Project #4 - Recommender

How to Solve This Problem

CPSC 1020

Test data

1

- Bob/Cats/5
- Suelyn/Harry Potter/1
- Carlos/Animal Farm/1
- Kalid/Watership Down /5
- Carlos/1984/-5
- Bob/Harry Potter /3
- Suelyn/1984/5
- Carlos/Harry Potter/5
- Suelyn/Animal Farm/-3
- Bob/Lord of the Rings/5
- Kalid/Animal Farm/-3
- Kalid /Harry Potter/3

Books

2

- ['Harry Potter', 'Animal Farm', 'Lord of the Rings', 'Watership Down', 'Cats', '1984']

Ratings

3

- {'Bob': [-3, 0, 5, 0, 5, 0],
- 'Suelyn': [1, -3, 0, 0, 0, 5],
- 'Carlos': [5, 1, 0, 0, 0, -5],
- 'Kalid': [3, -3, 0, 5, 0, 0]}

Book ratings averages

4

```
{[Cats,5.00],
[Lord of the Rings,5.00],
[Watership Down,5.00],
[Harry Potter,1.50],
[1984,0.00],
[Animal Farm,-1.67]}
```

5

Processing input recommend Kalid

Sorted similarities

6

- [(12, 'Suelyn'),
- (12, 'Carlos'),
- (-9, 'Bob')]

```
Average ratings for top 3 recommenders
[0, -1.00, 5.00, 1.00, 5.00, 0]
```

7

Books to recommend based on Kalid's similarity ratings

8

- [(1.0, 'Harry Potter'),
- (5.0, 'Cats'),
- (5.0, 'Lord of the Rings')]

EXAMPLE #1

Test file: ratings_small.dat

Command: recommend Kalid

- 1) Your program will read user date from a file that contains sets of three lines. Name of this data file comes from command line. The first line will contain a username, the second line a book title and the third line the rating that user gave the book. Here is a sample file called small_ratings.dat (displayed in 2 columns for display purposes):
-

```
Bob
Cats
5
Suelyn
Harry Potter
1
Carlos
Animal Farm
1
Kalid
Watership Down
5
Carlos
1984
-5
Bob
Harry Potter
-3
Suelyn
1984
5
Carlos
Harry Potter
5
Suelyn
Animal Farm
-3
Bob
Lord of the Rings
5
Kalid
Animal Farm
-3
Kalid
Harry Potter
3
```

Test data

- Bob/Cats/5
- Suelyn/Harry Potter/1
- Carlos/Animal Farm/1
- Kalid/Watership Down /5
- Carlos/1984/-5
- Bob/Harry Potter /3
- Suelyn/1984/5
- Carlos/Harry Potter/5
- Suelyn/Animal Farm/-3
- Bob/Lord of the Rings/5
- Kalid/Animal Farm/-3
- Kalid /Harry Potter/3

- 2) When your program starts it should read through the file and create a vector with one occurrence of each book in the file. Easiest way to do this is to first create a set, add all the books, then copy the books from the set to your vector. This will eliminate duplicate book titles.
- For example, the file in the repository (small_ratings.dat) might produce the set on the previous slide. Call this vector books_vector. It is best to make sure this vector is sorted in book title order.
- Books:
- `[1984, Animal Farm, Cats, Harry Potter, Lord of the Rings, Watership Down]`

- 3) Once you have this vector of books, create a map to store the rating data. Loop through the file again. This time add each person in the file as a key to the dictionary. The value that is associated with them should be a vector the same length as the list of books you created in your first pass through the file. Name this map ratings_map.
- You should store the rating at the same index that book's name appears at in in the sorted list of books. For example, when the first three lines of the file in the repository are read, we would add a mapping from the key Bob to a value of [0, 0, 5, 0, 0, 0]. Books that the user has not rated (or whose ratings we have not read yet, as in this case) should be represented by 0s.
- Ratings:
- `{Bob,[0,0,5,-3,5,0],`
- `Carlos,[-5,1,0,5,0,0],`
- `Kalid,[0,-3,0,3,0,5],`
- `Suelyn,[5,-3,0,1,0,0]}`

- 4) Compute the average recommendation for each book. We suggest that you figure this out by building up a vector of pairs containing the book title followed by the average. You can build up this vector by going through the vector of books one at a time and for each person in the map adding up their rating of that book and counting how many people in the map rated it something other than 0. The average score for that book is the sum of scores divided by the count of non-zero ratings. Once you have created this vector you can sort it by using the vector's sort function.
- Book ratings averages
- `{[Cats,5.00],`
- `[Lord of the Rings,5.00],`
- `[Watership Down,5.00],`
- `[Harry Potter,1.50],`
- `[1984,0.00],`
- `[Animal Farm,-1.67]}`

- 5) Read command, if “recommend” then read name, call it requested_user

- recommend Kalid

- 6) The first step to do this is to calculate the similarities between the requested_user and the other recommendation_users. We will use the dot product between the users' vector of ratings to calculate their similarity. This means that we will multiply each element in your user's vector with the element at the same index in the other user's vector and sum the result. For example, if we were looking for a recommendation for Kalid we would do the following to calculate his similarity to each of the other recommenders:
- `Kalid X Bob:    (0 * 0)  + (-3 * 0) + (0 * 5) + (3 * -3) + (0 * 5) + (5 * 0) = -9`
- `Kalid X Carlos: (0 * -5) + (-3 * 1) + (0 * 0) + (3 * 5)  + (0 * 0) + (5 * 0) = -3 + 15 = 12`
- `Kalid X Suelyn: (0 * 5) + (-3 * -3) + (0 * 0) + (3 * 1)  + (0 * 0) + (5 * 0) = 9 + 3 = 12`
- Compute this similarity for each recommendation_user in the dictionary. Store pairs containing the similarity number and the name of the other recommendation_user in a vector. You can use the vector's sort function to sort.
- Note that the requested_user that you are looking for will always be in the dictionary. We are not interested in how similar the requested_user is to themselves (since the requested_user will always be most similar to themselves). You may find it helpful not to add the user to the vector or to remove them after sorting.
- Sorted Similarities
- [(12, 'Suelyn'),
- (12, 'Carlos'),
- (-9, 'Bob')]

```
{Bob,[0, 0, 5, -3, 5, 0], Carlos,[-5, 1, 0, 5, 0, 0], Kalid,[0, -3, 0, 3, 0, 5], Suelyn,[5, -3, 0, 1, 0, 0] }
```

- 7) Now that we have a list of the most similar recommendation-users, we can use this to figure out which books to recommend. To generate recommendations take an average of the ratings of the three recommendation-users with the highest similarity to the requested-user you are looking for. NOTE: do not include zero ratings in the averages
- `Suelyn: [5, -3, 0,  1, 0, 0]`
- `Carlos: [-5, 1, 0,  5, 0, 0]`
- `Bob:    [0,  0, 5, -3, 5, 0]`
- `Averages: [(5-5+0)/2, (-3+1+0)/2, (0+0+5)/1, (1+5-3)/3, (0+0+5)/1, (0+0+0)/0]`
- `Averages: [    0,         -1.00,     5.00,      1.00,       5,          0]`

- `Average ratings for top 3 recommenders`
- `[0, -1.00, 5.00, 1.00, 5.00, 0]`

- Ratings:
- `{Bob,[0,0,5,-3,5,0],`
- `Carlos,[-5,1,0,5,0,0],`
- `Kalid,[0,-3,0,3,0,5],`
- `Suelyn,[5,-3,0,1,0,0]}`

- Sorted Similarities
- [(12, ‘Suelyn’),
- (12, ‘Carlos’),
- (-9, ‘Bob’)]

- 8) Once you have calculated these averages, create a list of tuples that contain book title and the average rating for all books that have positive ratings in the averages list. Then, sort this list. Now you have a list of books to recommend.

- `Average ratings for top 3 recommends`
- `[0, -1.00, 5.00, 1.00, 5.00, 0]`
- BOOK RECOMMENDATIONS BASED ON RECOMMENDER: Kalid
- `==========`
- `Cats 5.00`
- `Lord of the Rings 5.00`
- `Harry Potter 1.00`
    - `[1984, Animal Farm, Cats, Harry Potter, Lord of the Rings, Watership Down]`

EXAMPLE #2

Test file: ratings_small.dat

Command: recommend Suelyn

- 5) Read command, if “recommend” then read name, call it requested_user

- recommend Suelyn

- 6) The first step to do this is to calculate the similarities between the requested_user and the other recommendation_users. We will use the dot product between the users' vector of ratings to calculate their similarity. This means that we will multiply each element in your user's vector with the element at the same index in the other user's vector and sum the result. For example, if we were looking for a recommendation for Kalid we would do the following to calculate his similarity to each of the other recommenders:
- `Suelyn X Bob: 5 * 0 + -3 * 0 + 0 * 5 + 1 * -3 + 0 * 5 + 0 * 0 +  = -3`
- `Suelyn X Carlos: 5 * -5 + -3 * 1 + 0 * 0 + 1 * 5 + 0 * 0 + 0 * 0 +  = -23`
- `Suelyn X Kalid: 5 * 0 + -3 * -3 + 0 * 0 + 1 * 3 + 0 * 0 + 0 * 5 +  = 12`
- Compute this similarity for each recommendation_user in the dictionary. Store pairs containing the similarity number and the name of the other recommendation_user in a vector. You can use the vector's sort function to sort.
- Note that the requested_user that you are looking for will always be in the dictionary. We are not interested in how similar the requested_user is to themselves (since the requested_user will always be most similar to themselves). You may find it helpful not to add the user to the vector or to remove them after sorting.

- SORTED SIMILARITIES
- `[(Kalid, 12),`
- `(Bob, -3),`
- `(Carlos, -23)]`

```
{Bob,[0, 0, 5, -3, 5, 0], Carlos,[-5, 1, 0, 5, 0, 0], Kalid,[0, -3, 0, 3, 0, 5], Suelyn,[5, -3, 0, 1, 0, 0] }
```

- 7) Now that we have a list of the most similar recommendation-users, we can use this to figure out which books to recommend. To generate recommendations take an average of the ratings of the three recommendation-users with the highest similarity to the requested-user you are looking for. NOTE: do not include zero ratings in the averages
- `Kalid:  [ 0, -3, 0,  3, 0, 5]`
- `Bob:    [ 0,  0, 5, -3, 5, 0]`
- `Carlos: [-5,  1, 0,  5, 0, 0]`
- `Averages: [(0+0-5)/1, (-3+0+1)/2, (0+5+0)/1, (3-3+5)/3, (0+5+0)/1, (5+0+0)/1]`
- `Averages: [  -5,         -1,         5,         1.67,      5,         5]`

- `Average ratings for top 3 recommends`
- `[-5, -1, 5, 1.67, 5, 5]`

- Ratings:
  - `{Bob,[0,0,5,-3,5,0],`
  - `Carlos,[-5,1,0,5,0,0],`
  - `Kalid,[0,-3,0,3,0,5],`
  - `Suelyn,[5,-3,0,1,0,0]}`

- SORTED SIMILARITIES
  - Kalid, 12
  - Bob, -3
  - Carlos, -23

- 8) Once you have calculated these averages, create a list of tuples that contain book title and the average rating for all books that have positive ratings in the averages list. Then, sort this list. Now you have a list of books to recommend.

- `Average ratings for top 3 recommends`
- `[-5, -1, 5, 1.67, 5, 5]`
- BOOK RECOMMENDATIONS BASED ON RECOMMENDER: Suelyn
- `==========`
- `Cats 5.00`
- `Lord of the Rings 5.00`
- `Watership Down 5.00`
- `Harry Potter 1.67`
    - `[1984, Animal Farm, Cats, Harry Potter, Lord of the Rings, Watership Down]`