coursework1

September 30, 2024

1 Coursework 1: Movie ratings

This is the first coursework of ECS7023P Programming for AI and Data Science, which counts 35% towards the final grade of the module. The coursework is graded out of 100 marks.

Deadline: Monday 30th September, 2024 - 11.59pm

Marking criteria: While the most important marking criterion will be for the code to achieve the expected objective and output, marks will also be given for partial or close solutions, whereas marks can be deducted for code that is overly complex, inefficient, difficult to understand and/or to maintain.

Use of packages: For this exercise, in addition to the built-in python functions and elements that we have seen in the lectures (see lecture notes), you can only import the **csv** and **json** packages. No other packages are allowed. You cannot use other packages such as **pandas**, you won't get any marks for a question if you use it.

How to submit: You will submit a completed Jupyter notebook file with your solutions, as well as a PDF version of the Jupyter notebook which includes the outputs of your code. You need to submit the python code that produces the required answers. Answers produced through means other than python code will not be deemed acceptable.

Note: This is an individual coursework, the solutions you submit need to be your own and developed on your own.

1.1 Dataset

For this exercise, you are given a dataset that contains information about a collection of movies, along with ratings assigned by 2,500 users to those movies.

The dataset contains two files: * movies.json: a JSON file with information about movies, including their ID, title, language, release date, country(ies) of origin and genre(s). * ratings.csv: a CSV file that contains an entry for each movie rating, where a user ID rates a movie ID with a rating on a likert scale from 1 to 5 at a particular time.

The **movieId** column in ratings.csv can be linked to the IDs within movie.json, so you know which specific movie a user is rating in each case.

Note: ratings.csv contains entries only for movies that each user has rated, i.e. for many movies, a user may not have entered any ratings so we don't have that information.

1.2 Exercises

1.2.1 Question 1

1. Who is the most active user with the largest number of ratings? Print the user ID and the number of ratings for this user. **(10 marks)**

[1]:
```
# Step 1: Importing necessary package at the start of the block
import csv

# Step 2: Defining a function that returns the most active user based on the␣
 ↪count of user ratings
def most_active_user ():
    user_ratings_count = {} # Empty dictionary to store the count of ratings␣
 ↪per user

    with open('ratings.csv', 'r') as ratings_file:
        ratings = csv.reader(ratings_file)

        for row in ratings:
            user_id = row[0]  # User ID is in the first column as shown by the␣
 ↪output of some previously executed code
            if user_id in user_ratings_count:
                user_ratings_count[user_id] += 1  # Increment the rating count␣
 ↪for the user
            else:
                user_ratings_count[user_id] = 1  # Initialize the rating if␣
 ↪they haven't been encountered yet

# Step 3: Initializing variables to find the most active user
    most_active_user_id = ''
    most_active_user_count = 0

# Step 4: Finding the most active user (the user with the highest count of␣
 ↪ratings) through this FOR loop
    for user_id, count in user_ratings_count.items():
        if count > most_active_user_count:
            most_active_user_id = user_id
            most_active_user_count = count

    return print ('Most active user ID:', most_active_user_id, '\nNumber of␣
 ↪ratings:', most_active_user_count)

# Step 5: Calling the previously defined function
most_active_user ()
```

```
Most active user ID: 8659
Number of ratings: 3023
```

1.2.2 Question 2

2. What is the user who, having rated at least 25 movies, has the overall lowest rating average? Print the ID of this user and their rating average. **(10 marks)**

[2]:
```
# Step 1: Importing necessary package at the start of the block
import csv

# Step 2: Defining a function that returns the user with the lowest rating␣
 ↪average
def user_lowest_rating_average():
    user_ratings_count = {}
    user_ratings_sum = {}

    with open('ratings.csv', 'r') as ratings_file:
        ratings = csv.reader(ratings_file)
        next(ratings)  # Skipping the header ('rating' column) as I ran into an␣
 ↪error when converting rating into a float data type

        for row in ratings:
            user_id = row[0]
            rating = float(row[2])

            if user_id in user_ratings_count:
                user_ratings_count[user_id] += 1
                user_ratings_sum[user_id] += rating
            else:
                user_ratings_count[user_id] = 1
                user_ratings_sum[user_id] = rating

# Step 3: Initializing variables to find the user with the lowest rating average
    lowest_avg_user_id = ''
    lowest_avg_rating = float('inf')

# Step 4: Finding the user with the lowest rating average through this FOR loop
    for user_id in user_ratings_count:
        if user_ratings_count[user_id] >= 25:
            average_rating = user_ratings_sum[user_id] /␣
 ↪user_ratings_count[user_id]
            if average_rating < lowest_avg_rating:
                lowest_avg_rating = average_rating
                lowest_avg_user_id = user_id

    if lowest_avg_user_id != '' :
        return print('User ID with the lowest average rating (at least 25␣
 ↪ratings):', lowest_avg_user_id, '\nAverage Rating:', lowest_avg_rating)
    else:
        return print("No user found with at least 25 ratings.")
```

```
# Step 5: Calling the previously defined function
user_lowest_rating_average()
```

```
User ID with the lowest average rating (at least 25 ratings): 5228
Average Rating: 0.8343023255813954
```

1.2.3 Question 3

3. Given a year and a country as input, produce the statistics of genres for movies released in that year and country. To show the output of your code, print the results for 1995 as the input year and GB as the input country. **(10 marks)**

[3]:
```
# Step 1: Importing necessary package at the start of the block
import json

# Step 2: Defining a function that returns the statistics on movie genres for a␣
 ↪specific year and country
def genre_statistics(year, country):
    genre_count = {}  # Dictionary to store the count of genres

    with open('movies.json', 'r') as movies_file:
        for line in movies_file:
            movie = json.loads(line)  # Parse each movie as a JSON object
            movie_year = movie['releasedate'][:4]  # Extract year from the␣
 ↪'releasedate' field
            movie_country = movie['countries']  # Get the list of countries

# Step 3: Checking if the movie matches the year and contains the desired␣
 ↪country
            if movie_year == year and country in movie_country:
                for genre in movie['genres']:  # Loop through the genres of the␣
 ↪movie
                    if genre in genre_count:
                        genre_count[genre] += 1  # Increment count if genre␣
 ↪exists
                    else:
                        genre_count[genre] = 1  # Initialize count if genre␣
 ↪doesn't exist

# Step 4: Printing the statistics
    if genre_count:
        print('Statistics of genres for movies released in', year, 'in',␣
 ↪country, ':')
```

```
# Step 5: Sorting genres by count in descending order, since the first run of␣
 ↪this code was hideous and not easily readable
        for genre, count in sorted(genre_count.items(), key=lambda item:␣
 ↪item[1], reverse=True):
            print(genre + ': ' + str(count))
    else:
        print('No movies found for', year, 'in', country)

    return

# Step 6: As requested by the question: calling the previously defined function
#& printing the results for 1995 as the input year and GB as the input country:
genre_statistics('1995', 'GB')
```

```
Statistics of genres for movies released in 1995 in GB :
Drama: 37
Romance: 20
Comedy: 19
Thriller: 7
Action: 5
War: 5
Crime: 5
Adventure: 4
History: 4
Documentary: 4
Foreign: 4
TV Movie: 4
Horror: 4
Mystery: 3
Fantasy: 2
Family: 2
Animation: 2
Science Fiction: 2
Western: 1
```

1.2.4 Question 4

4. What is the title of the movie with the largest number of 3.5 ratings? How many 3.5 ratings does it have? **(15 marks)**

[4]:
```
# Step 1: Importing necessary packages at the start of the block
import csv
import json

# Step 2: Defining a function that returns the movie with the largest number of␣
 ↪x ratings (aiming to make it a dynamic function)
def movie_with_most_ratings(rating_value):
    rating_count = {}  # Dictionary to count ratings for each movie
```

```
    movie_titles = {}  # Dictionary to store movie titles

# Step 3: Reading Ratings csv File and Counting only ratings that match the␣
 ↪specified rating value (x)
    with open('ratings.csv', 'r') as ratings_file:
        ratings = csv.reader(ratings_file)
        next(ratings)  # Skip header as error was encountered

        for row in ratings:
            user_id = row[0]
            movie_id = row[1]
            rating = float(row[2])

            if rating == rating_value:
                if movie_id in rating_count:
                    rating_count[movie_id] += 1
                else:
                    rating_count[movie_id] = 1

# Step 4: Reading Movie JSON File to link movie IDs to titles and storing them
    with open('movies.json', 'r') as movies_file:
        for line in movies_file:
            movie = json.loads(line)
            movie_id = movie['id']  # Since movie ID column is titled as 'id'␣
 ↪in our json file
            title = movie['title']  # Since movie title column is titled as␣
 ↪'title' in our json file

            movie_titles[movie_id] = title # Store the title using the movie_id

# Step 5: Initializing variables to find the movie with the largest number of␣
 ↪(x) ratings
    most_ratings_movie_id = ''
    most_ratings_count = 0

    for movie_id, count in rating_count.items():
        if count > most_ratings_count:
            most_ratings_count = count
            most_ratings_movie_id = movie_id

# Step 6: Printing the result
    if most_ratings_movie_id  != '':
        movie_title = movie_titles[most_ratings_movie_id]
        print('Movie Title:', movie_title, '\nNumber of', rating_value,␣
 ↪'Ratings:', most_ratings_count)
    else:
        print('No ratings of', rating_value, 'found.')
```

```
# Step 7: Calling the function with a rating value of 3.5 as per the question:
movie_with_most_ratings(3.5)
```

```
Movie Title: Monsoon Wedding
Number of 3.5 Ratings: 654
```

1.2.5 Question 5

5. Write a python function which, given one or more countries as input parameter, produces a list of the top 5 movie titles with the highest average rating that match the country/ies. As an example to show your code, print the output for GB and US as the input countries. Note: the list of countries has to be part of the movie's countries, but not necessarily an exact match, e.g. a movie with GB, US, DE would be a match for GB, US as input parameter. **(15 marks)**

[5]:
```
# Step 1: Importing necessary packages at the start of the block
import csv
import json

# Step 2: Defining a function that returns the top movie titles with the␣
 ↪highest average rating for specific countries
def top_movies_by_country(*countries):
    rating_sum = {}  # Dictionary to store the sum of ratings for each movie
    rating_count = {}  # Dictionary to store the count of ratings for each movie
    movie_titles = {}  # Dictionary to store movie titles

# Step 3: Reading Ratings csv File and Adding to rating sum and count for each␣
 ↪movie
    with open('ratings.csv', 'r') as ratings_file:
        ratings = csv.reader(ratings_file)
        next(ratings)  # Skip header as errors were encountered
        for row in ratings:
            movie_id = row[1]
            rating = float(row[2])
            if movie_id in rating_sum:
                rating_sum[movie_id] += rating
                rating_count[movie_id] += 1
            else:
                rating_sum[movie_id] = rating
                rating_count[movie_id] = 1

# Step 4: Reading Movie JSON File to get movie ids, titles and countries
    with open('movies.json', 'r') as movies_file:
        for line in movies_file:
            movie = json.loads(line)
            movie_id = movie['id']
            title = movie['title']
```

```
            country = movie['countries']

            movie_titles[movie_id] = (title, country) # Store movie title and␣
 ↪country list in the dictionary using movie_id as the key

# Step 5: Creating a list of (title, average_rating) tuples for movies matching␣
 ↪the input countries
    movie_averages = []
    for movie_id in rating_count:
#Step 6: Checking if the movie's countries match the input countries
        if set(countries).issubset(movie_titles[movie_id][1]):
            average_rating = rating_sum[movie_id] / rating_count[movie_id]
            movie_averages.append((movie_titles[movie_id], average_rating))

# Step 7: Sorting movies by average rating in descending order and getting the␣
 ↪top 5
    movie_averages = sorted(movie_averages, key=lambda x: x[1], reverse=True)[:
 ↪5]

# Step 8: Printing the results
    print(f"Top 5 Movies for countries {', '.join(countries)}:")
    for title, avg_rating in movie_averages:
         print(f"{title[0]}: {avg_rating:.2f}")

# Step 9: Calling the function for GB & US as per the question:
top_movies_by_country('GB', 'US')
```

```
Top 5 Movies for countries GB, US:
The Winter Guest: 4.50
Merlin: 4.29
Treasure Island: 4.28
Terminator 3: Rise of the Machines: 4.18
The Scapegoat: 4.17
```

1.2.6 Question 6

6. Produce a list of all movie genres available in the dataset, with their overall average rating for each genre. Print also the name of the genre with the highest average rating. Note: ratings pertaining to movies with more than one genre contribute to the average of all the relevant genres. **(15 marks)**

[6]:
```
# Step 1: Importing necessary packages at the start of the block
import json
import csv

# Step 2: Defining a function that returns the average ratings for each movie␣
 ↪genre
def genre_average_ratings():
```

```
    genre_ratings = {}  # Dictionary to accumulate ratings per genre
    genre_counts = {}   # Dictionary to count how many ratings contribute to␣
 ↪each genre

# Step 3: Reading the Ratings csv File & storing ratings by movie ID in a␣
 ↪dictionary
    with open('ratings.csv', 'r') as ratings_file:
        ratings = csv.reader(ratings_file)
        next(ratings)  # Skip header as to prevent errors just in case

        movie_ratings = {} # Store ratings by movie ID
        for row in ratings:
            movie_id = row[1]  # Movie ID
            rating = float(row[2])  # Rating

# Step 4: Checking if the movie has been rated before and appending the new␣
 ↪rating to the list (if yes)
            if movie_id in movie_ratings:
                movie_ratings[movie_id].append(rating)
            else:
                movie_ratings[movie_id] = [rating]

# Step 5: Reading the Movies JSON File and getting all ratings for the movie
    with open('movies.json', 'r') as movies_file:
        for line in movies_file:
            movie = json.loads(line)
            movie_id = movie['id']  # Movie ID
            genres = movie['genres']  # List of genres

            ratings = movie_ratings.get(movie_id) # Get all ratings for the␣
 ↪movie

            if ratings is not None:
                # Distribute the rating across all genres
                for genre in genres:
                    for rating in ratings:
                        if genre in genre_ratings:
                            genre_ratings[genre] += rating
                            genre_counts[genre] += 1
                        else:
                            genre_ratings[genre] = rating
                            genre_counts[genre] = 1

# Step 6: Calculating the average ratings for each genre
    genre_average = {}
    for genre in genre_ratings:
        genre_average[genre] = genre_ratings[genre] / genre_counts[genre]
```

```
# Step 7: Printing the average ratings for each genre
    print("Average Ratings by Genre:")
    for genre, average in genre_average.items():
        print(f"{genre}: {average:.2f}")

# Step 8: Identifying and printing the genre with the highest average rating
    highest_avg_genre = max(genre_average, key=genre_average.get)
    highest_avg_rating = genre_average[highest_avg_genre]

    print(f"\nGenre with the highest average rating: {highest_avg_genre}␣
 ↪({highest_avg_rating:.2f})")

# Step 9: Calling the function to calculate and print genre average ratings
genre_average_ratings()
```

```
Average Ratings by Genre:
Animation: 3.59
Comedy: 3.53
Family: 3.39
Adventure: 3.50
Fantasy: 3.50
Action: 3.56
Crime: 3.57
Drama: 3.53
Thriller: 3.56
Romance: 3.54
Science Fiction: 3.53
Mystery: 3.65
Music: 3.44
Horror: 3.49
History: 3.42
War: 3.50
Documentary: 3.49
Western: 3.65
Foreign: 3.49
TV Movie: 3.62

Genre with the highest average rating: Western (3.65)
```

1.2.7 Question 7

7. We want to implement a small recommender system which, given a movie as input, recommends the most similar movie. The idea behind it is to recommend the movie with the most similar rating pattern to the movie provided as input (e.g. if our input movie has been liked by some users and disliked by others, we will try to recommend one where similar users liked and disliked it). To do this, we will measure the pairwise cosine similarities between the input movie and each of the other movies in the dataset, to find the one that maximises the

similarity.

The cosine similarity between two vectors (in python, lists) A and B is measured as:

$$\mathbf{A} \cdot \mathbf{B} / \|\mathbf{A}\| \|\mathbf{B}\| = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}}$$

That's the (element-wise) multiplication of both vectors, divided by the multiplication of their norms. As an example, if we have two vectors: * A = [0, 1, 2] * B = [1, 2, 3]

The cosine similarity between A and B is:

$sim(A, B) = (0 * 1 + 1 * 2 + 2 * 3)/(\sqrt{0^2 + 1^2 + 2^2} * \sqrt{1^2 + 2^2 + 3^2}) = 0.956182887$

See the toy example below where we have 3 movies and 5 users. Cells with a 0 indicate that the user hasn't rated that movie, whereas values 1-5 indicate that the user has rated the movie with that value:

	user 1	user 2	user 3	user 4	user 5
movie 1	5	0	2	1	5
movie 2	1	3	0	1	4
movie 3	4	0	2	1	0

Let's say in this case our input movie is movie 1, so we want to find the movie that's most similar to movie 1. We would compute the pairwise cosine similarities between movie 1 (our input movie) and every other movie in the dataset: * Cosine similarity between movie 1 and movie 2 is: 0.674699 * Cosine similarity between movie 1 and movie 3 is: 0.735612

Hence, we would recommend movie 3 as the one with the highest cosine similarity with movie 1.

To complete this question, write a python function which, given a movie as input, outputs the most similar movie as the recommended item based on highest cosine similarity with the input movie.

(30 marks)

NB that you can only use the *json* and *csv* packages. To calculate the square root of a value, you can use (1/2) as the exponent of a base number, e.g. 3**(1/2) calculates the square root of 3.

[7]:
```
# Step 1: Importing necessary packages at the start of the block
import json
import csv

# Step 2: Defining a function that returns a movie recommendation based on user␣
 ↪ratings and cosine similarity
def recommendation (input_movie):

# Step 3: Reading the Movies JSON File for movie titles and IDs
    movie_id_to_title = {}
    with open('movies.json', 'r') as movies_file:
        for line in movies_file:
            movie = json.loads(line)
```

```
            movie_id_to_title[movie['id']] = movie['title']

# Step 4: Reading the Ratings csv File for user ratings
    user_id_movie_rating = {}
    with open('ratings.csv', 'r') as ratings_file:
        ratings = csv.reader(ratings_file)
        next(ratings)  # Skip header to prevent errors
        for row in ratings:
            user_id = row[0]
            movie_id = row[1]
            rating = float(row[2])
            movie_id_movie_rating = {}
            movie_id_movie_rating [movie_id]= rating

            if user_id not in user_id_movie_rating:
                user_id_movie_rating [user_id] = movie_id_movie_rating
            else:
                user_id_movie_rating [user_id] [movie_id] = rating

# Step 5: Converting the input movie title to its corresponding movie ID
    input_movie = str([movie_id for movie_id, movie_title in movie_id_to_title.
 ↪items() if movie_title == input_movie][0])

# Step 6: Identifying users who have rated the input movie (to reduce the␣
 ↪dataset to be analyzed)
    user_input_movie = [] # users that rated 'input_movie'
    for user in user_id_movie_rating:
        if input_movie in user_id_movie_rating [user]:
            user_input_movie.append(user)

# Step 7: Creating a set of common movies (movies rated by users who rated the␣
 ↪input movie)
    common_movies = set()
    for user in user_input_movie:
        common_movies.update(user_id_movie_rating [user].keys())

# Step 8: Defining a helper function to get ratings for common movies
    def movie_ratings (movie_id):
        ratings =[]
        for user in user_input_movie:
            rating = user_id_movie_rating[user].get(movie_id,0) # gives rating␣
 ↪or gives zero 'in case the movie was not found'
            ratings.append(rating)
        return ratings

#Step 9: Creating a dictionary for common movies and their corresponding ratings
    common_movie_rating = {}
```

```
    for movie_id in common_movies:
        common_movie_rating [movie_id] = movie_ratings (movie_id)

# Step 10: Defining cosine similarity function to compare movie ratings (for␣
 ↪common movies)
    def cosine_similarity(A, B):
            dot_product = sum(a * b for a, b in zip(A, B))
            norm_A = sum(a ** 2 for a in A) ** (1 / 2)
            norm_B = sum(b ** 2 for b in B) ** (1 / 2)

            if norm_A == 0 or norm_B == 0:
                return 0  # Avoid division by zero

            return dot_product / (norm_A * norm_B)

# Step 11: Creating a dictionary to store cosine similarity between input movie␣
 ↪and other movies
    cosine_similarity_movie = {}
    for movie_id in common_movies:
        if movie_id != input_movie:
            A = common_movie_rating [input_movie]
            B = common_movie_rating [movie_id]
            cosine_similarity_movie [movie_id] = cosine_similarity (A,B)

# Step 12: Getting the movie with the maximum consine similarity value
    max_similarity_movie = max (cosine_similarity_movie.items (), key = lambda␣
 ↪item: item [1])

    max_similarity_movie_name = movie_id_to_title [max_similarity_movie[0]]

    max_similarity_movie_score = max_similarity_movie [1]

# Step 13: Printing the recommended movie
    print ('Recommended movie is', max_similarity_movie_name, '\nWith␣
 ↪similarity index of', max_similarity_movie_score)
    return

# Step 14: Calling the function for Toy Story as an example
recommendation ('Toy Story')
```

```
Recommended movie is Men in Black II
With similarity index of 0.8824425280748285
```

[ ]: