CSCI 1300 Introduction to Computer Programming

Instructor: Knox

Assignment 10

Due Saturday, 6th May 2017

**Recommendation System**



Assignment 10 is putting all your knowledge together into solving a real world problem. We will provide you with a test data set with which you can test your code, but your grade for your assignment will be based on processing a different data set. You will not know the exact data being used, but will be told the purpose of each test case and the outcome.

You will be implementing a recommendation system similar to the ones you see on Netflix, Amazon, or Barnes&Noble. Your recommender class will read a set of reader ratings for a set of books and provide functions/methods to analyze the data and suggest new books for a given reader.

For this assignment, you need to submit one file called Assignment10\_LastName.py,

where LastName is your last name. Please include comments in your code that explain what your code is doing. The comments should also include your name, recitation TA, and the assignment number. Each function should be commented with the functions purpose and description of the parameters.

**Submitting Your Code to the Autograder:**

Before you submit your code to COG, make sure it runs on your computer. If it doesn’t run on the VM, it won’t run on COG.  The computer science autograder, known as COG, can be found here: https://web-cog-csci1300.cs.colorado.edu

* Login to COG using your identikey and password.
* Select the CSCI1300 - Assignment # from the dropdown.
* Upload your .zip file and click Submit.

**Submitting Your Code to Moodle:**

You must also submit your code to Moodle to get full credit for the assignment, even if the computer science auto-grader gives you a perfect score.

**Files**

There are two files that are used in this assignment: a list of books (books.txt) and a list of ratings from readers (ratings.txt). There are ratings associated with each book in the books file for each reader. The order of the ratings by an individual reader is the same as the order of the books in the books.txt data file.

The ratings given by a reader are shown in the table below. Be careful to note that if a reader has read the book, then the rating is a non-zero.

**Rating System:**

|  |  |
| --- | --- |
| Rating | Meaning |
| 0 | Haven’t read it |
| -5 | Hated it |
| -3 | Didn’t like it |
| 1 | Ok. neither hot nor cold about it |
| 3 | Liked it! |
| 5 | Really liked it! |

The data file containing the books contains a single line per book. Each entry has the author’s name and title of the book separated by commas.

**books.txt**



The data file containing the reader’s ratings contains one line per entry. Each entry has the reader’s user\_id followed by a list of ratings separated by spaces on the second line. The user\_id cannot contain spaces. The ratings are listed in the order of the books supplied in the books data file.

**ratings.txt**



The way to map the ratings to the correct book is that the first rating for a given user is for the first book listed in the book file, and the second rating for a user is for the second book listed in the book file, etc. For example, Reuven gave a 5 to The Hitchhiker’s Guide, a ‐5 for Watership Down, and a 0 (have not read the book) for The Five People You Meet in Heaven.

**Part 1: Create functions that parse the data files and create dictionaries**

1. Write a function ***read\_books(file\_name)*** which takes in the name of a file as parameter and returns a dictionary. The key should be in the index of the book within the file. The value for each dictionary entry is a list containing the author and the title. For instance, after reading and adding the first entry into the dictionary, it would look like this:

{0: [The Hitchhiker's Guide To The Galaxy , Douglas Adams] }

The function should return ***None*** if the file could not be read, otherwise it should return a dictionary filled with data from that file.

1. Write a function ***read\_users(user\_file)*** which takes in the name of a file as parameter and returns a dictionary. The dictionary created should use the user\_id as the key and the list of ratings as value. For instance, the first element added to your dictionary would look like the following:   
   {Ben : [5 0 0 0 0 0 0 1 0 1 ­3 5 0 0 0 5 5 0 0 0 0 5 0 0 0 0 0 0 0 0 1 3 0 1 0 ­5 0 0 5 5 0 5 5 5 0 5 5 0 0 0 5 5 5 5 ­5]}

The function should return None if the file cannot be read, otherwise it should return a dictionary filled with the data from the given file.

1. Write a function ***calculate\_average\_rating()*** which calculates the average rating of every book and stores it in a dictionary. The dictionary created should have the book index and it’s average rating as the key: value pair respectively. For instance, the first element of your dictionary should look like the following:

{ 0 : 3.83, …}

1. Write a function ***lookup\_average\_rating(index)*** which takes in the book index as parameter and returns a string that is a formatted combination of the book’s average rating, the title of the book, and the author.

(3.83) The Hitchhiker's Guide To The Galaxy by Douglas

**Part 2: Create a class *Recommender***

The functions created in part 1 are incorporated as methods into the new class ***Recommender***. You will provide a constructor that takes two filenames (books\_filename, ratings\_filename) as parameters and creates the required dictionaries as class data members. You will also provide other methods as described below.

1. Write a method ***calc\_similarity(user1, user2)*** to find the similarity between two user ratings. This method is similar to the method used in DNA assignment as it compares the individual ratings from each user to determine the similarity. This method will be used to find the most similar user to the user requiring recommendations.

The similarity between two users is calculated by treating each of their ratings as a mathematical vector and calculating the dot product of these two vectors. (Remember that the dot product is just the sum of the products of each of the corresponding elements.)

For example, suppose we had 3 books and ratings as follows:

Terry 5 3 1

Bob 5 1 5

Tracey 1 5 3

Kalid 1 3 0

The calculation for similarity between

* Terry and Bob: (5 x 5) + (3 x 1) + (1 x 5) = 25 + 3 + 5 = 33
* Terry and Tracey: (5 x 1) + (3 x 5) + (1 x 3) = 5 + 15 + 3 = 23
* Terry and Kalid: (5 x 1) + (3 x 3) + (1 x 0) = 5 + 9 + 0 = 14

Once you have calculated the pair‐wise similarity between Terry and every other user, you can then identify whose ratings are most similar to Terry’s. In this case Bob is most similar to Terry.

1. Write a function ***get\_most\_similar\_user(current\_user\_id)*** which takes in the ***current\_user\_id*** and returns the user\_id of the user whose similarity score with the current\_user\_id is the highest. This method will look through the other users in the ratings data to find the user whose ratings are the most similar to the ***current\_user\_id***’s ratings.
2. Write a function ***recommend\_books(current\_user\_id)***  to find a set of recommendations of new books to read for a that user id. The function will return a set of recommendations in a list, where each element has the format shown below (average rating is shown in the parenthesis):



This method should use the methods already described above to find the most similar user to the ***current\_user\_id.*** The method will recommend all the books that the similar user has rated as a 3 or 5 that​ **our current userhas not read yet​.** Remember, if the user has not given any rating for that book that means they have not read the book yet.