Recommender Systems

**Prof. Dietmar Jannach Alpen‐Adria‐Universität Klagenfurt**

**Assignment 3**

**Implementing a collaborative recommender**

Implement a user‐based nearest neighbor recommendation algorithm. Write a program that:

1. Accepts a user ID as an input (on the console),
2. Then shows the titles and genres of up to 15 movies that this user has rated1,
3. Then displays the 10 movies with the highest predicted relevance score2 according to the nearest neighbor technique3.

Use the MovieLens1M dataset for testing your program. Structure your program code in functions and/or classes. Implement appropriate error handling procedures. Do not use any library for neither implementing the nearest‐neighbor method nor measuring user similarity.

Check the recommendations for plausibility by inspecting the recommendations.

Run experiments with different neighborhood sizes and look at the effects of changing this parameter.

CODE:

import pandas as pd  
import numpy as np  
import zipfile  
  
def load\_data(zip\_path):  
 with zipfile.ZipFile(zip\_path, 'r') as z:  
 with z.open('ml-latest-small/ratings.csv') as f:  
 ratings = pd.read\_csv(f)  
 with z.open('ml-latest-small/movies.csv') as f:  
 movies = pd.read\_csv(f)  
 return ratings, movies  
  
def show\_user\_movies(ratings, movies, user\_id):  
 user\_ratings = ratings[ratings['userId'] == user\_id]  
 user\_movies = user\_ratings.merge(movies, on='movieId')  
 print("Movies rated by User:", user\_id)  
 print(user\_movies[['title', 'genres']].head(15))  
  
def calculate\_similarity(ratings, user\_id):  
 matrix = ratings.pivot\_table(index='userId', columns='movieId', values='rating')  
 user\_ratings = matrix.loc[user\_id].dropna()  
 others = matrix.drop(user\_id)  
 similarities = {}  
 for other\_id, other\_ratings in others.iterrows():  
 common = other\_ratings.dropna().index.intersection(user\_ratings.index)  
 if len(common) >= 3:  
 diff\_user = user\_ratings[common] - other\_ratings[common]  
 sim = 1 - (np.sqrt(np.sum(diff\_user\*\*2)) / len(common))  
 similarities[other\_id] = sim  
 return sorted(similarities.items(), key=lambda x: x[1], reverse=True)  
  
def predict\_scores(ratings, user\_id, top\_neighbors):  
 neighbors = calculate\_similarity(ratings, user\_id)[:top\_neighbors]  
 user\_unrated\_movies = ratings[ratings['userId'] == user\_id]['movieId'].unique()  
 all\_ratings = ratings[ratings['movieId'].isin(user\_unrated\_movies)]  
 predictions = {}  
 for movie\_id in user\_unrated\_movies:  
 weighted\_sum = 0  
 sim\_sum = 0  
 for neighbor\_id, similarity in neighbors:  
 neighbor\_rating = all\_ratings[(all\_ratings['userId'] == neighbor\_id) & (all\_ratings['movieId'] == movie\_id)]['rating']  
 if not neighbor\_rating.empty:  
 weighted\_sum += neighbor\_rating.iloc[0] \* similarity  
 sim\_sum += similarity  
 if sim\_sum != 0:  
 predictions[movie\_id] = weighted\_sum / sim\_sum  
 return predictions  
  
def recommend\_movies(movies, predictions):  
 movie\_scores = pd.DataFrame(list(predictions.items()), columns=['MovieID', 'PredictedRating'])  
 top\_movies = movie\_scores.sort\_values(by='PredictedRating', ascending=False).head(10)  
 recommended\_movies = movies[movies['movieId'].isin(top\_movies['MovieID'])]  
 return recommended\_movies  
  
  
def main():  
 zip\_path = r'C:\Users\user\Documents\Uni\Semester 2\Information Search & Recommendation Systems\HW\ml-latest-small.zip'  
 ratings, movies = load\_data(zip\_path)  
 try:  
 user\_id = int(input("Enter a user ID: "))  
 if user\_id not in ratings['userId'].values:  
 raise ValueError("User ID not found in the dataset.")  
  
 show\_user\_movies(ratings, movies, user\_id)  
  
 # Set a range for neighborhood sizes to test  
 min\_neighbors = 1  
 max\_neighbors = 20  
  
 # Run the experiment with different neighborhood sizes  
 for neighbors in range(min\_neighbors, max\_neighbors + 1):  
 print(f"\nTesting with {neighbors} neighbors:")  
 predictions = predict\_scores(ratings, user\_id, neighbors)  
 recommended\_movies = recommend\_movies(movies, predictions)  
 print("Recommended Movies:")  
 print(recommended\_movies[['title', 'genres']])  
  
 except ValueError as ve:  
 print(ve)  
 except Exception as e:  
 print(f"An error occurred: {e}")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

1 You can use *pandas.merge* to merge tables.

2 You can use *pandas.DataFrame.sample* to test your code (because of computation time).

3 You can use *pandas.pivot\_table* to create the User‐Item table.