

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.py***. 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.

1. Once you have your code running on your virtual machine (VM), submit a zip containing ***assignment10.py*** to the autograder COG .
<https://web-cog-csci1300.cs.colorado.edu>
2. You **must also** submit your code to Moodle to get full credit for the assignment.
 - 10 points: Comments at the top of your source files should include your name, recitation TA, and the assignment and problem number.
 - 10 points: Algorithm descriptions comments in your code submission to describe what your code is doing.
 - 10 points: Code in your **main()** function that is testing the code in **assignment10.py**. 10 points will be deducted if the main function in the file you submit to moodle does not have code testing the functions you have written in **assignment10.py**. Remember that COG *does not* run your main(), it runs its own main. (Code you write in main() will not impact your COG runs).

Creating a complex class

This assignment is broken up into two parts to show you the steps involved in creating a larger class implementation. The first step is to write the functions that read, parse, and store the data into variables to be used in further processing. In part one, we have listed the set of functions that concentrate on that goal and provide you with a simple set of functions that you can test to make sure they work **before** attempting to place it all in a class.

Once you have successfully written the code for the functions in the first part, you can incorporate those functions into a class. You will need to modify the functions to become class methods by changing the code that uses variables local to the function, to code that uses class member variables.

Once the methods for reading, parsing, and storing the data have been implemented, you can add new methods to the class that take advantage of that data. **Again we have listed the methods in the order that you should implement the code.** The next layer of complexity requires the calculation of a similarity between two users' ratings. This method requires that the user ratings be already available (read, parsed, and stored) for use in the calculation. Finding the most similar user to a given user requires the comparison of the given user's ratings to every other user's ratings to find the best match. The last step in creating your complex recommender system is to actually recommend books based on the ratings of books in the reading list of the most similar user.

Each of these steps has relied on the methods from the previous step to be working correctly. This strategy of writing a little bit of code and testing it, then adding more functionality and testing it, is known as a **bottom up** implementation strategy. It builds the base (most fundamental) functions and methods first, then adds new layers of complexity that takes advantage of the methods already tested. By implementing and testing each layer independently, you isolate the possible places where problems can occur and makes the debugging process easier.

Final Note

When writing your functions and class methods, you will need to write code that tests those methods. You can place your testing code anywhere in your python source file, however we suggest that you create a "my_tests()" function similar to how you added your test code into a main function in a C++ source file. Although Python does not explicitly call that function for you, you can mimic that behavior by placing a call to "my_tests()" at the bottom of your source file.

COG will import your class definition and code into our testing source file. **You should not be printing anything from your class methods, only from your own test functions.** Your test functions should only be run if you are running your file, but not if the file is being imported. We have explained the use of "if __name__ == '__main__': my_tests()" as the way to prevent your tests from running if the file is being imported.

A template assignment10.py file has been uploaded on Moodle along with this write-up. It is recommended that you use that file to complete this assignment.

Data Files used in your Recommendation System

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.**

books.txt

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

```
Douglas Adams,The Hitchhiker's Guide To The Galaxy
Richard Adams,Watership Down
Mitch Albom,The Five People You Meet in Heaven
Laurie Halse Anderson,Speak
Maya Angelou,I Know Why the Caged Bird Sings
Jay Asher,Thirteen Reasons Why
Isaac Asimov,Foundation Series
Ann Brashares,The Sisterhood of the Travelling Pants
```

ratings.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. The user_id cannot contain spaces. The ratings are listed in the order of the books supplied in the books data file.

```
Ben 5 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 1 -3 0 1 0 -5
Moose 5 5 0 5 0 0 3 0 0 1 0 5 3 0 5 0 3 -3 5 0 0 3 0 0 5 0 0 4 0 0 3 5 0 0 5 0
Ted 5 -5 0 0 0 0 -3 -5 0 1 -5 5 0 1 0 1 -3 1 5 0 0 0 0 0 0 3 0 0 0 0 -5 1 0 1 0 5
Franny 3 3 5 0 0 0 3 0 0 3 0 3 0 0 0 0 0 3 0 5 0 0 0 1 3 1 0 0 0 0 0 3 0 -3 0 0
Cust2 3 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 -3 0 0 0 0 0 0 0 5 0 0 0 0
Francois 3 3 5 0 0 0 3 0 0 3 0 3 0 0 0 0 0 3 0 5 0 0 0 1 3 1 0 0 0 0 0 3 0 3 0 0
```

The n-th rating on a line in the ratings file corresponds to the n-th line in the books file. For example, Ted 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.

A template assignment10.py file has been uploaded on Moodle along with this write-up. It is recommended that you use that file to complete this assignment.

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!

Part 1: Create functions that parse the data files and create dictionaries (30 Points)

- a) Write a function ***read_books(file_name)*** which takes in the name of a file as parameter and returns a list. Each list item will itself be a list of two items: title and author. **Note: The index of the book in this list is the book's index within the user's ratings.**

```
[ ["The Hitchhiker's Guide To The Galaxy", "Douglas Adams"],  
  ["Watership Down", "Richard Adams"],  
  ["The Five People You Meet in Heaven", "Mitch Albom"],  
  [ "Speak", "Laurie Halse Anderson"],  
  ...  
]
```

The function should return ***None*** if the file could not be read. Otherwise, it should return a list filled with data from that file. **Note: if the file is empty, you should return an empty list.**

- b) Write a function ***read_users(user_file)*** which takes in the name of a file as parameter and returns a dictionary. Each line in the file will be one entry in the dictionary. The key for this dictionary should be the user_id. The value for each entry should be a list of that user's ratings. For instance, the first element added to your dictionary would look like the following:

A template assignment10.py file has been uploaded on Moodle along with this write-up. It is recommended that you use that file to complete this assignment.

```
{
  "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],
  "Moose": [5, 5, 0, 5, 0, 0, 3, 0, 0, 1, 0, 5, 3, 0, 5, 0, 3 -3, 5, 0, 0, 3, 0, 0, 5, 0, 0,
            4, 0, 0, 3, 5, 0, 0, 5, 0],
  "Ted": [5 -5, 0, 0, 0, 0 -3 -5, 0, 1 -5, 5, 0, 1, 0, 1 -3, 1, 5, 0, 0, 0, 0, 0, 0, 3, 0,
          0, 0, 0 -5, 1, 0, 1, 0, 5],
  "Franny": [3, 3, 5, 0, 0, 0, 3, 0, 0, 3, 0, 3, 0, 0, 0, 0, 0, 3, 0, 5, 0, 0, 0, 1, 3, 1,
             0, 0, 0, 0, 0, 3, 0 -3, 0, 0],
  "Cust": [2, 3, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0 -3, 0, 0,
           0, 0, 0, 0, 0, 5, 0, 0, 0],
  "Francois": [3, 3, 5, 0, 0, 0, 3, 0, 0, 3, 0, 3, 0, 0, 0, 0, 0, 3, 0, 5, 0, 0, 0, 1, 3, 1,
                0, 0, 0, 0, 0, 3, 0, 3, 0, 0]
}
```

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. **Note: if the file is empty, you should return an empty dictionary.**

- c) Write a function ***calculate_average_rating(ratings_dict)*** which calculates the average rating of every book and returns a list. Using the ratings from every user, the average value of the ratings can be calculated. **Remember that a zero rating means the user has not read that book and the rating should not be considered in the average calculation.**

Example for the above **user dictionary**, the **average rating** will contain:

[3.83, 1.8, 5, ...]

- d) Write a function ***lookup_average_rating(index, book_dict, avg_ratings_dict)*** which takes in the book index, the book data (from ***read_books***) and the list containing the average ratings by each user (from ***calculate_average_rating***) as parameters and returns a string that is a formatted combination of the book's average rating, the title of the book, and the author.

Example:

```
lookup_average_rating(0, book_dict, avg_ratings_dict)
Output:
(3.83) The Hitchhiker's Guide To The Galaxy by Douglas Adams
```

Part 2: Create a class ***Recommender*** (70 points)

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 and lists as class data members. When you incorporate the functions created in part 1, you will need to change their behaviour slightly to set or initialize class members **(40 Points for conversion to class)**.

A template assignment10.py file has been uploaded on Moodle along with this write-up. It is recommended that you use that file to complete this assignment.

Instance Variables:

- **book_list** : list to store the books read from file.
- **user_dictionary**: dictionary which stores the users and their ratings for the books.
- **average_rating_list** : list to store average ratings

Class Methods

- **read_books(file_name)** method should add entries to a member list variable for books, instead of returning a new list.
- **read_users(user_file)** method should add entries to a member dictionary variable for ratings, instead of returning a new dictionary.
- **calculate_average_rating()** method should remove the parameter and use the member variable for ratings. The method should add entries to the member variable for average ratings, instead of returning a new list.
- **lookup_average_rating(index)** method should remove the dictionary parameters and use the appropriate member variables to return the string.

You will also provide other methods as described below to provide the recommendations of books to read for a given user.

- a. **(10 Points)** Write a method **calc_similarity(user1, user2)** which takes 2 user ids to find and return the similarity between all of the ratings of the two given users. This method is similar to the method used in the DNA assignment, as it compares the individual ratings from the two users to determine the overall similarity between two users.

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. See the example below.)*

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

```
Terry 5 3 1
Bob 5 1 5
```

The calculation for similarity between Terry and Bob will be:

➤ Terry and Bob: $(5 \times 5) + (3 \times 1) + (1 \times 5) = 25 + 3 + 5 = 33$

- b. **(10 Points)** Write a function **get_most_similar_user(current_user_id)** which takes a **current_user_id** and returns the user_id of 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. If 2 users have same (highest) similarity score, return the first one.

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

```
Terry 5 3 1
```

A template assignment10.py file has been uploaded on Moodle along with this write-up. It is recommended that you use that file to complete this assignment.

```
Bob 5 1 5
Tracey 1 5 3
Kalid 1 3 0
```

The calculation for similarity between Terry and all other users will be:

- Terry and Bob: $(5 \times 5) + (3 \times 1) + (1 \times 5) = 25 + 3 + 5 = 33$
- Terry and Tracey: $(5 \times 1) + (3 \times 5) + (1 \times 3) = 5 + 15 + 3 = 23$
- Terry and Kalid: $(5 \times 1) + (3 \times 3) + (1 \times 0) = 5 + 9 + 0 = 14$

Once you have calculated the pairwise 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.

Algorithm Hint: this function is similar to the find best match for a genome.

- c. (10 Points) Write a function ***recommend_books(current_user_id)*** to find a set of recommendations of new books to read for a given user id. The function will return a set of recommendations in a list.

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 the ***current_user_id*** has not read yet. Remember, if a user has not given any rating for a book that means they have not read that book yet

The elements of the list returned should have the format shown below (average rating is shown in the parenthesis) *Remember we have created a function already to create this format for an individual book.*

```
["(3.95) The Hitchhiker's Guide To The Galaxy by Douglas Adams",
 "(2.98) Watership Down by Richard Adams",
 "(4.3) Foundation Series by Isaac Asimov",
 "(3.26) Ender's Game by Orson Scott Card",
 "(2.87) The Lord of the Rings by J R R Tolkien"]
```

Algorithm Hint:

```
given current_user
find most similar user
for each book liked by the similar user
    if current user has not read the book
        add recommendation for this book
```

A template assignment10.py file has been uploaded on Moodle along with this write-up. It is recommended that you use that file to complete this assignment.