MOVIE RECOMENDATION SYSTEM

Recommender System is 8 system that seeks to predict or fier preferences according to the user' choices. Recommender systems are stlczed in a variety of areas including movies, music, news, books, research articles, search queries, social tags, and products in general Recommender systems produce a ist of recommendations in any of the two ways ~

Collaborative filtering: Collaborative filtering approaches build » model from the user's past behavior (Le. tems purchased or searched by the user) as well as similar decisions made by other users. This model is then used to predict stems (or ratings for tems) that users may have an terest in.

Content-based filtering: Content-based filtering approaches uses a series of discrete characteristics of an item in order to recommend additional items with similar properties. Content-based fering methods are totally based on a description of the item and a profile of the user's preferences. recommends ftems based on the users past preferences. Let's develop 8 basic recommendation system using Python and Pandas.

Let's develop a basic recommendation system by suggesting items that are most similar to a particular tem, in this case, movies. just tells what movies/items are most similar to the user's movie choice.

IMPORT LIBRARIES

```
import numpy as np
import pandas as pd
import sklearn
import matplotlib.pyplot as plt
import seaborn as sns
```

IMPORT DATASET

```
ratings = pd.read_csv("ratings-lab 10.csv")
ratings.head()
movies = pd.read_csv("movies-lab 10.csv")
movies.head()
```

	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

```
n_ratings = len(ratings)
n_movies = len(ratings['movieId'].unique())
n_users = len(ratings['userId'].unique())
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)}")
user_freq = ratings[['userId', 'movieId']].groupby('userId').count().reset_index()
user_freq.columns = ['userId', 'n_ratings']
user_freq.head()
```

```
Number of ratings: 100836
```

```
# Find Lowest and Highest rated movies:
mean_rating = ratings.groupby('movieId')[['rating']].mean()
# Lowest rated movies
lowest_rated = mean_rating['rating'].idxmin()
movies.loc[movies['movieId'] == lowest_rated]
# Highest rated movies
highest_rated = mean_rating['rating'].idxmax()
movies.loc[movies['movieId'] == highest_rated]
# show number of people who rated movies rated movie highest
ratings[ratings['movieId']==highest_rated]
# show number of people who rated movies rated movie lowest
ratings[ratings['movieId']==lowest_rated]
```

	userId	movieId	rating	timestamp
13633	89	3604	0.5	1520408880

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

```
from scipy.sparse import csr_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
X, user_mapper, movie_mapper, user_inv_mapper, movie_inv_mapper = create_matrix(ratings)
from sklearn.neighbors import NearestNeighbors
"""
Find 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
movie_titles = dict(zip(movies['movieId'], movies['title']))
movie_id = 3
similar_ids = find_similar_movies(movie_id, X, k=10)
movie_title = movie_titles[movie_id]
print(f"Since you watched {movie_title}")
for i in similar_ids:
    print(movie_titles[i])
```

```
Since you watched Grumpier Old Men (1995)
Grumpy Old Men (1993)
Striptease (1996)
Nutty Professor, The (1996)
Twister (1996)
Father of the Bride Part II (1995)
Broken Arrow (1996)
Bio-Dome (1996)
Truth About Cats & Dogs, The (1996)
Sabrina (1995)
Birdcage, The (1996)
```