Week 9 Assignment - Recommender System

Name: Kesav Adithya Venkidusamy
Course: DSC630 - Predictive Analytics
Instructor: Fadi Alsaleem

Using the small MovieLens data set, create a recommender system that allows users to input a movie they like (in the data set) and recommends ten other movies for them to watch. In your write-up, clearly explain the recommender system process and all steps performed. If you are using a method found online, be sure to reference the source. You can use R or Python to complete this assignment. Submit your code and output to the submission link. Make sure to add comments to all of your code and to document your steps, process, and analysis.

Recommender System

Recommendation engines are a subclass of machine learning which generally deal with ranking or rating products / users. Loosely defined, a recommender system is a system which predicts ratings a user might give to a specific item. These predictions will then be ranked and returned back to the user.

Recommender systems are often seen as a “black box”, the model created by the large companies are not very easily interpretable. The results which are generated are often recommendations for the user for things that they need / want but are unaware that they need / want it until they’ve been recommended to them.

Types of Recommender System

Recommender System is different types:

Collaborative Filtering: Collaborative Filtering recommends items based on similarity measures between users and/or items. The basic assumption behind the algorithm is that users with similar interests have common preferences.

Content-Based Recommendation: It is supervised machine learning used to induce a classifier to discriminate between interesting and uninteresting items for the user.

In this exercise, we are going to use Collaborative Filtering to recommend the movies similar to what user has watched. Basically, user will provide the movie that he has watched and system will suggest the list of movies to watch.

Step 1: Importing all the libraries required for data processing

In [2]:
```
## Importing libraries required for this assignment
import numpy as np
import pandas as pd
import sklearn
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
## Ignore the warnings
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

In [4]:
```
## Display all columns in pandas dataframe
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
```

Step 2: As as Load the Dataset into dataframe

As an initial step, we will create 2 dataframes; ratings_df -> one of the ratings given to the movies by users and movies_df -> list of the movies and genres.

In [5]:
```
## Load the ratings data into a dataframe
ratings_df = pd.read_csv("ratings.csv")
ratings_df.head()
```

Out[5]:

	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

In [6]:
```
## Load the movies data into a dataframe
movies_df = pd.read_csv("movies.csv")
movies_df.head()
```

Out[6]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

Step 3: Calculating the stats on the datasets

In [7]:
```
##Calculating the total number of records present in ratings_df
## Total number of unique movies from ratings_df
## Total number of unique users from ratings_df
n_ratings = len(ratings_df)
n_movies = len(ratings_df['movieId'].unique())
n_users = len(ratings_df['userId'].unique())
```

In [8]:
```
## Printing the number of ratings. unique movieid, unique users and average user and movies
print(f"Number of ratings: {n_ratings}")
print(f"Number of unique movieId's: {n_movies}")
print(f"Number of unique users: {n_users}")
print(f"Average ratings per user: {round(n_ratings/n_users, 2)}")
print(f"Average ratings per movie: {round(n_ratings/n_movies, 2)}")
```

```
Number of ratings: 100836
Number of unique movieId's: 9724
Number of unique users: 610
Average ratings per user: 165.3
Average ratings per movie: 10.37
```

In the above step, we calculating the total number of ratings given to the movies, number of unique movies present in the dataset, number of unique users present in the rating dataframe and average ratings per user and movies.

On an average, a user has provided 165.3 ratings for the movies and each movie has recieved 10.37 ratings from the users

In [9]:
```
## Calculate the count of movies watched by user frequency
user_freq = ratings_df[['userId', 'movieId']].groupby('userId').count().reset_index()
```

```
user_freq.columns = ['userId', 'n_ratings']
user_freq.head()
```

Out[9]:

	userId	n_ratings
0	1	232
1	2	29
2	3	39
3	4	216
4	5	44

In [10]:

```
# Find Lowest and Highest rated movies:
mean_rating = ratings_df.groupby('movieId')[['rating']].mean()
# Lowest rated movies
lowest_rated = mean_rating['rating'].idxmin()
movies_df.loc[movies_df['movieId'] == lowest_rated]
```

Out[10]:

	movieId	title	genres
2689	3604	Gypsy (1962)	Musical

In [11]:

```
# Highest rated movies
highest_rated = mean_rating['rating'].idxmax()
movies_df.loc[movies_df['movieId'] == highest_rated]
```

Out[11]:

	movieId	title	genres
48	53	Lamerica (1994)	Adventure|Drama

In [12]:

```
# show number of people who rated movies rated movie highest
ratings_df[ratings_df['movieId']==highest_rated]
# show number of people who rated movies rated movie lowest
ratings_df[ratings_df['movieId']==lowest_rated]
```

Out[12]:

	userId	movieId	rating	timestamp

	userId	movieId	rating	timestamp
13633	89	3604	0.5	1520408880

In [13]:
```
## the above movies has very low dataset. We will use bayesian average
movie_stats = ratings_df.groupby('movieId')[['rating']].agg(['count', 'mean'])
movie_stats.columns = movie_stats.columns.droplevel()
```

In the above steps, we calculated the count of ratings provided by each user present in the dataset, movies that recieved lowest and highest ratings from the users. Fianlly, we have also shown number of users rated the lowestest and highest rating movies.

Step 4: Creating user and movie matrix using csr_matrix available in scipy.sparse library

In [14]:
```
# Importing Library to create user-item matrix using scipy csr matrix
from scipy.sparse import csr_matrix
```

In [15]:
```
## Function to create user item matrix
def create_matrix(df):

    N = len(df['userId'].unique())
    M = len(df['movieId'].unique())

    # Map Ids to indices
    user_mapper = dict(zip(np.unique(df["userId"]), list(range(N))))
    movie_mapper = dict(zip(np.unique(df["movieId"]), list(range(M))))

    # Map indices to IDs
    user_inv_mapper = dict(zip(list(range(N)), np.unique(df["userId"])))
    movie_inv_mapper = dict(zip(list(range(M)), np.unique(df["movieId"])))

    user_index = [user_mapper[i] for i in df['userId']]
    movie_index = [movie_mapper[i] for i in df['movieId']]

    X = csr_matrix((df["rating"], (movie_index, user_index)), shape=(M, N))

    return X, user_mapper, movie_mapper, user_inv_mapper, movie_inv_mapper
```

In [16]:
```
## Calling the create matrix function and assign to the variables
X, user_mapper, movie_mapper, user_inv_mapper, movie_inv_mapper = create_matrix(ratings_df)
```

In this step, we created a function to build matrix between users and movies. Initially, the length of users and movies present in the dataset has been taken. Then identifiers have been assigned to user id and movie id after removing the duplicates present in the dataset.

CSR Matrix has been created with the list of user id and movie ids present in the dataset. Upon creating the matrix, the following values are returned from the function.

1. X: Matrix between movie ids and user ids.
2. user_mapper: Here, unique id has been assigned to each user id and created dictionary of key value pairs.
3. movie_mapper: Here, unique id has been assigned to each movie id and created dictionary of key value pairs.
4. user_inv_mapper: Mapping indices to each user id
5. movie_inv_mapper: Mapping indices to each movie id

Step 5: Function to find similar movies using KNN algorithm

In [29]:
```
## Importing the library to calculate the similar movies using KNN
from sklearn.neighbors import NearestNeighbors
```

In [30]:
```
## Function to find the similar movies using KNN
def find_similar_movies(movie_id, X, k, metric='cosine', show_distance=False):

    neighbour_ids = []

    movie_ind = movie_mapper[movie_id]
    movie_vec = X[movie_ind]
    k+=1
    kNN = NearestNeighbors(n_neighbors=k, algorithm="brute", metric=metric)
    kNN.fit(X)
    movie_vec = movie_vec.reshape(1,-1)
    neighbour = kNN.kneighbors(movie_vec, return_distance=show_distance)
    for i in range(0,k):
        n = neighbour.item(i)
        neighbour_ids.append(movie_inv_mapper[n])
    neighbour_ids.pop(0)
    return neighbour_ids
```

A function has been created to find similar movies based on the movie id provided as input to the user. Following are the parameters used as a input to the function.

1. movie_id: Movie id provided by the user; This is the movie user has watched and he wants movies similar to this
2. X: CSR Matrix created between user ids and movie ids
3. k: Number of neighbors based on the movie id requested by the user
4. metric: Cosine similarity measures the similarity between two vectors of an inner product space. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction. It is often used to measure document similarity in text analysis.

Upon calling the function, it calculates the neighbors based on the user input (k) values and returns all the adjacent movie ids.

Step 6: Calculating movie watch list based on watched movie

In [31]:
```
## Creating dictionary with movie id as key and title as value
movie_titles = dict(zip(movies_df['movieId'], movies_df['title']))
```

In [32]:
```
## Get imput from user for movie id
min_movie_id = min(movie_mapper.keys())
max_movie_id = max(movie_mapper.keys())
print("The minimum and maximum movie id {} and {}".format(min_movie_id, max_movie_id))
```

```
The minimum and maximum movie id 1 and 193609
```

In [45]:
```
## Get the input movie id from the user
while True:
    print("\nPlease enter the movie id between {} and  {}: ".format(min_movie_id, max_movie_id))
    movie_id = int(input())
    if int(movie_id) in movie_mapper.keys():
        print("The movie id {} is present in the mapper list".format(movie_id))
        similar_ids = find_similar_movies(movie_id, X, k=10)
        movie_title = movie_titles[movie_id]
        print(f"\n\033[1mSince you watched the movie \'{movie_title}\', below are some other recommendations\033[0m")
        for i in similar_ids:
            print(movie_titles[i])
        print("\nDo you want to check for other movies (Y/N):")
        user_yn = input()
        if user_yn.upper() == 'Y':
            continue
        else:
            break
    else:
        print("The movie id {} is not present in the mapper list".format(movie_id))
```

```
        print("Please enter someother value")
```

```
Please enter the movie id between 1 and  193609:

The movie id 1 is present in the mapper list

Since you watched the movie 'Toy Story (1995)', below are some other recommendations
Toy Story 2 (1999)
Jurassic Park (1993)
Independence Day (a.k.a. ID4) (1996)
Star Wars: Episode IV - A New Hope (1977)
Forrest Gump (1994)
Lion King, The (1994)
Star Wars: Episode VI - Return of the Jedi (1983)
Mission: Impossible (1996)
Groundhog Day (1993)
Back to the Future (1985)

Do you want to check for other movies (Y/N):

Please enter the movie id between 1 and  193609:

The movie id 1000 is not present in the mapper list
Please enter someother value

Please enter the movie id between 1 and  193609:

The movie id 2 is present in the mapper list

Since you watched the movie 'Jumanji (1995)', below are some other recommendations
Lion King, The (1994)
Mrs. Doubtfire (1993)
Mask, The (1994)
Jurassic Park (1993)
Home Alone (1990)
Nightmare Before Christmas, The (1993)
Aladdin (1992)
Beauty and the Beast (1991)
Ace Ventura: When Nature Calls (1995)
Santa Clause, The (1994)

Do you want to check for other movies (Y/N):
```

A custom function has been created as above to get input from user on movie id; The movie id has been passed to find_similar_movies function which returns the list of 10 movies similar to the movie watched by the user.

If user wants to continue finding the list based on other movie, he just provide the input as "Y" and continue with this search. If he decides to end the search, he just provide the input as "N".

In addition, if the movie id provided as input by the user is not present in the list, the function will ask the user to provide the correct id.

Reference

https://www.geeksforgeeks.org/recommendation-system-in-python/

In [ ]: