Movie Recommendations

Motivation

One of the aim of almost all the platforms is to give the users a reason to keep coming back to the platform. With the advent of tools and techniques, the platforms are able to intuitively "suggest" the relevant content to the user. This problem of building a recommender system was of my interest with an aim to understand the mechanism behind such suggestions. These systems can be thought as a good replacement of search tab. As in, instead of user searching on the platform, it should suggest content which is relevant to the user, looking at its previous trends. Players like Amazon, Netflix, YouTube can categorize their humongous content with respect to every user individually and recommend information that would lead to more customer engagement.

Problem Statement

With respect to this project, I've restricted myself to the problem of Movie recommendations. Given a set of users, their previously watched movies, what new movies could I suggest the user that he is most likely to see. This information would be retrieved from the given set of dataset, observing at patterns of the other users. It is even possible to gather information from other sources, which is for now beyond the scope of this project.

Data set

The data set I'll be working on is called the Movielens. It contains data of 671 users created between 1995 to 2016, who have collectively rated around 9000 movies. The users rate movies on a scale of 1-5. All of them have rated at least 20 movies. The timestamp when the user rates a movie is also given. The movie is mapped to a unique id and also bucketed into different genres.

Approach and Algorithm

One of the most accepted and commonly used algorithm to tackle the problem of building a recommender system is Collaborative Filtering. As the name suggests, it is a filtering technique that identifies information that is relevant to a user, looking at patterns in the dataset. This algorithm is known to have varied applications beyond recommender system.

The basic assumption/intuition in the technique is that if a user A rates a movie "Batman" 5, then another user B who is found to have similar patterns as that of A would most likely like batman. Thus, at a very high level, the problem can be formulated as given a particular user, find similar users and recommend movies that these users have rated high, but is not seen by the user. In doing this, the algorithm should identify patterns, relations among users and then recommend various items to different users.

I have looked upon three basic approaches to the recommender system.

1. Find the most "similar" users in the database and then identify the relevant content to be recommended
2. Use unsupervised technique to cluster the users and filter the useful content based on user data
3. A hybrid of both above models. Where the users are clustered first and then the most similar user is found to recommend information

Similar Users:

The algorithm sees users as vector, with values equal to the rating that they have given to different movies. Most of the values in the vector will be zero, as it is unlikely that the user has seen and reviewed most of the 9000 movies in the set. This is one of the most important problem of recommendation system, handling sparse datasets.

For example, consider the following grid:

User	Superman	Batman	Ironman
A	5	4	5
B		2	
C	4		

It is evident that user A likes super-hero movies. Of the other two users, B has rated 'Batman' as only 2 where as C has rated 'Superman' 5. We can intuitively say that user C is similar to A than B and can recommend movies like 'Batman' or 'Ironman' to user C.

Similarity in users can be measured using various metrics like: Pearson correlation, Euclidian distance, cosine similarity. I have explored cosine similarity and Euclidian distance in this project.

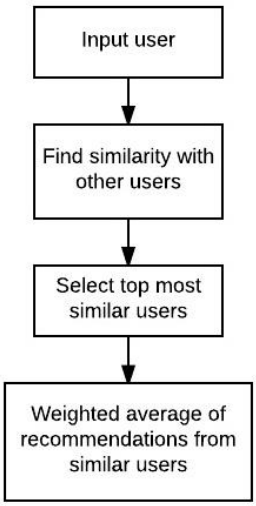
Cosine similarity = $\frac{A.B}{|A|*|B|} = \frac{\sum a_{Ai} a_{Bi}}{\sqrt{\sum a_{Ai}^2}\sqrt{\sum a_{Bi}^2}}$

Where A = vector of user 1 and B = vector of user 2. |A| is the modulus of vector. Maximum value of cosine similarity, most similar is user A to user B.

Euclidian distance = $\sqrt{\sum_{i=1}^{n}(a_{Ai} - a_{Bi})}$

Where a_{Ai} is the rating by user A to i^{th} movie.

Flowchart:



- The user for which the recommendations are calculated is chosen.
- The previous ratings of the user are compared with others and the similarity is stored in an array.
- The top 'k' users are chosen and a list of movies that they have seen and are missing in the user is identified.
- A weighted average of these movies with respect to the ratings that user has given them are taken into consideration.
- The list is again sorted with respect to ratings and it provided as an output.

In unsupervised technique, one of the most widely used method is K-means clustering. Where the dataset is clustered based on the Euclidian distance among the points. A starting point of this algorithm are randomly selected 'k' users as cluster midpoints and then they are updated after every iteration. The recommendations are given based on the frequency of movies seen and rated positively in the cluster. For this model, I took help of SKlearn's Kmeans package. The input to it was a pandas dataframe created post data processing where I have stored the user vectors in row, not considering the time stamp.

In a hybrid model, first k-means clustering is performed. The user is then identified to which cluster will it belong to based on the shortest distance from the cluster midpoints. Once the cluster is identified, most similar users are found, and the recommendations are reported.

Handling dataset in python:

The code I've written stores the information of each user as python dictionaries. For example,

{User1 : { Movie1 : [5, 9324342]; Here, User 1 has rated Movie1, 5 at timestamp 9324342
Movie2 : [2, 5464564]; Similarly for other users, movies.
Movie3 : [4, 6735745] … };

User2 : {Movie5 : [3, 6456455];
Movie1 : [4, 5675656] …}; …}

The output for model 1 and 3 is in terms of list of tuples:

[(93, 5.0),
(209, 5.0),
(318, 5.0),
(513, 5.0),
(642, 5.0),
(14, 5.0)]

It is expected that the user if watched movie id 93, would rate 5. Hence is recommended by the model.

The output for model 2 is just a list of movie ids:

[93,
209,
318,
513,
642,
14]

Conclusions

1) In the first approach so far, I have talked about user-user recommendation, another form is item-item recommendations which consider the movie column as vectors and then follow similar analysis. Both these methods face the problem of sparse matrix and can be tackled using dimensionality reduction. These methods are iterative and hence very costly. Many a times the similarity values are stored in an array and because of this memoization attribute, this method is also called "memory-based approach" at times. This method gives the movie id and also the expected rating that the user will give to the movie. The model can output as many movies as required using this technique.

2) Clustering of data points help reduce the complexity by a good factor. At a very high level, this approach can be seen as one of the machine learning technique. It tries to give recommendations based on the cluster analysis. The most frequently watched,

positively rated movie is recommended in decreasing order of significance. This approach is sometime referred as “model-based” recommendations system. Other useful models can be k-nearest neighbors or Bayesian models. This k-means approach cannot really predict what rating will the user would give to the movie.

3) The hybrid model uses the best of both models. In case of large datasets, this approach can reduce the complexity of the problem greatly. It first reduces its search based on cluster the user belong to. Further which, the most similar user is found using the iterative search approach. Finally, the recommendations are given with respect the weighted average of the movies that most similar users have rated positively.

The results were observed to be quite different in each case. Also, one point should be noted that the ratings predicted were 5 for almost all the recommended products, so if we were to compare the results between two approaches, we would have to consider all the 5 rated items at once, else it won’t be a good comparison.

In order to improve the recommendations, more attributes of the movie should be taken into consideration, like genre, date of release. Adding more features is always preferred to improve the accuracy of the system.

Guide to use the code:

The zip folder contains two csv files: ratings.csv and user_movieid.csv.

Please update the file location in the filename variable in the .py file.

Run the py file. The output should be the recommendations of all the three models one after the other.

References:

https://en.wikipedia.org/wiki/Collaborative_filtering

http://infolab.stanford.edu/~ullman/mmds/ch9.pdf

https://blog.statsbot.co/recommendation-system-algorithms-ba67f39ac9a3

http://files.grouplens.org/datasets/movielens/ml-latest-small-README.html