Assignment 3: User-based Collaborative Filtering

(Adapted from University of Minnesota CSci 1901H Class project)

Assignment Overview

In this assignment you will implement a simple user-based collaborative filtering recommender system. The ratings data you will use for developing and testing your program was generated by students.

Write your own code!

For this assignment to be an effective learning experience, you must write your own code! I emphasize this point because you will be able to find Python implementations of most or perhaps even all of the required functions on the web. Please do not look for or at any such code! **Do no share code with other students in the class!!**

Here's why:

- The most obvious reason is that it will be a huge temptation to cheat: if you include code written by anyone else in your solution to the assignment, you will be cheating. As mentioned in the syllabus, this is a very serious offense, and may lead to you failing the class.
- However, even if you do not directly include any code you look at in your solution, it surely will influence your coding. Put another way, it will short-circuit the process of you figuring out how to solve the problem, and will thus decrease how much you learn.

So, just don't look on the web for any code relevant to these problems. Don't do it.

Format of ratings file

The file consists of one rating event per line. Each rating event is of the form:

```
user_id\trating\tmovie_title
```

The `user_id` is a string that contains only alphanumeric characters and hyphens (no whitespace, no tabs). The `rating` is one of the float values 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, and 5.0 The movie_title is a string that may contain space characters (to separate the words). The three fields -- `user_id`, `rating`, and the `movie_title` -- are separated by a single tab character (`\t`).

Submission Details

You have been given a skeleton python program collabFilter.py. You will turn in your version of collabFilter.py with the four required functions implemented.

What you will turn in: `lastname_firstname_collabFilter.py`

Part 1

Implement the function readRatings(ratings_file)

The function reads in the provided ratings file in the format described above. The Input is the location of the ratings file. The Output is a dictionary whose keys are user_ids, and whose values are dictionaries whose keys are movie titles and whose values are the user's rating for those movies.

For example:

```
{
  'Terveen':{
    'Pulp Fiction': 5,
    'Armageddon':2.0,
    'Star Wars: Episode I - The Phantom Menace':1.0,
    'Once Upon a Time in Mexico':3.0
  },
  'Kluver':{
    'Harold and Kumar Go To White Castle':5.0,
    'Bridges of Madison County':5.0,
    'Lord of the Rings: Return of The King':2.0,
    'BattleField Earth':4.0
  }
}
```

Hint: Using the split() function will make it very easy to separate the `user_id`, `rating`, and `movie_title` fields.

Hint: Learn about dictionaries of dictionaries. You are creating a dictionary whose keys are user_ids and whose values are dictionaries (whose keys are movie_titles and whose values are ratings).

Part 2

Implement the function similarity(user_ratings_1, user_ratings_2)

user_ratigs_1 and user_ratings_2 are dictionaries that contain the ratings for two users in the format produced by readRatings(). Returns a float value between 1 (indicating total agreement) and -1 (indicating total disagreement).

You will compute similarity using the Pearson correlation coefficient. This is described in the readings provided in class and in the Wikipedia article on collaborative filtering.

IMPORTANT CLARIFICATION: When you compute the term for the average of each user's ratings, you should include **all** the user's ratings, not just ratings of items in the intersection of the two user's ratings.

However, **when computing the summations in the Pearson correlation equation**, you should only use the items that both users have rated when determining the similarity of the users. If the users have not rated any movies in common, you should return a similarity value of 0.0.

As always, you may define any helper functions you find useful, for this and any of the levels of the assignment.

Part 3

Implement the function nearestNeighbors(user_id, all_user_ratings, k). This function identifies the k-nearest-neighbors to a user specified by user_id using your similarity function.

The method returns a **list** of the k "nearest neighbors" of the user `user_id`. Each element in the list is a tuple consisting of two elements: (uid_j, sim_j). The uid_j is the ID for one of the k nearest neighbors of user_id; the sim_j is the similarity of uid_jto user_id.

`k` is an integer.

The `k` nearest neighbors are the `k` users with the highest similarity scores to user `user_id`.

Sample output from nearestNeighbors would look like:

```
[("kluver", 0.91), ("lange", 0.78), ("tveite", 0.74), ("terveen", 0.37)]
```

To compute the k nearest neighbors, you must first compute the similarity of `user_id` to all the other users (be sure not to compare user_id to him/herself!). There are several ways to proceed next, but an effective way is simply to build a list of (`uid_j`, `sim_j`) pairs as you compute the similarity scores, then sort this list by the similarity scores, and finally return a list consisting of the first k items.

Hint: To sort a list of tuples by the second value of the tuples, you should look at the **sorted** and **lambda** functions in python. Also look at how to sort in reverse or descending order.

Part 4

Implement a function to predict a rating for a user U and item I. This function will use the nearestNeighbor algorithm. You will compute a simple weighted average of the ratings provided by the k nearest neighbors.

predict(item, k_nearest_neighbors, all_user_ratings)
where: item is the item for which you want to make a prediction (for a given user). k_nearest_neighbors is a list of tuples that is the output of nearestNeighbors. all_user_ratings is the output of readRatings.

The output is a float between 0.5 and 5.0.

IMPORTANT CLARIFICATION: The neighbors of some users for some k will not have ratngs for every item. If the passed neighbors don't have ratings for the passed item return a prediction of 0.

Here is the idea for your solution:

- Remove any of the k-nearest-neighbors who don't have a rating for item.
- Compute a weight for each of the neighbors that determines how much each neighbor's opinion of item should influence the prediction for user U. Do this as follows:
    - Sum the similarities (to user U) of each of the k neighbors. Call this S. S = sum(sim[i])
- The prediction of how much user U will like item now is simply:

  sum(sim[i] * rating[i]) / S, where rating[i] is user i's rating of item

Running your code

The program takes 5 arguments: the name of the ratings file, the id of user1 (a string), the id of user2 (a string), a movie name that you will predict the rating for (a string), and an integer k to indicate the number of k_nearest_neighbors.

```
python collabFilter.py ratingsFileName user1 user2 movieName k
```

The program will output:

a) all user ratings as a dictionary of dictionaries
b) the similarity value for user1 and user2
c) the k_nearest_neighbor list for user1, which consists of tuples with (uid_j, sim_j) values
d) the predicted rating for the movie name

Sample program output is provided for the following arguments:

```
python collabFilter.py ratings-dataset.tsv Kluver terveen 'The Fugitive' 10
```

The nearest neighbor data structure gives you similarity values for the 10 nearest neighbors, which will allow you to verify that your similarity calculation is correct.