Homework 2 - A Collaborative Filtering Recommender Systems

Due Date: Tuesday, October 20 by 3:30 pm

Assignment Overview

In this assignment you will implement a simple collaborative filtering recommender system. You will implement one major function for each of the C-level, B-level, and A-level requirements, and another major function for extra credit.
The ratings data you will use for developing and testing your program on will be generated by you, the students. Everyone in the class will rate a set of movies, and the resulting ratings will be stored in a file that you will use as input to your program.

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!
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`\t`rating`\t`movie_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).

Homework requirements

You are required to turn in two files recommend.py, containing the required functions, and README.txt, which must specify which level you are attempting/claiming to have completed. As always, if you do the homework with a partner, both of you must include your names in the README.txt file.
To help you gauge your progress and test your code, we will develop a test suite you can use to test your code. Don't treat the test suite as complete: we will test and inspect your code using more tests than those we have provided.

Grade level descriptions

C-level

You will write two functions:

- readRatings(ratings_file)
    - Reads in a ratings file in the format described above. The input is the location of the ratings file (i.e., readRatings("ratings.tsv")). 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
        }
      }
      ```
    - Using the split() function will make it very easy to separate the user_id, rating, and movie_title fields
- 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().
      ```

    - This function 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 Wikipedia article on collaborative filtering: since this is stated in terms of user ratings, you can follow it pretty exactly in your implementation.

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.

Because some people in the class have seen different movies you should perform the computation only over the movies that both users have rated. If the users have not rated any movies in common you should return a similarity of 0.

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

B-Level

You will implement the k-nearest-neighbors algorithm using your similarity function.

- nearestNeighbors(user_id, all_user_ratings, k)
    - This function returns a list of the k "nearest neighbors" of the user user_id. k is an integer (don't worry about error checking -- you can assume it really is an integer). The k nearest neighbors are just the k users with the highest similarity scores to user user_id.
    - Each item in the returned list is a tuple consisting of two elements: the first element is a user_id -- call it uid_j -- and the second element is the similarity score of user uid_j to user user_id -- call this sim_j

  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.

A-Level

Your task at the A-level is to predict a rating for a user U and item I. We will assume you've already written and have access to the nearestNeighbor algorithm. Write the following function:

- predict(item, k_nearest_neighbors, all_user_ratings)
    - 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 should be a float between 0.5 and 5.0.

IMPORTANT CLARIFICATION: The neighbors of some users for some k will not have ratings 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.

Extra Credit

Your task is to build on what you've done at the previous level to write a function to recommend items to a target user U.

- recommend(user_id, all_user_ratings)
- Return a list of pairs (tuples) whose first elements are movies and whose second elements are the predicted ratings for the movie by user user_id, based on the data in all_user_ratings(which as usual is the output of readRatings)

Here's the idea:

- Find the k-nearest neighbors (**k=10**)
- Find items that **any** of the neighbors have rated highly that user user_id has not rated. Define "rated highly" as ratings of 4.0, 4.5, and 5.0.
- This will give you a list of items (remember to remove duplicates).
- Now, compute a prediction for each of the items in the list (see A-level), and build the list of of pairs: [(movie-1, prediction-1), ..., (movie-n, prediction-n)]
- Sort the list by prediction scores: you've got the recommendations!