Algorithms & Data Structures – 2

Final Project – Algorithm

I have made some modifications to the algorithm and the input data structures after my submission, and this modification is what I have submitted. So please treat this document as my final document

The project description says that "*You are given a list of movies (their names) and a list of similarities between movies (pairs of movies that are similar). You are also given a list of user's friends and for each friend a list of movies that he has already seen.*"

Now from the webinar and the discussion forums I understand that there is some flexibility we have been afforded in terms of choosing the data input formats and using Python package implementations for accomplishing this task.

Question – 1

So, here are my assumptions for the input data format. I'll also list down the notations which I am going to use here

1) My universe consists of M movies and N friends
2) There is a dictionary where the key is a friend and the value is a list of movies watched my the friend who is key. The number of keys will be same as the number of friends.
3) There is list of movies.
4) There is a list of friends.
5) There is a dictionary where the key is a movie and the value is a list of movies similar to the key movie. The number of keys will be same as the number of movies. This is an Adjacency List.
6) These constitute the only input data.

To accomplish this task, I detail the algorithm which I will be using below. For implementation I will be using the Pandas package because I can index the rows and columns using movie names directly rather than relying on a separate list to keep track of the movies and interact with the program through integer indices.

First, I'll import the dictionary which has the friends and the list of movies watched by each friend. It will look something like below. The name of the friend is the key and the value is the list of movies watched by that friend. Let's call this dictionary "friends_watched".

{"Friend-0": ["Movie-1", "Movie-2", "Movie-3", "Movie-4", "Movie-6"], "Friend-1": ["Movie-0", "Movie-1", "Movie-2", "Movie-3"], "Friend-2": [Movie-2", "Movie-3", "Movie-5", "Movie-6"], "Friend-3": ["Movie-0", "Movie-1", "Movie-2", "Movie-3", "Movie-4", "Movie-5"]}

Then the list of movies and the list of friends are imported. These two will look like below respectively. And call them “movies_list” and “friends_list” respectively.

0	1	2	3	4	5	6
Movie-0	Movie-1	Movie-2	Movie-3	Movie-4	Movie-5	Movie-6

0	1	2	3
Friend-0	Friend-1	Friend-2	Friend-3

This is followed by importing the dictionary of movie similarity which will look like below. The key here will be the name of a movie and the value will be list of movies similar to that movie. Let’s call this dictionary “movies_similarity”.

{“Movie-0”: [“Movie-3”, “Movie-4”], “Movie-1”: [“Movie-6”], “Movie-2”: [“Movie-5”], “Movie-3”: [“Movie-0”, “Movie-4”], “Movie-4”: [“Movie-0”, “Movie-3”], “Movie-5”: [“Movie-2”], “Movie-6”: [“Movie-1”]}

Now my data is ready. And the algorithm takes over.

We first calculate the length of “movies_list” (call it m) which is basically the number of movies. We then calculate the length of “friends_list” (call it n) which is basically the number of friends.

First I create a pandas dataframe “friends_movies” which will have columns equal to the number of movies and rows equal to the number of friends. The columns of this data frame are the names of the movies which I get from “movies_list”. The rows of this data frame are the names of the friends which I get from “friends_list”.

Now I run a scan of the dictionary “friends_watched” and then update the values of friends_movies based on the whether a friend has watched a movie. If so the value is updated to 1, else it remains as 0. Once this operation is completed the data frame “friends_movies” will look like below.

	Movie-0	Movie-1	Movie-2	Movie-3	Movie-4	Movie-5	Movie-6
Friend-0	0	1	1	1	1	0	1
Friend-1	1	1	1	1	0	0	0
Friend-2	0	0	1	1	0	1	1
Friend-3	1	1	1	1	1	1	0

To calculate F (Discussability) now, for each movie we just add up the values in the cells corresponding to that particular column for each movie. And we repeat this for each movie. We store this info into a new data frame, (call it the “Result” data frame) which has movies as its row index. We put a column name “F” to this column.

	F
Movie-0	2
Movie-1	3
Movie-2	4
Movie-3	4
Movie-4	2
Movie-5	2
Movie-6	2

Now to get the S (Uniqueness or inverse uniqueness to be precise) for each movie, the F that we have calculated and stored will come in handy. For each movie from the dictionary we imported we know the list of similar movies. And from that list we use the movie key to get to all the similar movies which have been watched by the friends. And then we calculate the total number of friends who have watched each similar movie and add that up. We then divide the figure obtained by the total number of friends (n) and take the inverse.

To see why this works, let's take an example. Say we want to find out the Uniqueness of Movie-0. We can query the dictionary and get the list of similar movies. These are Movie-3 and Movie-4, as we can see below, I have just taken the relevant portion from the main matrix and shown it here.

	Movie-3	Movie-4
Friend-0	1	1
Friend-1	1	0
Friend-2	1	0
Friend-3	1	1

Now Friend-0 has watched 2 similar movies, Friend-1 has watched 1 similar movie, Friend-2 has watched 1 similar movie and Friend-3 has watched 2 similar movies. So, the sum calculated friend wise (row wise) will be 2 + 1 + 1 + 2 and this is divided by the total number of friends. However, this sum will be equal to column wise sums too. And we know the column sums which we have already calculated stored as F in the new data frame. So, we fetch the column sums corresponding to the movies Movie-3 and Movie-4 from there and add it (like below) and then divide by the total number of friends.

	F
Movie-3	4
Movie-4	2

The result will be added as a second column into this "Result" data frame that we created. We will add a column name "S" to this row. It will look like the below. Please note that a value of 0 in the S is replaced by -1 (using a check before sending it to "Result" data frame.

	F	S
Movie-0	2	1.5

	F	S
Movie-1	3	0.5
Movie-2	4	0.5
Movie-3	4	1
Movie-4	2	1.5
Movie-5	2	1
Movie-6	2	0.75

Now we just proceed to divide “F” column by the “S” column and the result is saved in the third column of “Result” data frame with a column name “F/S”.

	F	S	F/S
Movie-0	2	1.5	1.33333333
Movie-1	3	0.5	6
Movie-2	4	0.5	8
Movie-3	4	1	4
Movie-4	2	1.5	1.33333333
Movie-5	2	1	2
Movie-6	2	0.75	2.66666667

Then we scan the third row (F/S) to get the movie with the maximum F/S score. A point to note here,

1) If there are multiple movies with same max F/S, I will choose the first movie that I encounter in the scan, basically it will be a scan of the row across all the columns and compare which one is the maximum. So, start with the F/S of Movie-0 (movie in the first column) and assign it to be maximum and keep track of the movie. Then compare to the values of each of the other movies. The maximum won’t be changed unless another movie which has an F/S greater than the current maximum is seen. Then the movie is updated as the current maximum. After the scan of the entire row, the movie with the maximum F/S is returned.
2) If the value of S is 0 for any movie, we store it as 1 (basically a small number which doesn’t really skew the calculations), so movies for which similar movies have not been watched by people will very low score for inverse uniqueness and if a lot of people have watched the movie, the F/S will be boosted because of the high Discussability.

Question – 2

I think in the first question I have explained the working of the algorithm. And the working is pretty simple. And since we can assume that the data is not invalid, I don’t see any flaws in the working of the algorithm and it will return the correct value.

Question – 3

- Data Preparation
  - The step involves creating the data frame "friends_movies" and then scanning the "friends_watched dictionary"
  - For each friend, we get the list of movies watched. This is a search against a dictionary key, so that's $O(1)$. Then based on the list of movies we will convert the values of 0 int the "friends_movies" data frame and make it 1. In the worst case if every friend has watched every movie, then it could take M operations, so worst case time complexity is $O(M)$
  - So for all the movies its $O(MN)$ where let me remind you that that N is the number of friends and M is the number of movies.

- F-Step
  - For each movie, we need to calculate the number of friends who have watched that movie. So, we have to basically go through the rows to add up the number of people who have watched the movie. So, for one movie its $O(N)$.
  - So, for all the movies its $O(MN)$.
- S-Step
  - For each movie, we get the list of similar movies. This is a search against a dictionary key, so that's $O(1)$. Then based on the list of similar movies we fetch the F values from the "Result" data frame for each of the movies in the list. Now here it's a bit tricky, because if all of our movies are similar, this will include fetching all the F values which will be $O(M)$. In Graph Theoretic terms the graph of movies will be clique with all movies connected by where edges represent similarity.
  - And if we repeat that for all the movies, the it will be $O(M^2)$. However, we need to understand that, this is a worst case and in reality the number of similar movies may be significantly less than M for each movie.
- F/S-Step
  - Here we just traverse the "Result" data frame row indexed by F/S and select the movie with the maximum F/S value. So, this is $O(M)$.
- So in total, the Time Complexity of the algorithm in the worst case is

  $$O(MN + MN + M^2 + M)$$

  And this becomes

  $$O(MN + M^2)$$

Question – 4

- We need to store the data frame with rows as friends and columns as movies.
  - Space complexity of this data frame is $O(MN)$
- We do not need to save the list of friends and the list of movies because once we assign the row indices and column indices of the above matrix we can get rid of these lists.
  - So, what would have taken $O(N)$ and $O(M)$ can be released.

- We need to save the dictionary with the similarity relationships.
    - This again in the worst case can be $O(M^2)$. However, in reality the number of similar movies may be significantly less than M for each movie.
- And finally we have the "Result" data frame which is $O(M)$
- So, in the worst case even if don't delete the list of friends and movies, we will have a worst case Space Complexity of

$$O(MN + N + M + M^2 + M)$$

Which turns out to

$$O(MN + M^2)$$