

Homework Assignment 1

Movie Recommendation System

Worth 120 points.

Posted Friday, Feb 9

Due ~~Wednesday Feb 28~~ Thursday, Feb 29 at 11 PM in Canvas

You are required to design a prototype movie recommendation program in Python as detailed in the assignment description.

You will work individually.

We will NOT be using Autolab in this class for submitting assignments. This means your program will only be graded once, after the submission deadline has passed.

Make sure you abide by the [DCS Academic Integrity Policy for Programming Assignments](#).

What to submit

Write all the required functions described below in the given [template file named hw1.py](#).

Fill in the function code for all the functions in this template. Note: Every function has a `pass` statement because Python does not allow empty functions. The `pass` statement does nothing, it's simply a filler. You can delete it when you write in your code.

You may implement other helper functions as needed.

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.

Make sure you write ALL your test calls in the `main()` function. **Do NOT write ANY code outside of any of the functions.**

When you are done writing and testing, **submit ONLY the filled-in hw1.py file to Canvas.**

Do NOT submit a Jupyter notebook, it will not be accepted for grading.

You are allowed up to 5 submissions, **only the last submission will be graded.**

How to test your code

You can test your program by calling your functions in the **hw1.py** file from the `main()` function. **All test code must be in the `main()` function ONLY. If you write ANY code outside of any of the functions, you will lose credit.**

In Terminal, execute your program like this:

```
> python hw1.py
```

This was explained in class: see `jan22_notes.ipynb/jan22_notes.html` between cells #23 and #24 - Writing and executing standalone (outside Jupyter notebook) Python programs.

Make sure the test files are in the same folder as the program. You may develop and test your code in a Jupyter notebook, but for submission you will need to move your code over to `hw1.py` and execute it as above to make sure it works correctly. The process of moving code from a Jupyter notebook to a Python file has been explained in class.

You can run your tests on the given ratings and movies files, but testing on only these files may not be sufficient. You should make your own test files as well, to make sure that you cover the various paths of logic in your functions. You are not required to submit any of your test files.

You may assume that all parameter values to your functions will be legitimate, so you are not required to check whether the parameter values are valid.

In any function that requires the returned values to be sorted or ranked, ties may be broken arbitrarily between equal values.

You may retain the `main()` function when submitting, but we will IGNORE it.

For this assignment only, grading will be done by an AUTOGRADER program that will call the functions in your `hw1.py`, and check the returned value against the expected correct value. It will NOT call your `main()` function.

The AUTOGRADER does not look at printed output, so anything you print in your program will be ignored. There will not be any manual inspection of code, credit is based solely on whether your functions return correct results.

Partial Credit

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](#).

Note: A movie name includes the year, since it's possible different movies have the same title, but were made in different years. However, no two movies will have the same name in the same year.

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.

For all computation of rating, do not round up (or otherwise modify) the rating unless otherwise specified.

Implementation

Task 1: Reading Data

1. [10 pts] Write a function `read_ratings_data(f)` that takes in a ratings **file name**, and returns a dictionary. (Note: the parameter is a file name string such as "myratings.txt", NOT a file pointer.) The dictionary should have movie as key, and the list of all ratings for it 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 name** and returns a dictionary. The dictionary should have a one-to-one mapping from movie to genre.

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

Watch out for leading and trailing whitespaces in movie name and genre name, and remove them before storing in the dictionary.

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 than n movies, it should return all movies in ranked order of average ratings from highest to lowest.

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. If there are fewer than `n` movies, it should return all movies in ranked order of average ratings from highest to lowest.

Genres will be from those in the movie:genre dictionary created in Task 1.2. The genre name will exactly match one of the genres in the dictionary, so you do not need to do any upper or lower case conversion.

4. [8 pts] Genre Rating

One important analysis for 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. If there are fewer than `n` genres, it should return all genres in ranked order of average ratings from highest to lowest. 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 a list of the movies they rated, along with the rating they gave. 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. You can handle user ID as int or string type, but make sure you consistently use it as the same type everywhere in your code.

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. If multiple genres have the same highest ratings for the user, return any one of genres (arbitrarily) as the top genre.
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 as 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. If fewer than 3 movies make the cut, then return all the movies that make the cut in ranked order of average ratings from highest to lowest.