given a user. One common metric is Root Mean Square Error (RMSE). The lower the RMSE, the better the performance. For this purpose we'll use Ratings table.

userId	movieId	rating	timestamp
470	3435	3.0	1234089005
554	1333	4.0	1012753453
452	4304	3.0	1044681166
15	1573	4.0	997938476
510	84	4.0	947112271

Table 4: Ratings table sample

After trainig SVD, 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. Let us pick user with user Id 1 and check the ratings she/he has given.

userId	movie_id	title	rating
1	2105	American Pie	4.0
1	2294	Jay and Silent Bob Strike Back	2.0

Table 5: Movies rating on userid 1

Let's predict, which rate this user would give to the movie Swimming Pool. After predicting, the given result is 2.78. One startling feature of this recommender system is that it doesn't care what the movie is (or what it contains). It works purely on the basis of an assigned movie ID and tries to predict ratings based on how the other users have predicted the movie

3 Conclusion

We create recommenders using demographic , content-based and collaborative filtering. While demographic filtering is very elemantary 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. This model was very baseline and only provides a fundamental framework to start with.