CS2204 Homework: Movie Ratings

Objectives

- Learn to use Python dictionaries (a lot)
- Learn to use Python sets (if you want to)
- Practice sorting with dictionaries and sets
- Understand a simple similarity metric
- Implement a collaborative filtering algorithm

Background

In this homework assignment, you are going to implement a movie recommendation system. First, you will find movies and genres which are the most popular in the overall database. Second, you will implement a *collaborative filtering* approach, with which you can provide a more personalized recommendation based on the *taste profile* of a user.

You are given a real[1] movie and rating database. The data loading code is provided, so you have access to this dataset through two variables/objects: `movies` and `users`. The functions that you will need to implement will receive these two objects as parameters. It is **extremely** important to familiarize yourself with these two data structures before you start working on the solution(s).

The `movies` object is a dictionary, where the *key* is the string identifier of a movie (e.g.: `"tt0112817"`). Indeed, these identifiers are used by the IMDB website, so you can check out these movies by appending the identifier to the `https://www.imdb.com/title/` URL (e.g., https://www.imdb.com/title/tt0112817 (https://www.imdb.com/title/tt0112817)). The *value* is another dictionary for each element, which describes the movie with a `title` (string) and `genres` (list of strings) data elements. Let's see a fragment of the `movies` dictionary:

```
movies = {
    'tt0058331': {
        'title': 'Mary Poppins',
        'genres': ['Comedy', 'Family', 'Fantasy']
    },
    'tt0033563': {
        'title': 'Dumbo',
        'genres': ['Animation', 'Family']
    },
    ...
```

The `users` object is also a dictionary, where the *key* is a string identifier of the user (e.g.: `"u92517"`). You probably do not need to process/use these identifiers to solve the tasks. The *value* portion is another dictionary that contains one or more items, mapping a movie identifier (see above) to a rating value (float). These are the movie ratings of that particular user. Let's see a fragment of the `users` dictionary:

```
users = {
    'u92517': {
        'tt0133093': 2.0,
        'tt0120737': 5.0,
        'tt0167260': 5.0
    },
    'u43453': {
        'tt0108052': 3.0,
        'tt0116329': 4.0,
        'tt0083866': 4.0,
        'tt0081375': 3.0,
        'tt0075686': 4.0,
        'tt0086425': 3.0
    },
    ...
```

As you can see, the *connections* between these two data structures are created by the movie ids.

Tasks

You need to finish the `movies.py` source file. Note: you can use the simple tests provided at the end of the file (by uncommenting those lines). Feel free to change/add/delete any testing code. The outputs of this testing code are not going to be graded. The validator program will use your code directly.

1. Implement the `unrated_movies` function. You are given the dataset (`movies` and `users` as described above). Return a list of all **movie titles** which are not rated by any of the users. *Hint:* try to use Python sets and their operators to solve this task. Also, the most effective way to create a set containing all the keys of an existing dictionary is `key_set = set(my_dict.keys())` (10 pts)
2. Implement the `most_rated_movies` function. You are given the dataset (`movies` and `users` as described above) along with an `n_top` integer parameter. Return a list of the `n_top` most rated **movie titles**. Note: the rating values are not important, only how many ratings are provided by all the users for each movie. *Hint:* you may want to experiment with the `collections.Counter` data type to solve this problem. (15 pts)
3. Implement the `highest_rated_movies` function. You are given the dataset (`movies` and `users` as described above) along with the `n_top` and `n_min_ratings` integer parameters. Return a list of the `n_top` **movie titles** based on their **average** rating value. Note: you should consider only those movies, which have at least `n_min_ratings` available. Once you solved the problem, it is interesting to experiment with how this second parameter can dramatically change the rankings. *Hint:* you may find it easier to collect all the ratings for each movie in a new dictionary (move id -> list of rankings) and then implement the filtering, averaging, and sorting sub-tasks on this temporary dictionary. (20 pts)
4. Implement the `most_popular_genres` function. You are given the dataset (`movies` and `users` as described above) along with the `n_top` integer parameter. You need to return a list of `n_top` strings, which are the most popular genres based on the average ranking of the movies. It is similar to the previous assignment, but you need to collect the rankings for the genres (e.g.. if a movie receives a ranking of 4.0 from a user and belongs to two genres, then both genres will receive this 4.0 ranking). (20 pts)

5. Implement the `taste_similarity` function. For that you do not need to have access to the dataset. The taste similarity metric is based on the well-known cosine similarity (https://en.wikipedia.org/wiki/Cosine_similarity) metric. You should return a float value (between `0.0 - 1.0`) which is calculated on the ratings information of two users as follows:

$$\frac{user1_{r1} \cdot user2_{r1} + user1_{r2} \cdot user2_{r2} + \dots}{\sqrt{user1_{r1}^2 + user1_{r2}^2 + \dots} \cdot \sqrt{user2_{r1}^2 + user2_{r2}^2 + \dots}}$$

where $user1_{r1}$ and $user2_{r1}$ are the rating values of `user1` and `user2` for the **same** movie. Similarily, $user1_{r2}$ and $user2_{r2}$ is another ratings pair for another movie. **If a movie is not rated by both users, use 0 for the missing rating.** (15 pts)

6. Implement the `suggest_movie` function. You are given the dataset (`movies` and `users` as described above) and the rating dictionary of a new/unknown user. First, find another user in the `users` dataset, which has the highest *taste similarity* to this new/unknown user, then select the movie which is rated the highest by the found user, but is not among the new/unknown user's rated movies. If all the movies of the found user are rated by new/unknown user, return `None`. This algorithm is the core idea of most movie/book/product recommendation systems (20 pts).

Grading

You can use the attached `validator.py` program to check your work (and the instructor's original mistakes). It will also estimate your final score for the homework. The program is in the same folder as your homework assignment. Open the `validator.py` script and run it in the Spyder environment to track your progress.

Penalties

Points will be deducted if you fail to set `__author__` variable (-10 pts) and for **each PEP 8 style errors** (-1 pt for each) in your program.

Submission

Please, upload the final version of the following file(s) (**and only those files**) to Brightspace:

- `movies.py`

Footnotes

1: Kaggle - The Movies Dataset, https://www.kaggle.com/rounakbanik/the-movies-dataset (https://www.kaggle.com/rounakbanik/the-movies-dataset). *Creative Commons CC0 1.0,* Retrieved on 9/23/2020.