Homework Assignment 1

Movie Recommendation System

Worth 120 points

Posted Friday, Sep 24

Due Monday, Oct 11 at 11 PM in Canvas

Late Submission: By Tuesday, Oct 12, 11 PM in Canvas - 15 point penalty

You are required to design a prototype movie recommendation program in Python according to the following guidelines.

You have the option to work individually or with a partner. But no groups over 2 people! If you choose to work with a partner, you don't have to tell us upfront who you will be working with. We will post an announcement on Canvas in a few days that will detail how to inform us about your partnership.

Make sure you abide by the **DCS Acadmic Integrity Policy for Programming Assignments.**

Write all the required functions described below in a file named **hw1.py** You may also implement other helper functions as needed.

Submit ONLY your **hw1.py** file to Canvas.
If you are working with a partner, submit only ONE copy (either of you can submit, you will both get the same score.)

Do not submit any other files, do not submit a zip file. We don't need you to submit any files you may have used to test your program. (Make sure to test your programs on files other than the samples we have provided, to cover the various paths of logic in your code.)

You are allowed up to 3 submissions (total over regular and late submissions), only the last submission will be graded. Note: If you have a late submission, it will override any submissions you may have made before the regular deadline, and you will be assessed the 15 point penalty.

You can test your program by adding calls to various functions in the **hw1.py** file. You may retain this test code when submitting, but we will ignore all of it. When we test your submission, we will call the required functions individually on our own test data, and points will be assigned to each function based on the correctness of the returned value.

There is no partial credit for code structure, etc. Credit is given only when correct values are returned from your functions. However, each function will be tested on several cases. So for instance, if a function runs correctly on 2 out of 3 test cases, you will get full points for the 2 cases and zero for the third. (In this sense, there is partial credit for each function.)

Data Input

- **Ratings file**: A text file that contains movie ratings. Each line has the name (with year) of a movie, its rating (range 0-5 inclusive), and the id of the user who rated the movie. A movie can have multiple ratings from different users. A user can rate a particular movie only once. A user can however rate multiple movies. Here's a **sample ratings file**

- **Movies file**: A text file that contains the genres of movies. Each line has a genre, a movie id, and the name (with year) of the movie. To keep it simple, each movie belongs to a single genre. Here's a **sample movies file**

You may assume that input files will be correctly formatted, and data types will be as expected. So you don't need to write code to catch any formatting or data typing errors.

Task 1: Reading Data

1. [10 pts] Write a function read_ratings_data(f) that takes in a ratings file, and returns a dictionary. The dictionary should have movie as key, and the corresponding list of ratings as value.

   For example: `movie_ratings_dict = { "The Lion King (2019)" : [6.0, 7.5, 5.1], "Titanic (1997)": [7] }`

2. [10 pts] Write a function read_movie_genre(f) that takes in a movies file and returns a dictionary. The dictionary should have a one-to-one mapping between movie and genre.

   For example `{ "Toy Story (1995)" : "Adventure", "Golden Eye (1995)" : "Action" }`

Watch out for unwanted leading and trailing whitespaces in movie and genre, remove them when encountered.

Task 2: Processing Data

1. [8 pts] Genre dictionary

   Write a function create_genre_dict that takes as a parameter a movie-to-genre dictionary, of the kind created in Task 1.2. The function should return another dictionary in which a genre is mapped to all the movies in that genre.

   For example: `{ genre1: [ m1, m2, m3], genre2: [m6, m7] }`

2. [8 pts] Average Rating

   Write a function calculate_average_rating that takes as a parameter a ratings dictionary, of the kind created in Task 1.1. It should return a dictionary where the movie is mapped to its average rating computed from the ratings list.

   For example: `{"Spider-Man (2002)": [3,2,4,5]} ==> {"Spider-Man (2002)": 3.5}`

Task 3: Recommendation

1. [10 pts] Popularity based

   In services such as Netflix and Spotify, you often see recommendations with the heading “Popular movies” or “Trending top 10”.

   Write a function get_popular_movies that takes as parameters a dictionary of movie-to-average rating (as created in Task 2.2), and an integer n (default should be 10). The function should return a dictionary (movie:average rating, same structure as input dictionary) of top n movies based on the average ratings. (If there are fewer movies than n, it should all return all movies in order of top average ratings.)

2. [8 pts] Threshold Rating

   Write a function filter_movies that takes as parameters a dictionary of movie-to-average rating (same as for the popularity based function above), and a threshold rating with default value of 3. The function should filter movies based on the threshold rating, and return a dictionary with same structure as the input. For example, if the threshold rating is 3.5, the returned dictionary should have only those movies from the input whose average rating is equal to or greater than 3.5.

3. [12 pts] Popularity + Genre based

   In most recommendation systems, genre of the movie/song/book plays an important role. Often features like popularity, genre, artist are combined to present recommendations to a user.

   Write a function get_popular_in_genre that, given a genre, a genre-to-movies dictionary (as created in Task 2.1), a dictionary of movie:average rating (as created in Task 2.2), and an integer n (default 5), returns the top n most popular movies in that genre based on the average ratings. The return value should be a dictionary of movie-to-average rating of movies that make the cut. Genre categories will be from those in the movie:genre dictionary created in Task 1.2. Your code should handle the case when there are fewer than n movies in the data, as in Task 3.1 above.

4. [8 pts] Genre Rating

   One important analysis for the content platforms is to determine ratings by genre.

   Write a function get_genre_rating that takes the same parameters as get_popular_in_genre above, except for n, and returns the average rating of the movies in the given genre.

5. [12 pts] Genre Popularity

   Write a function genre_popularity that takes as parameters a genre-to-movies dictionary (as created in Task 2.1), a movie-to-average rating dictionary (as created in Task 2.2), and n (default 5), and returns the top-n rated genres as a dictionary of genre:average rating. Hint: Use the above get_genre_rating function as a helper.

Task 4 (User Focused)

1. [10 pts] Read the ratings file to return a user-to-movies dictionary that maps user ID to the associated movies and the corresponding ratings. Write a function named read_user_ratings for this, with the ratings file as the parameter.

   For example: `{ u1: [ (m1, r1), (m2, r2) ], u2: [ (m3, r3), (m8, r8) ] }`

   where `ui` is user ID, `mi` is movie, `ri` is corresponding rating.

2. [12 pts] Write a function get_user_genre that takes as parameters a user id, the user-to-movies dictionary (as created in Task 4.1 above), and the movie-to-genre dictionary (as created in Task 1.2), and returns the top genre that the user likes based on the user's ratings.

   Here, the top genre for the user will be determined by taking the average rating of the movies genre-wise that the user has rated.

3. [12 pts] Recommend 3 most popular (highest average rating) movies from the user's top genre that the user has not yet rated. Write a function recommend_movies for this, that takes a parameters a user id, the user-to-movies dictionary (as created in Task 4.1 above), the movie-to-genre dictionary (as created in Task 1.2), and the movie-to-average rating dictionary (as created in Task 2.2). The

function should return a dictionary of movie-to-average rating. (Return all if fewer than 3 movies make the cut.)