Movie Recommender System

Joel Paulsson & Adrian Andersson
Linköping University

March 23, 2020

A recommender system is a program that recommends item that is interesting or useful for users based on the previous interactions. Media corporations, E-commerce companies and streaming services like *Netflix*, *HBO* and *Spotify* have their own kind of recommender systems to propose their consumers new, and tailored products, to offer a better experience with personalized content. This tailored experience for users is mostly positive but there is also an ethical dilemma which will be discussed in the conclusion. This informal article will explain the concepts of simple recommender systems and present a method of implementing this in python with the *MovieLens 20M dataset* [2]. The implementation is inspired by a blog article by Eric Le [1].
Recommender systems can be divided into three different types: content-, collaboration-, and popularity-based. The content-based method seeks out similarities between items based on a user's history while the collaborative methods seeks out similarities between users based on their interactions with items. Interactions with items is in our case if a user has watched a movie. Our goal was to implement a collaborative method using an item-item filtering and a co-occurrence matrix and answer the question: *Based on what movies a user have liked in the past what other similar movies will that user probably like based on other users with similar taste.*

1 Method

To our aid we have used the libraries and imports listed below.

```
#    IMPORTS     #
import pandas as pd
from sklearn.model_selection import train_test_split
import numpy as np
import time
import joblib as jl
import pylab as pl
```

1.1 Preprocessing

For our purpose the relevant files from the dataset are *'rating.csv'* and *'movie.csv'* which contains the columns *user_id, movie_id, rating* and *movie_id, title, genres* respectively. Pandas can be used to read the relevant *.csv*-files into dataframes. In this implementation the first 50000 rows of the datasets are considered. The resulting dataframes will look like in table 1.

Table 1: *Unmanipulated dataframes before merge.*

user_id	movie_id	rating
1	2	3.5
1	29	3.5
1	32	3.5
1	47	3.5
⋮	⋮	⋮

movie_id	title	genres
1	Toy Story	Adventure|Animation|...
2	Jumanji	Adventure|Children|...
3	Grumpier Old Men	Comedy|Romance
4	Waiting to Exhale	Comedy
⋮	⋮	⋮

To actually be able to use these dataframes we need to do some preprocessing. For example we need to drop irrelevant columns, redefine the column names, and merge the two dataframes into one. The following code will perform this for us.

```
#   ===== REARRANGE THE DATA ====== #
movie_df_1 = pd.read_csv(filePathData, nrows = 50000, header=None, low_memory=False)
movie_df_1 = movie_df_1.drop([3], axis=1)
movie_df_1 = movie_df_1.drop([0], axis=0)
movie_df_1.columns = ['user_id', 'movie_id', 'rating']
print(movie_df_1.head())
print("Length of dataframe:", movie_df_1.shape)

movie_df_2 = pd.read_csv(filePathMeta, header=None, low_memory=False)
movie_df_2 = movie_df_2.drop([0], axis=0)
movie_df_2.columns = ['movie_id', 'title', 'genres']
print(movie_df_2.head())
print("Length of dataframe:", movie_df_2.shape)
```

To merge the dataframes the code below is used.Furthermore we need to get rid of the rows that contain a user rating lower than a given threshold, as a bad rating should not influence the recommendations.

```
#   ===== MERGE THE TWO DATAFRAMES  =====   #
movie_df = pd.merge(movie_df_1, movie_df_2, on="movie_id", how = 'right')


#   ====== REMOVE LOW RATINGS ======    #

print(movie_df.head())
print("Length of dataframe:", movie_df.shape)
print(movie_df.dtypes)

movie_df['rating'] = movie_df.rating.astype(float)

print(movie_df.dtypes)

movie_df = movie_df[ ~(movie_df['rating'] <= 3)]
```

The reprocessing is now complete and the final dataframe is included in the table below.

user_id	movie_id	rating	title	genres
1	2	3.5	Jumanji	Adventure|Children|Fantasy
91	2	3.5	Jumanji	Adventure|Children|Fantasy
119	2	4.0	Jumanji	Adventure|Children|Fantasy
142	2	4.0	Jumanji	Adventure|Children|Fantasy
⋮	⋮	⋮	⋮	⋮

1.2 Item-item filtering

Item-item collaborative filtering, or sometimes called item-based filtering, focus on similarity between items a user have interacted with in a specific dataset. This can be implemented by defining a co-occurrence matrix based on the *Jaccard similarity*. The Jaccard similarity is a measure of similarity based on the intersection and union of two sets. In this application the sets are a user's high rated movies and all high rated movies. This means that we can formulate a question: *Based on what movies a user have liked in the past, what other similar movies will that user probably like based on other users with similar taste?* And to answer this question we need to create an instance (model) of our recommender class and supply it with some training data.

```
# =====  CREATE MODEL ===== #
is_model = Recommenders.item_similarity_recommender_py()
is_model.create(train_data, 'user_id', 'movie_id')
```

To construct the co-occurence matrix we need to follow the steps in Algorithm 1, below. Note that this step needs to be produced for all users in the dataset.

Algorithm 1: Constructing the co-occurence matrix.

```
co_occurence_matrix[length(all_movies)][length(user_movies)]
for i, in range: all_movies do
    viewers[] = all_movies[i].viewers
    for j in range user_movies do
        viewers[] = user_movies[j].viewers
        viewer_intersection = viewers[] ∩ viewers[]
        viewer_union = viewers[] ∪ viewers[]
        co_occurence_matrix[i][j] = viewer_intersection / viewer_union
    end
end
```

Furthermore, to generate a few top recommendations to a specific user, the columns of the co-occurence matrix are summed and normalized. We then sort the best scores for each movie in ascending order and remove a user's already watched movies as they will have the highest possible similarity score of 1 before returning the top 10 most similar movies.

2 Evaluation

To evaluate our model we need some common measure to be able to compare it with similar recommender systems. For this purpose a precision-recall curve is computed. In our case precision is defined as the number of correctly recommended movies divided by the total amount of recommended movies. Recall is defined as the number of correctly recommended movies divided by the total number of recommendations a user should receive. The average of the precision and recall over a set amount of users are then plotted on the x- and y-axis respectively. The resulting graph can be seen in the figure 1. A problem with the precision and recall measures is that as the number of recommendations increases the precision decreases and the recall increases. The F1 score in equation (1) might therefore be a a better alternative for evaluation of the model. From the table below it is evident that the recommended movies are relevant, i.e. they belong to the same genre and in one example a prequel is recommended.

$$F1 = \frac{2 * precision * recall}{precision + recall} \tag{1}$$

Table 2: *Example of recommended movies and a user's movie history.*

User 91's watched movies	User 91's recommended movies
Batman (1989)	RoboCop (1987)
Full Metal Jacket (1987)	Reservoir Dogs (1992)
Die Hard: With a Vengeance (1995)	Die Hard (1988)
Hotel Rwanda (2004)	Beetlejuice (1988)
Enemy at the Gates (2001)	Citizen Kane (1941)
Ronin (1998)	Terminator, The (1984)
⋮	⋮

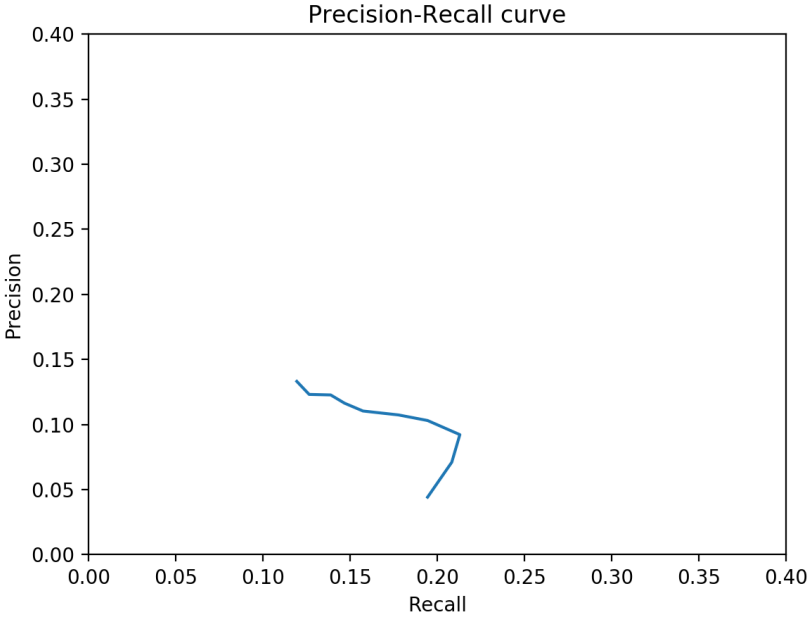

Figure 1: *Precision-recall graph.*

3 Conclusions

Keeping in mind that the number of movies are upwards 20000 we are content with the results of our model. Further work might involve comparing our collaborative-based model with a content-based model and by giving high rated movies a heavy weight associated with them. In the tutorial [1] a popularity model was implemented but was disregarded in our implementation as it's recommendations were not personalized and performed very poorly compared to the collaborative model. For real applications a model is most likely to be a combination of content- and collaborative-models to take advantage of the strengths of both models in order to achieve the best results. Notably, our model depend on a combination of feedback, both explicit in the form of ratings, but also implicit in the form of movies a user have watched. Both which will affect the movies recommended.

Using machine learning to create recommender systems is something that most big companies in many industries do to offer a better experience with personalized content for their consumers and consistently increase interest and sales. In order to have a precise recommender system large data of the users behaviour is required. That is why collecting and saving data has become a big thing in recent years.

By personalizing what content a user gets exposed to companies have the power to highlight some content and completely hide other. This might be an ethical problem in some areas, like news about politics for example. If the recommendations are too narrow only one type of news is shown to a user. This could be a democratic issue since voters might not get exposed to all sides of a story.

To combat this a random recommendation amongst other recommendations could be a solution. A user would thus get exposed by alternative media while still being recommended personalized content. In the previous news example this might lead to voters making a better informed decision based on what news they are exposed to.

For our recommender system regarding movies, we might want to adapt the same approach with a random recommendation to avoid for a user to get stuck with only one type of recommendations. Looking at table 2 one can see that user 91 likes action movies, and getting similar recommendations, which is good. But with previous discussion in mind there should be another genre represented among the recommendations, for example comedy.

Bibliography

[1] Eric Le. *How to Build a Simple Song Recommender*. URL: https://towardsdatascience.com/how-to-build-a-simple-song-recommender-296fcbc8c85.

[2] *MovieLens 20M Dataset*. URL: https://www.kaggle.com/grouplens/movielens-20m-dataset.