Author: Anjani Bonda

Course: DSC630 - Predictive Analytics

Date: 05 Nov 2022

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.

In this exercise, we are going to use Collaborative Filtering to recommend the movies for the user.

Step 1: Import all the libraries required for data processing

In [1]:
```
## Import required libraries.
import numpy as np
import pandas as pd
import sklearn
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
## Ignore warnings
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

In [3]:
```
## Display all columns in pandas dataframe
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
```

Step 2: Load the movielens dataset

We will create 2 dataframes. 1. ratings_df -> movie ratings given by users. 2. movies_df -> list of movies and genres.

In [4]:
```
## Load the ratings data into a dataframe
ratings_df = pd.read_csv("ratings.csv")
ratings_df.head()
```

Out[4]:

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

Out[5]:

	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

Step3: Calculate stats/metrics on the above datasets.

In [6]:
```
## Calculate the total # of records present in ratings_df
## Total # of unique movies from ratings_df
## Total # 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 [7]:
```
## Print the # of ratings. unique movieid, unique users and average user and mo
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 calculated the total # of ratings given to the movies, # of unique movies in the dataset, # of unique users in the ratings dataframe and average ratings per user and movies.

On an average, a user has provided 165.3 # of ratings for the movies and each movie has received 10.37 # of ratings from the users.

In [8]:
```
## Calculate the count of movies watched by user frequency
```

```
user_freq = ratings_df[['userId', 'movieId']].groupby('userId').count().reset_i
user_freq.columns = ['userId', 'n_ratings']
user_freq.head()
```

Out[8]:

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

In [9]:

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

Out[9]:

	movieId	title	genres
2689	3604	Gypsy (1962)	Musical

In [10]:

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

Out[10]:

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

In [13]:

```
# show users who rated movies highest
ratings_df[ratings_df['movieId']==highest_rated]
```

Out[13]:

	userId	movieId	rating	timestamp
13368	85	53	5.0	889468268
96115	603	53	5.0	963180003

In [12]:

```
# show users who rated movies lowest
ratings_df[ratings_df['movieId']==lowest_rated]
```

Out[12]:

	userId	movieId	rating	timestamp
13633	89	3604	0.5	1520408880

In [14]:

```
# 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. Finally, we have also shown number of users rating the lowest and highest rating movies.

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

In [15]:
```
# Import Library to create user-item matrix using scipy csr matrix
from scipy.sparse import csr_matrix
```

In [16]:
```
## 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 [17]:
```
## Call the create matrix function and assign to the variables
X, user_mapper, movie_mapper, user_inv_mapper, movie_inv_mapper = create_matrix
```

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.

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

Step 5: Function to find similar movies using KNN algorithm

In [18]:
```
## Import the library to calculate the similar movies using KNN
from sklearn.neighbors import NearestNeighbors
```

In [19]:
```
## 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.

movie_id: Movie id provided by the user; This is the movie user has watched and he wants movies similar to this X: CSR Matrix created between user ids and movie ids k: Number of neighbors based on the movie id requested by the user 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: Calculate movie watch list based on watched movie

In [20]:

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

In [21]:

```
## Get user input 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_mov
```

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

In [22]:

```
## Get the user input of movie id.
while True:
    print("\nPlease enter the movie id between {} and  {}: ".format(min_movie_i
    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 a
        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_
        print("Please enter someother value")
```

```
Please enter the movie id between 1 and  193609:
2
The movie id 2 is present in the mapper list

Since you watched the movie 'Jumanji (1995)', below are some other recommendat
ions
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):
N
```

A custom function has been created as above to get user input 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 would just provide the input as "Y" and continue the search. If he decides to end the search, he would just provide the input as "N".

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

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

In [ ]: