Project 1: 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.

Data

First, there’s a list of books in “author,title” format in the file *booklist.txt*:

```
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 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 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 someone 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 similarity score. A dot product of two vectors (same-size lists) is the sum of the products of corresponding list members, as explained below.

For example, 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 similarity between Rabia and Bob is calculated as the dot product: (5 x 5) + (3 x -3) + (-5 x 5) = 25 - 9 - 25 = -9. The similarity between Rabia and Suelyn is: (5 x 1) + (3 x 5) + (-5 x -3) = 5 + 15 + 15 = 35. The similarity between Rabia and Kalid is (5 x 1) + (3 x 3) + (-5 x 0) = 5 + 9 + 0 = 14. So Suelyn, having the highest similarity score when compared to Rabia, is most like Rabia in the books she likes. In general, if both people like a book (rating it with a positive number) it increases their similarity score, and if both people dislike a book (both giving it a negative number), it also increases their similarity (because a negative multiplied by a negative is positive).

So, 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 `nfriends` closest "friends" (the readers with the highest similarity score when compared to the reader in question) liked (with a rating of 3 or 5) that the reader in question has not yet read. The variable `nfriends` is a function parameter with a default value of 2 (see below).

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")
…
```

As you can see, albus dumbledore's results were obtained from his `nfriends = 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 friends roflol and shannon. Note that the reader names are converted to lower case as you read them in from the file. Also, no fancy formatting is required; the friends are in a list and the book entries are tuples (preceded by a tab character: '\t').

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

- `friends(name,nfriends=2)`

  This function returns a person's top `nfriends` friends as a sorted list of strings. The test code will call this with values other than 2 for `nfriends`. Sample output:

  ```
  friends('megan') ➔ ['roflol', 'shannon']
  friends('megan',3) ➔ ['cust8', 'roflol', 'shannon']
  ```

- `recommendations(name,nfriends=2)`

  This function returns a *sorted list* of the books recommended for the person. Remember that the index is the position of the book in the list of books as read in from booklist.txt. The test code will call this with values other than 2 for `nfriends`. Sample output:

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

  recommend('megan',3) ➔ [('Laurie Halse Anderson', 'Speak'), ('Ann
  Brashares', 'The Sisterhood of the Travelling Pants'), ('Meg Cabot',
  'The Princess Diaries'), ('Tite Kubo', 'Bleach (graphic novel)'),
  ('Gary Paulsen', 'Hatchet'), ('Jodi Picoult', "My Sister's Keeper"),
  ('Jeff Smith', 'Bone Series'), ('John Wyndham', 'The Chrysalids')]
  ```

Implementation Notes

Here is a sequence to develop this project incrementally:

- Read the book data into a list of `(author,title)` tuples
- 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
- Write a function `dotprod(x,y)`
- Compute similarity scores for each user. Map each user to another dictionary which maps the name of the other readers to his/her similarity score with the original reader. For example: `{'ben': {'moose': 134, 'reuven': 145,…}, 'reuven':{'ben': 145, 'moose': 60, 'cust1': 39,…}, …}`.
- Write the `friends` function using `heapq.nlargest` (so you don't have to sort the whole thing)
- Write `recommend` calling `friends`
- Have your main logic call `friends` and `recommend` to produce the required report.
- Name your module *bookrecs.py*