Project 2: Book Recommendations

CS 1410

Background

When buying things online you have probably noticed that you are presented with other items that "you might also like" or that "other customers also bought". In this project you will recommend books to a reader based on what other readers with *similar tastes* have liked.

Netflix awarded one million dollars to the winners of the Netflix Prize. The competition simply asked for an algorithm that would perform 10% better than their own algorithm. Making good predictions about people's preferences was that important to this company. It is also a very current area of research in machine learning, which is part of the area of computer science called artificial intelligence.

Objective

In this assignment you will compute book recommendations for readers based on other readers with similar tastes in books. The purpose of this assignment is to use common Python **data structures** (list, dictionary) and **file operations** in a program that is **larger** than any you may have encountered up to this point. It is important that you understand the different parts of the program and plan ahead of time how you will implement them. A suggested order for design and development appears below.

Data

First, there's a list of books in "author,title" format in the file *booklist.txt* in Files/Projects/Project 1:

Douglas Adams,The Hitchhiker's Guide To The Galaxy
Richard Adams,Watership Down
Mitch Albom,The Five People You Meet in Heaven
…

There is also a file there with user ratings for each book (*ratings.txt*):

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
Moose
5 5 0 0 0 0 3 0 0 1 0 5 3 0 5 0 3 3 5 0 0 0 0 0 5 0 0 0 0 0 3 5 0 0 0 0 0 5 -3 0 0 0 5 0 0 0 0 0 0 5 5 0 3 0 0
…

The ratings match the *index* of the book in the *booklist.txt* file. For example, the first rating of 5 from Ben applies to *Hitchhiker's Guide to the Galaxy* (`booklist[0]`), and the next 0 means Ben

hasn't read *Watership Down* (`booklist[1]`). The meaning of the rating numbers is explained in the table below.

Rating	Meaning
-5	Hated it!
-3	Didn't like it
0	Haven't read it
1	It's okay
3	Liked It
5	Really liked it!

You will determine recommendations for a reader by looking at other readers that are "close" to him or her in their tastes. This is done quite cleverly by using the **dot product** of their respective ratings as a "affinity score". A dot product of two vectors (same-size lists) is the sum of the products of list values in corresponding positions, as explained below. We will call these 2 readers the "friends" of the reader in question.

Suppose we had 3 books in our database and Rabia rated them [5, 3,-5], Suelyn rated them [1, 5,-3], Bob rated them [5, -3, 5], and Kalid rated them [1, 3, 0]. The affinity between Rabia and Bob is calculated as the dot product: (5 x 5) + (3 x -3) + (-5 x 5) = 25 - 9 - 25 = -9. The affinity between Rabia and Suelyn is: (5 x 1) + (3 x 5) + (-5 x -3) = 5 + 15 + 15 = 35. The affinity between Rabia and Kalid is (5 x 1) + (3 x 3) + (-5 x 0) = 5 + 9 + 0 = 14. So Suelyn, having the highest affinity score when compared to Rabia, is most like Rabia in her taste in books. In general, if both people like a book (rating it with a positive number) it increases their affinity score, and if both people dislike a book (both giving it a negative number), it also increases their affinity (because a negative multiplied by a negative is positive).

In the sample data above, the dot product of Ben and Moose is:

$$5 \cdot 5 + 0 \cdot 5 + 0 \cdot 0 + \cdots + 5 \cdot 3 + 5 \cdot 0 + 5 \cdot 0 = 134$$

The higher the dot product of the respective ratings of two people, the more alike are their tastes in books. In this project, you will recommend *all books* that the 2 closest "friends" (the two readers with the highest affinity scores when compared to the reader in question) liked (with a rating of 3 or 5) that the reader in question has not yet read (rating of 0).

Requirements

When your module is run as a **main** module, print a report of all book recommendations for each reader to the file *recommendations.txt*, sorted by reader name, and the books are sorted by the composite key (author last name, author first name, book title), as follows:

```
albus dumbledore: ['joshua', 'tiffany']
       ('Douglas Adams', "The Hitchhiker's Guide To The Galaxy")
       ('Dan Brown', 'The Da Vinci Code')
       ('F. Scott Fitzgerald', 'The Great Gatsby')
       ('Cornelia Funke', 'Inkheart')
       ('William Goldman', 'The Princess Bride')
```

```
        ('C S Lewis', 'The Lion the Witch and the Wardrobe')
        ('Gary Paulsen', 'Hatchet')
        ('Jodi Picoult', "My Sister's Keeper")
        ('Philip Pullman', 'The Golden Compass')
        ('Louis Sachar', 'Holes')
        ('J R R Tolkien', 'The Hobbit')
        ('J R R Tolkien', 'The Lord of the Rings')
        ('Eric Walters', 'Shattered')
        ('H G Wells', 'The War Of The Worlds')
        ('John Wyndham', 'The Chrysalids')

alexandra: ['roflol', 'shannon']
        ('Douglas Adams', "The Hitchhiker's Guide To The Galaxy")
        ('Dan Brown', 'The Da Vinci Code')
        ('Orson Scott Card', "Ender's Game")
…

zax: ['cust8', 'shannon']
        ('Douglas Adams', "The Hitchhiker's Guide To The Galaxy")
        ('Laurie Halse Anderson', 'Speak')
        ('Ann Brashares', 'The Sisterhood of the Travelling Pants')
        ('Dan Brown', 'The Da Vinci Code')
        ('Meg Cabot', 'The Princess Diaries')
        ('Cornelia Funke', 'Inkheart')
        ('William Golding', 'Lord of the Flies')
        ('Masashi Kishimoto', 'Naruto')
        ('Tite Kubo', 'Bleach (graphic novel)')
        ('Harper Lee', 'To Kill a Mockingbird')
        ('Robert Ludlum', 'The Bourne Series')
        ('Stephenie Meyer', 'Twilight Series')
        ('Gary Paulsen', 'Hatchet')
        ('Jodi Picoult', "My Sister's Keeper")
        ('Philip Pullman', 'The Golden Compass')
        ('Louis Sachar', 'Holes')
        ('H G Wells', 'The War Of The Worlds')
```

As you can see, *albus dumbledore*'s results were obtained from his top 2 friends *joshua* and *tiffany*. The recommended books are those that he hasn't yet read that *joshua* or *tiffany* rated a 3 or a 5, and similarly for *alexandra* and her top friends *roflol* and *shannon*. Note that the reader names are converted to lower case as you read them in from the file (use the **lower** method for strings). Also, no fancy formatting is required; the friends are in a list and the book entries are tuples (print the tuples preceded by a tab character: '\t').

Your program will also be tested by importing it and calling the following global functions:

- `friends(name)`

  This function returns a person's top 2 friends as a *sorted* list of strings. Sample output:

  `friends('megan')` ➔ `['roflol', 'shannon']`

- `recommend(name)`

This function returns a *sorted list* of the books recommended for the person, derived from the books liked by the top 2 friends. Remember that the index is the position of the book in the list of books as read in from booklist.txt. Sample output:

recommend('megan') ➔ `[('Meg Cabot', 'The Princess Diaries'), ('Gary Paulsen', 'Hatchet'), ('Jodi Picoult', "My Sister's Keeper"), ('Jeff Smith', 'Bone Series')]`

The list is *sorted* first by author last name, then author first name, and then by title.

- `report()`
  This function returns a *single string* that holds the full report illustrated above. This function should be called from your main function. In fact, use the following code for your main section:

```
def main():
    """ Prints recommendations for all readers """
    with open('recommendations.txt', 'w') as rec_file:
        print(report(), file=rec_file)

if __name__ == '__main__':
    main()
```

Implementation Notes

Items to consider:

1. When it comes time to recommend books for a reader, *reuven*, say, you need to compute dot products ("affinity scores") between *reuven* and *all* other readers. (You will do this for *all* readers.) You then need to determine the 2 readers with the highest dot-product scores are compared to *reuven.*
   a. How will you **store** these scores? Remember, you have to do this for *every reader*. Compute the affinity scores once only for each possible pair of readers.
   b. You also need to be able to determine the 2 readers with the highest affinity scores for each reader. How will you do this? How will you store these names?
2. When `recommend` is called, you need to determine which books the top 2 friends have rated 3 or 5, collecting only those books that the reader in question hasn't yet read. You need to return a list of tuples containing all of the recommended books sorted as explained above. Don't save this list, because it could change in a future project where we allow the books, readers and ratings to change. Just return it.

Here is a suggested sequence to develop this project incrementally:

- Read the book data into a **list** of `(author,title)` **tuples**. Do this at the module/file level.
- Read the ratings data into a **dictionary** keyed by each name (*converted to lower case*). The value for each key is a list of the ratings for that reader, preserving the original order. Do this also at the module level so the data is available when the functions mentioned below are called.
- Write a function `dotprod(x,y)`

- Compute affinity scores for each user. Store it in a data structure at the module level.
- Write the `friends` function, which returns a sorted list of the names of the two readers with the highest affinity scores compared to the reader in question. Use the **sorted** function for all sorting in this program.
- Write `recommend` by calling `friends` and then getting the recommended books from the two friends obtained.
- All of the above should execute when the module is loaded or imported, so you need to make sure you have computed the friends and affinity scores at the module level, independent of **main**. Your **main** function only prints the full report, as previously shown.
- Name your module *bookrecs.py*.

Make sure each step is complete before moving on to the next.

Note: you will need a working version of the functions described above for Program 4!