Movie recommendation system with Collaborative Filtering using K-NN

Debdeep Bose. Author,MSc. Big Data Analytics & Artificial Intelligence (Group A), Letterkenny Institute of Technology, Donegal, Ireland

***Abstract*— Recommendation system can be defined as a system that produces individual recommendations (a personalized way of possible options) as an output based on their previous choices which are considered an input by the system. Most of the products that we use today are powered by recommendation system. Such as books, music, news, items etc. However, in this paper we would be discussing about a recommender system for movies on Memory based, Model based collaborative filtering and Content-based filtering while comparing the accuracy of each recommendation system.**

***Index Terms*— Recommendation system, collaborative Filtering, content-based filtering**

I. Introduction

A Recommendation system is a smart system that can suggest items or products based on the previous information or data of the individual user. Currently, the recommendation system is used in all the major search engine websites or any e-commerce giants. The user almost gets overrun with product or item recommendation in these kinds of businesses. Majority of them however use the individuals past/historic data or preliminary matches to recommend. There are multiple studies that have shown that recommendation system proves to give colossal value to the businesses. Although recommendation systems are the background source of these businesses. It takes a few machine learning techniques and algorithms to build one. ”A recommendation system can be largely described into three categories: collaborative filtering systems, content-based systems and hybrid systems”. [1] Collaborative filtering is dependent on the relation between users and its items. However, bare bone collaborative filtering will not be a suitable choice for recommendations of movies. Rating a movie or a choice of music is a very profound personal experience of an individual, it cannot be generic like a grocery product. Their mood, experience, stage of life can all be an important factor or attribute for any kind of recommendation. Hence, the logic of suggesting an individual with a movie based on his/her previous choices can be a challenging effort. Recommending a movie to a person whom you know personally can be comparatively a very easy task as a few attributes such as character, age, personality, trait and movie taste is already known. Dismally, the movie recommendation system in unaware of such anomalies. Hence the personalised decision making can be quite a difficult task. To produce accurate results of recommendation data from multiple users have to be considered and used. Especially focusing on how to utilize the user’s particular movie rating data to predict the users next selection more effectively. To achieve this, the recommendation system that has been implemented here uses collaborative filtering which uses rating data in the form of matrix where the movies are represented in columns and users in rows and implementing K-nearest neighbours’ model to measure the similarity of the user using distance metrics to fill in the blanks for ungiven rating and then come up with a recommendation. [1]

Under the supervision of Dr. Shagufta Henna, Letterkenny Institute of Technology, Letterkenny, CO. Donegal

II. Literature Review

Generally, a recommendation system hugely depends on the inputs of a user and their relationship between the products.

A. Content-based Filtering

The recommendation system relies on the likeness of the products. The overall idea is if you like a particular product then it is highly considered or accepted that you will also like a similar product. It works well when the property or the characteristics of a product can be determined easily. In terms of Movie recommendation system, A content-based recommendation system can recommend a movie based on user data provided by them explicitly, following which a user profile is generated. This profile is further used to make suggestions which becomes more accurate overtime.

In content-based system the ”concepts of Term Frequency and Inverse document frequency are used for filtering systems and information retrieval”. [2] The primary use of these terms is to derive the importance of any movie. Term frequency can be described as the frequency or the number of times the word in a document. Whereas, inverse document frequency is the whole collection of documents. For Instance, suppose in a search engine we type ‘The result of English Premier League’. It is quite assertive that occurrence of the word ‘The’ will be more frequent than ‘English premier league’ for any other search queries. Therefore, the importance of ‘English premier league’ is very high with regards to ‘The’. In such cases term frequency and inverse document frequency weighting is used to determine the importance. [2]

B. Memory-based Collaborative Filtering

The memory based collaborative filtering is totally based on previous behaviour and not on the context. It is generally considering the similarity choices and preferences of two users. It gives recommendations to a user based on the preferences of a similar user. For example, if an individual likes a movie A, X and Z and another individual likes a movie X, Z and U. Then it is assumed that the first individual is more likely to watch the movie U. This is also one of the most often used algorithm because it does not have to depend on any extra information. It is a unique property of the algorithm that allows it to feature learn on its own. [2] The two major types of memory-based collaborative filtering are

1) User-User collaborative filtering: The major criteria here is to find look alike or similar users based on their past selections. The algorithm is time and resource consuming but very effective too. Therefore, it requires and very strong parallel computing system because the algorithm is hard to implement with huge base platform.

2) Item-item collaborative filtering: This type of filtering mechanism is quite like the previous algorithm. But, instead of focusing on user similarity this targets the movie. It is used to recommend similar movies to user who have rated he movies from the dataset. This algorithm as compared to the previous one barely uses any resource and time and a correlated movie matrix is framed over the period.

However, in either scenario there is a similarity matrix that is getting built which consists of a few distance metrics that is used for measuring the similarity. Firstly, Jaccard similarity in which the similarity is based on the amount of users who have rated the movies. It is applied to the scenarios when there is no available numeric rating but rather just available Boolean value. Secondly, we have cosine similarity which considers the closest vectors with smaller angles and larger cosines. Lastly, there is Pearson similarity which is considered the coefficient among two vectors. [2]

C. Model-based collaborative filtering

As we saw in the previous model that how users rating data can be used to recommend movies. However, using that can give high root mean squared which later on gives bad recommendations when the data set becomes enormous. The model-based collaborative filtering is based on a unsupervised learning method where it is exposed to Dimensionality reduction and latent variable decomposition. This model is used for the recommendation systems where it can deal with sparsity and scalability. The major target here is to learn the characteristics of the ratings and features such as user preference and item attributes. To predict the blank ratings with the help of dot product of its items and users. When given a sparse matrix with multiple dimensions the model can be restructured with the help of matrix factorization. Well, on a higher level it uses singular value decomposition in order to decompose into diagonal matrix and double unitary matrix. [2]

III. System Overview

The recommendation system that we are focused relies on model based collaborative filtering with the implementation of K-nearest neighbour algorithm.Here, This recommended system is based on using collaborative filtering which implies user rating data. The representation of rating data is in the form of a matrix where the movies are represented in columns and users in rows. However, there are instances where there are blank cells that represent no ratings given in the matrix. Hence in order to find the unrated movie we use K nearest neighbor model which measures the similarity of the user using distance metrics and try to fill the blanks and then finally try to come up with a recommendation.

TABLE I
Matrix Representation

Matrix	Movie A	Movie B	Movie C	Movie D	Movie E
User A	4	3		5	
User B	3		4		3
User C	5	2			4
User D	2	1	4		
User E	3			2	4

A. Dataset Description

The collaborative filtering is widely used in many recommendation systems. As previously described in other section content based and collaborative filtering. The published online data set is obtained from 'GroupLens' which is a research project from university of Minnesota. The data was collected from September 1997 to October 2018. There are four different data types such as user ratings, tags, movies and links. Following that the dataset consists 100,000 plus ratings and movies and 10000 users. The data types that are available have been taken over a huge period of time where individual users have their unique user ID. The individual users have multiple movie ratings, tags and links with different genres. [3]

B. Algorithm and Methodology Explanation

The recommendation system that has been explained in the previous sections try to make recommendations based on users rating data. However, it can be implemented on smaller datasets which would probably provide higher root mean squared error. Hence, in order to reduce the error we have to apply dimensionality reduction technique to extract preferences and choices from the dataset. The reasons for applying dimensionality reduction is to discover hidden correlations from the dataset and removal of irrelevant and noisy attributes which are of no use. This makes the visualization and interpretation of the data easier also makes data processing and storage easier. Matrix factorization has received higher exposure because it acts as an unsupervised learning technique. It is also widely used as a recommendation system because it deals better in terms of scalability and variable decomposition.

The platform that has been used here is Databricks community edition where I have created a spark cluster of version 6.2 and after creating the cluster the movielens dataset is imported. Following which the pre-processing of data is done where we split the training and testing the data. The reason for using matrix factorization of singular value decomposition is that it can decompose a matrix into its best lower rank approximation of the actual matrix. Finally, after feature extraction and construction of rating matrix we apply the machine learning algorithm of K nearest neighbour (K-NN) and then obtain output as recommendation of movies. K-NN input contains the closest K trained examples and finds the top N recommendations in the feature space. Below diagram is the pipeline representation of my methodology.

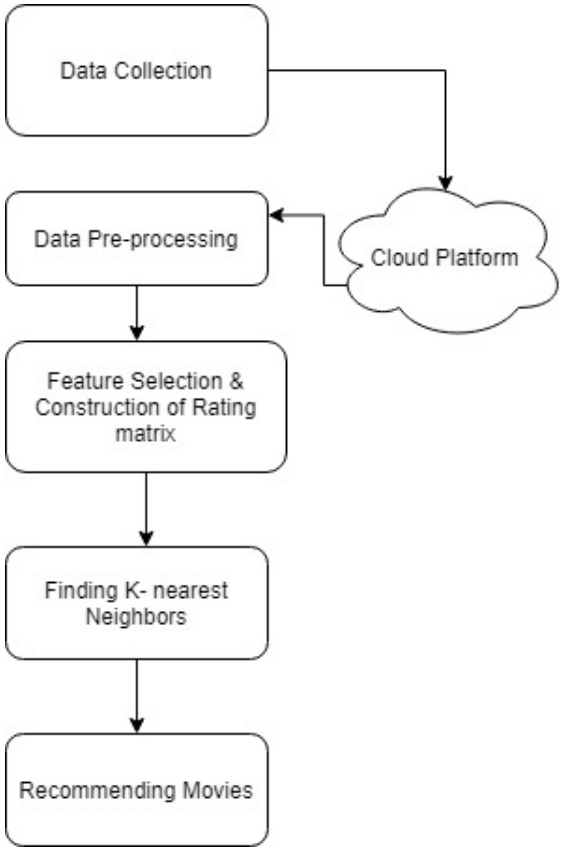

Fig. 1. Recommendation System Pipeline

IV. Implementation with Code

As we know the data set has been obtained from Movielens. Now the obtained data is imported to databricks notebook in a spark cluster and visualised. [4]

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

from pyspark.sql.types import FloatType, IntegerType
# data from MovieLens|GroupLens(http://files.grouplens.org/datasets/movielens/ml-latest-small.zip)
# File location and type
file_location = "/FileStore/tables/movies.csv"
file_type = "csv"
display(df)
movie_data = df.toPandas()
movie_data.head()
```

Fig. 2. Importing data and visualizing into notebook

After the data is imported and visualised we pre-process the data for training and testing purposes.

```
# Data Preprocessing for huge dataset (However here Not Required)
# to select only those movies whose id is present in movie_data
# movie_ratings = movie_ratings[movie_ratings['movieId'].isin(movie_data.index)]
```

```
def favMovie(userId, N):
    userRatings = movie_ratings[movie_ratings.userId==userId]
    sortedRatings = pd.DataFrame.sort_values(userRatings,['rating'] ,ascending=[0])[:N]
    sortedRatings['Title'] = sortedRatings['movieId'].apply(movieTitle)
    sortedRatings['Genre'] = sortedRatings['movieId'].apply(movieGenre)
    return sortedRatings
favMovie(1, 10)
```

	userId	movieId	rating	Title	Genre
4	1	1172	4.0	Cinema Paradiso (Nuovo cinema Paradiso) (1989)	Drama
13	1	2105	4.0	Tron (1982)	Action|Adventure|Sci-Fi
12	1	1953	4.0	French Connection, The (1971)	Action|Crime|Thriller
8	1	1339	3.5	Dracula (Bram Stoker's Dracula) (1992)	Fantasy|Horror|Romance|Thriller
19	1	3671	3.0	Blazing Saddles (1974)	Comedy|Western
1	1	1029	3.0	Dumbo (1941)	Animation|Children|Drama|Musical
2	1	1061	3.0	Sleepers (1996)	Thriller
14	1	2150	3.0	Gods Must Be Crazy, The (1980)	Adventure|Comedy
17	1	2455	2.5	Fly, The (1986)	Drama|Horror|Sci-Fi|Thriller
0	1	31	2.5	Dangerous Minds (1995)	Drama

Fig. 3. Pre-processing of Data

Now we set up the rating matrix in order to count avoid the blank or unrated movies.

```
## Data Preprocessing to obtain less sparse matrix for huge dataset(However here Not Required)

    ## Take only those movies which are seen by more than 10 users
#movie_ratings = movie_ratings[movie_ratings.index.isin(userPerMovieID[userPerMovieID > 10].index)]
#movie_ratings.shape
```

```
userMovieRatingMatrix = pd.pivot_table(movie_ratings, index=['userId'],columns=['movieId'] ,values='ratin
g')
userMovieRatingMatrix.head(10)
```

movieId	1	2	3	4	5	6	7	8	9	10	...	161084	161155	161594	161830	161918	161944	1623
userId																		
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7	3.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	3.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
10	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN

10 rows × 9066 columns

Fig. 4. Setting up rating matrix

Now we will be implementing K-Nearest neighbors into the notebook and wrapping it in a function.

```
# Wrapping it up in a function
def nearestNeighbours(user, K=10):
    allusers = pd.DataFrame(userMovieRatingMatrix.index)
    allusers = allusers[allusers.userId != user]
    allusers['distance'] = allusers['userId'].apply(lambda x: distance(user, x))
    KNearestUsers = allusers.sort_values(['distance'], ascending=True)['userId'][:K]
    return KNearestUsers

KNearestNeighbours = nearestNeighbours(1,5)
KNearestNeighbours
```

```
579    580
451    452
14      15
563    564
72      73
Name: userId, dtype: int64
```

Fig. 5. Implementing K-nearest neighbors

Finally, after the implementation of K-NN successfully. we take the nearest neighbors rating and get an average rating based on seen movie by the active user. Also, we remove the movies which are duplicated and seen by the user. Later, after implementation we wrap it in a function and get the out as the top recommendations for the users.

```
# Nearest Neighbours ratings
NNratings = userMovieRatingMatrix[userMovieRatingMatrix.index.isin(KNearestNeighbours)]
NNratings
#Getting the average rating of each movie seen by Nearest Neighbours of active user
avgRating = NNratings.apply(np.nanmean).dropna()
avgRating.head()
# Removing the movies which are already seen by user
moviesAlreadySeen = userMovieRatingMatrix.transpose()[user].dropna().indexmoviesAlreadySeen
avgRating = avgRating[~avgRating.index.isin(moviesAlreadySeen)]
N=3
topNMovieId = avgRating.sort_values(ascending=False).index[:N]
topNMovieId
pd.Series(topNMovieId).apply(movieTitle)

pd.Series(topNMovieId).apply(movieGenre)
# Wrapping it up in a function
def topN(user,N=3):
    KnearestUsers = nearestNeighbours(user)
    NNRatings = userMovieRatingMatrix[userMovieRatingMatrix.index.isin(KnearestUsers)]
    avgRating = NNRatings.apply(np.nanmean).dropna()
    moviesAlreadySeen = userMovieRatingMatrix.transpose()[user].dropna().index
    avgRating = avgRating[~avgRating.index.isin(moviesAlreadySeen)]
    topNMovieId = avgRating.sort_values(ascending=False).index[:N]
    return pd.DataFrame({'Movie':pd.Series(topNMovieId).apply(movieTitle),
                         'Genre':pd.Series(topNMovieId).apply(movieGenre)})
# To remove the RunTimeWarning error
import warnings
warnings.filterwarnings("ignore", category=RuntimeWarning)
topN(3,5)
```

Fig. 6. Getting recommended output

	Movie	Genre
0	Marathon Man (1976)	Crime|Drama|Thriller
1	Shogun Assassin (1980)	Action|Adventure
2	Love and Death (1975)	Comedy
3	George of the Jungle (1997)	Children|Comedy
4	Unvanquished, The (Aparajito) (1957)	Drama

Fig. 7. Recommendations

V. Experiment Results

The results have been compared with the individual models also as mentioned in the earlier sections every dataset contains test and train data separately. The system is based on using collaborative filtering which implies user rating data. The representation of rating data is in the form of a matrix where the movies are represented in columns and users in rows. However, there are instances where there are blank cells that represent no ratings given in the matrix. Now when we compare the recommendations obtained from all the three implementations which is content-based filtering, memory-based collaborative filtering and our model of matrix factorization. we can find that our model has a decent accuracy as compared to the other two. The table below explains and compares the accuracy obtained by each recommendation system.

TABLE II
COMPARISON ACCURACY TABLE OF RECOMMENDATION SYSTEM

Recommendation Model used	Model Accuracy
Content Based Filtering	0.5986
Item-item based collaborative filtering	0.4122
User-user based collaborative filtering	0.6991
Matrix factorization using K-NN	0.8736

Therefore, after comparison of various recommendation systems. We can observe that the Model-based collaborative system with matrix factorisation has the highest accuracy of recommendations for an individual based on user rating and genre. Visualizing data based on the recommendation produced by the output.

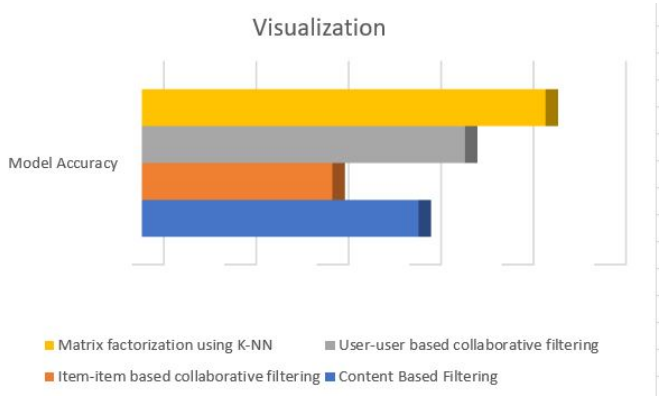

Fig. 8. Visualizing the accuracy

VI. Conclusion

This recommendation system with the combination of many evaluation metrics makes itself relevant and useful. This paper focused on the idea behind matrix factorisation and implementation on K nearest neighbor and finding the ”N” recommendations. This is a more personalised approach done via model based collaborative filtering The performance of the model however has the scope of improvement. Additional factors from the dataset which may provide more insights can be considered in the future. Nevertheless, identifying correlations between customer or a user satisfaction to give recommendations with various different factors can also be a way in the future.

VII. Software Packages

The source code is written in Python and the platform that is being used is Databricks community edition. The ipynb file is uploaded to the authors github account. [5] The software package also consists of individual dataset and its explanation.

References and Footnotes

References

[1] K. Liao. Prototyping a recommender system step by step part 1: KNN item-based collaborative filtering. [Online]. Available: https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-1-knn-item-based-collaborative-filtering-637969614ea

[2] J. Le. The 4 recommendation engines that can predict your movie tastes. [Online]. Available: https://medium.com/@james_aka_yale/the-4-recommendation-engines-that-can-predict-your-movie-tastes-bbec857b8223

[3] F. M. Harper and J. A. Konstan, “The MovieLens datasets: History and context,” vol. 5, no. 4, pp. 1–19. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2866565.2827872

[4] R. Jain, “Rohit9314/movie-recommendation-system,” original-date: 2018-06-23T09:10:20Z. [Online]. Available: https://github.com/Rohit9314/Movie-Recommendation-System

[5] DebdeepLazyCoder/data-analytics-assignment. [Online]. Available: https://github.com/DebdeepLazyCoder/Data-Analytics-Assignment