FRIEND RECOMMENDATION SYSTEM

MACHINE LEARNING CS6375

Final Project

12-09-2018

UNIVERSITY OF TEXAS AT DALLAS

Teammates:
VINEET AMONKAR (VVA180000)
SATHYARAJ NATESAN (SXN180006)
BHARAT SIMHA REDDY R S (BXR170015)

ABSTRACT:

● We will implement a friend recommendation system in python using collaborative filtering.
● We plan to use and analyze two methods for recommending friends to a user based on their common friends and influence of number of friends.
● To find the potential friends we will measure them using the 'score' metric where a higher score would mean that user is a better candidate for being friend.
● We will then test the recommendation system accuracy by removing the friendship edges of all friends in the Facebook data and then compute which method gives the most similar recommendations to that of the original data.
● We would perform many trails of randomly chosen users from the data to compute accuracies.
● A subset map of friendship of the users will be displayed using edges and nodes in the form of graphs in python.

1. Introduction:

A **recommender system** or a **recommendation system** is a subclass of information filtering system that seeks to predict the "rating" or "preference" that a user would give to an item. Recommender systems have become increasingly popular in recent years and are utilized in a variety of areas including movies, music, news, books, research articles, search queries, social tags, and products in general. There are also recommender systems for experts, collaborators, jokes, restaurants, garments, financial services, life insurance, online dating, and Twitter pages. Facebook suggests people you could be friends with. The actual algorithms used by these companies are closely-guarded trade secrets. There are two general approaches: collaborative filtering and content-based filtering.

Content based filtering:

• **Idea**: If you like an item then you will also like a “similar” item

• Based on similarity of the items being recommended

• It generally works well when it’s easy to determine the context/properties of each item. For instance, when we are recommending the same kind of item like a movie recommendation or song recommendation.

Collaborative filtering:

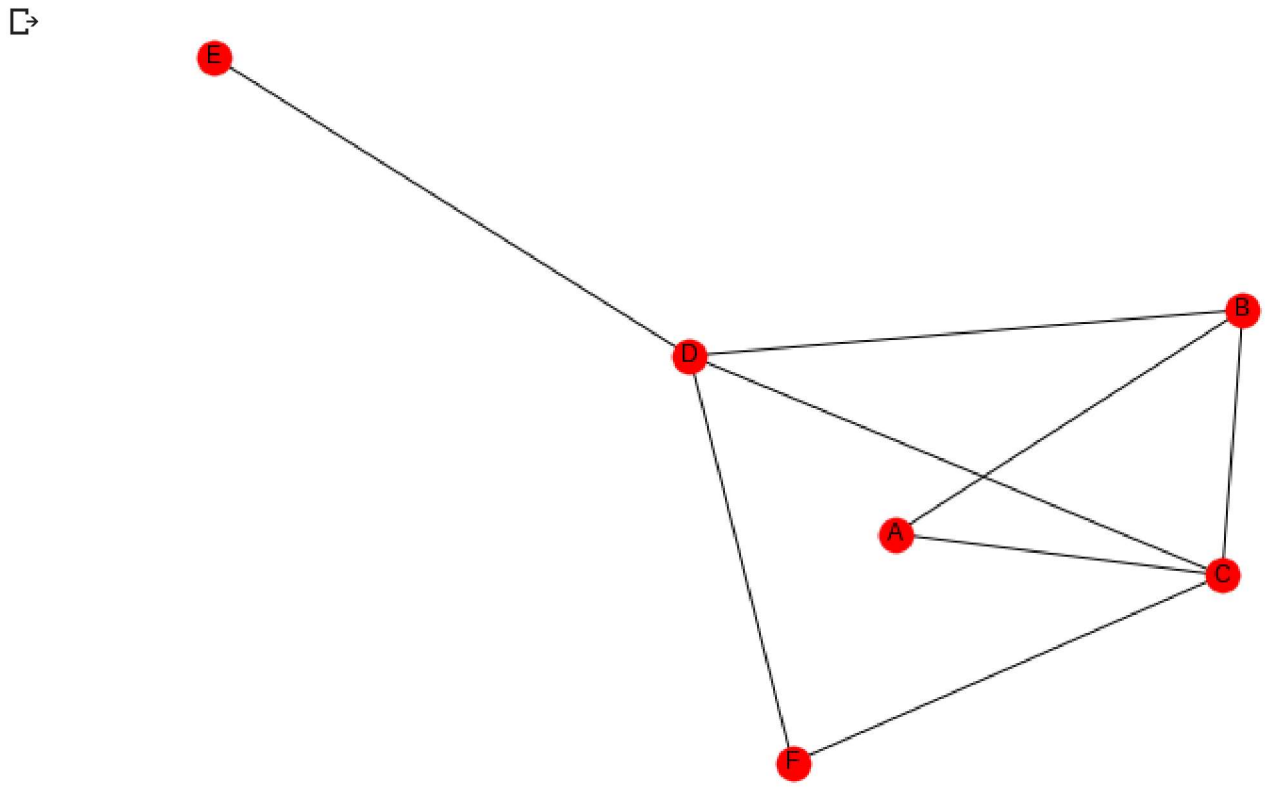
• **Idea**: If a person A likes item 1, 2, 3 and B like 2,3,4 then they have similar interests and A should like item 4 and B should like item 1.

• This algorithm is entirely based on the past behavior and not on the context. This makes it one of the most commonly used algorithm as it is not dependent on any additional information.

• For instance: product recommendations by e-commerce player like Amazon and merchant recommendations by banks like American Express . • User-User Collaborative filtering, Item-Item Collaborative filtering and Other simpler algorithms

In friend recommendation systems, the dataset contains people and their connections or friendships with other people. These connections are used as edges and people are used as nodes in a graph. The graph obtained from the dataset is called social network of friends. For a user, the recommendation system predicts the most suitable candidate for becoming the friend of the user

Practice Graph



2. Algorithm 1 - Recommend by number of common friends

2.1 Task definition:

One of the important factors while recommending new friends to a user is the number of mutual or common friends between them. As per the definition of collaborative filtering if two users have similar past preferences then their next maybe same as well, this makes common friends an important criteria. We will take graph of friends as input and output recommendation list of new friends based on number of common friends.

2.2 Algorithm definition:

The Algorithm can be easily understood by using our practice graph. There are five nodes in the graph namely A,B,C,D,E and F. An edge between any two nodes (for e.g A and B in the graph) implies that they are already friends.

Lets say we want to find new friend recommendations for node A, from the graph it is clear that A is already friends with B and C. Hence the recommendation list won't include B, C and the user himself in our case A.

We have implemented seven functions to achieve this, they are -

1) friends(graph,user)
2) friends_of_friends(graph,user)
3) common_friends(graph,user1,user2)
4) number_of_common_friends_map(graph,user)
5) number_map_sorted_list(friendmap)
6) number_map_sorted_list(friendmap)
7) recommend_by_number_of_common_friends(graph,user)

The **friends** function takes the graph and user name as input and returns a set of the friends of the user.

```
m= friends(p_graph,'A')
print(m)
```

```
{'C', 'B'}
```

The **friends_of_friends** method takes the graph and user as the input and returns a set of users who are friends of friends of the input user. The output set neither includes the input user nor any of his friends for e.g in our practice graph, A is friends with B and C who are friends with D and F hence the output set for A will contain D and F nodes.

```
n = friends_of_friends(p_graph,'A')
print(n)
```

```
{'D', 'F'}
```

The **common_friends** function take graph, user1 and user2 as input and returns the set of common friends between user 1 and user 2. In our graph A and B have one common friend C.

```
o = common_friends(p_graph,'A','B')
```

```
print(o)
```

```
{'C'}
```

The **number_of_common_friends_map** method takes the graph and user as the input and returns a map whose keys are users and values are count of common friends between them and input user. These users can neither be the the input user nor can they be friends of the input user but they should have atleast one common friend with the input user.
In our graph A has two common friends with D and 1 common friend with F. E won't be a part of the output for A because they have 0 common friends also B and C won't be included because they are already friends with A.

```
q2 = number_of_common_friends_map(p_graph,'A')
print(q2)
```

```
{'D': 2, 'F': 1}
```

The **number_map_sorted_list** method takes the map whose keys are users and values are number of common friends as input and gives a list of users as output sorted in descending order of their number of common friends.
For A the order will be D then F. If there are are same number of common friends then the users will be arranged in their natural order.

```
r = number_map_sorted_list(q2)
print(r)
```

```
['D', 'F']
```

The **recommend_by_number_of_common_friends** takes the graph and user as input. It calls number_of_common_friends_map and number_map_sorted_list methods and returns the sorted friend recommendation list for the input user as output.

From our graph we can see that the recommendation List for D only has A and the list for F has B,E and A.

```
t = recommend_by_number_of_common_friends(p_graph,'D')
print("Recommendation List For D",t)
u = recommend_by_number_of_common_friends(p_graph,'F')
print("Recommendation List for F",u)
```

```
Recommendation List For D ['A']
Recommendation List for F ['B', 'E', 'A']
```

3. Algorithm 2 - Recommend by influence:

3.1 Task definition:

Consider the following hypothetical situation.

Two of your friends are X and Y
X has only two friends (you and one other person).
Y has 7 billion friends.
X and Y have no friends in common (besides you).

Since X is highly selective in terms of friendship, and is a friend of yours, you are likely to have a lot in common with J.D.'s other friend. On the other hand, Y is indiscriminate and there is little reason to believe that you should be friendly with any one of Y's other friends.

Incorporate the above idea into your friend recommendation algorithm. Here is the concrete way that you will do so. We call the technique "influence scoring".

Suppose that user1 and user2 have three friends in common: f1, f2, and f3.

In this problem, the score for user2 as a friend of user1 is

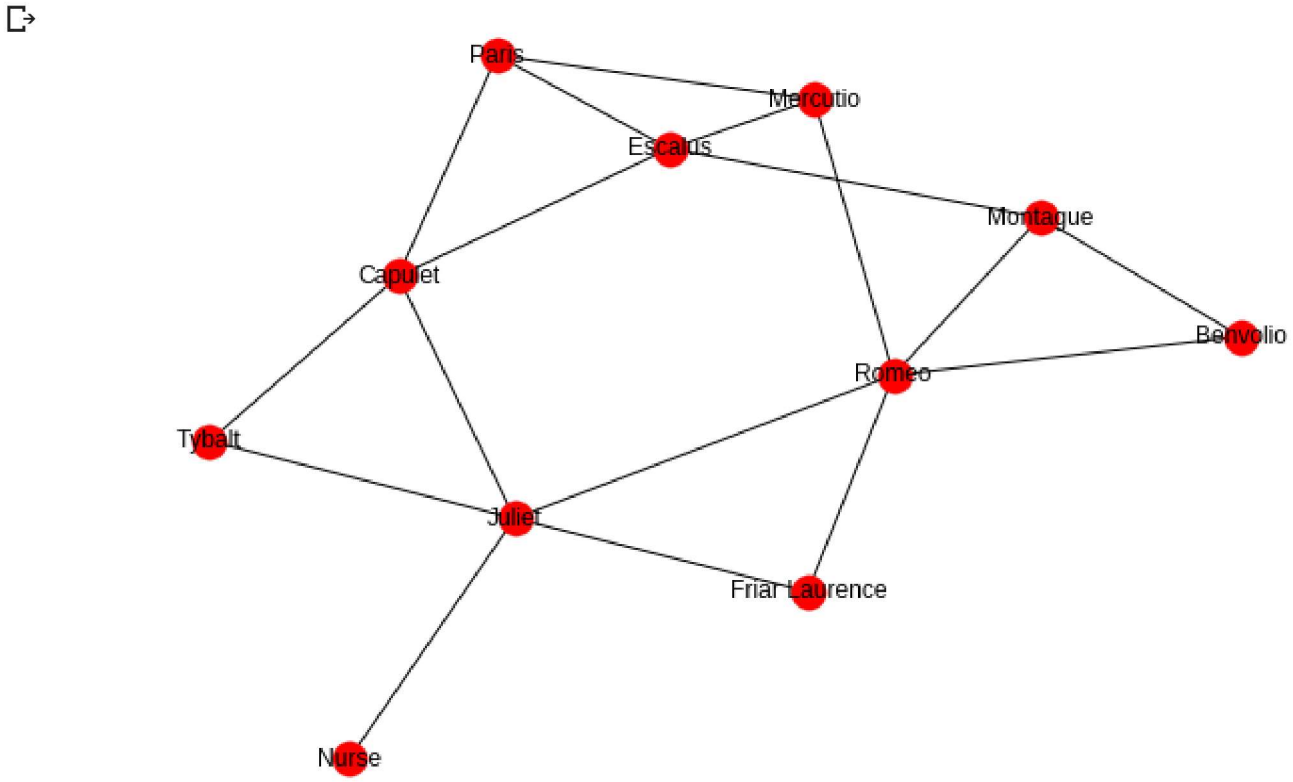
$$Score = \frac{1}{numfriends(f1)} + \frac{1}{numfriends(f2)} + \frac{1}{numfriends(f3)}$$

where numfriends(f) is the number of friends that f has.

In other words, each friend F of user1 has a total influence score of 1 to contribute and divides it equally among all of F's friends.

In the example above, X's other friend would have a score of $\frac{1}{2}$, and each of Y's friends would have a score of $\frac{1}{7 \text{ billion}}$.

```
nx.draw(rj,with_labels=True)
plt.show()
print(influence_map(rj, "Juliet"))
recommend_by_influence(rj, "Juliet")
```

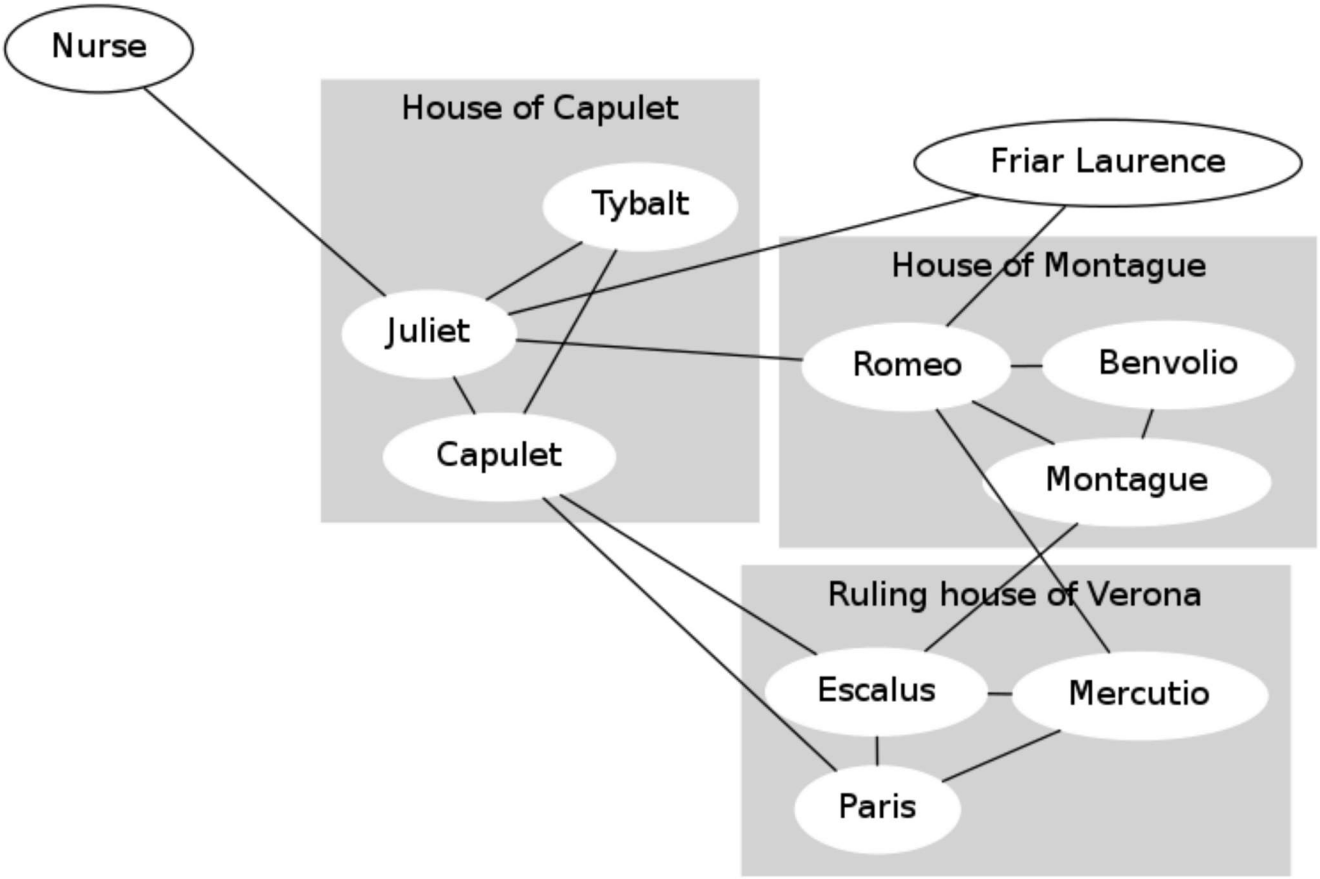



```
{'Montague': 0.2, 'Benvolio': 0.2, 'Mercutio': 0.2, 'Escalus': 0.25, 'Paris': 0.25}
['Escalus', 'Paris', 'Montague', 'Benvolio', 'Mercutio']
```

3.2 Algorithm definition:

This algorithm can also be easily understood by using a practice graph, this time we will use the friendship graph for some of the characters of “Romeo and Juliet”

Like the previous graph, an edge between person A and person B means that A considers B a friend, and also B considers A a friend.



We have implemented two methods **influence_map(graph,user)** and **recommend_by_influence(graph, user)** , the first method returns a map of users who have atleast one friend in common with the input user and are not the friends of input user. The values are their scores calculated by the score formula explained above. The Second method calls the influence_graph method and then just sorts the keys of the map according to the descending order of their values.

4. Comparing both the Algorithms

4.1 Task definition:

Now as we have implemented both the algorithms it's time to start comparing them. We execute both the algorithms on the practice graph and find out the number of users for whom both the algorithms give same recommendations and the number of users for whom the recommendations are different.

4.2 Algorithm definition:

We have implemented the diff_algo(graph) method to compare both the algorithms. It takes the graph as input and prints two separate list of users, one of which includes users having same recommendations and the other one has users with different recommendations.

```
diff_algo(rj)

```

```
unchanged =  ['Friar Laurence', 'Mercutio', 'Nurse']
changed =  ['Juliet', 'Tybalt', 'Capulet', 'Benvolio', 'Montague', 'Escalus', 'Romeo',
```

5. Loading Facebook Data

5.1 Task definition:

In this task we will create a graph of users using the facebook dataset and print the number of nodes and edges in the graph.

5.2 Algorithm definition:

We use methods create_facebook_graph and draw_facebook_graph to create and draw the graph respectively. The Facebook file name is facebook_combined.txt. It contains two columns of user id1 and user id2. Every row in the dataset implies that there is the friendship edge between the two user ids.

```
file_path = "facebook_combined.txt"
fb_graph = create_facebook_graph(file_path)
```

```
Name:
Type: Graph
Number of nodes: 4039
Number of edges: 88234
Average degree:  43.6910
```

6. Recommend by number of common friends on the Facebook data.

6.1 Task definition:

Here we will execute the recommend by number of common friends on Facebook data.

6.2 Algorithm definition:

We have implemented the top_10_common_fb() method. It uses the recommend_by_number_of_common_friends method. We will find the top 10 friend recommendation using algorithm-1 of user id 1000,2000,3000 and 4000

```
top_10_common_fb()
```

```
rec for 1000 is [1366, 1141, 1236, 1709, 1090, 1308, 1472, 1785, 1051, 1070]
rec for 3000 is [3426, 2755, 2782, 2750, 2864, 2974, 2916, 2940, 2994, 3022]
rec for 2000 is [2447, 1973, 1980, 1991, 2006, 2004, 2378, 2583, 2640, 1919]
rec for 4000 is [4014, 3982, 3988, 3993, 3997, 4004, 4019, 4023, 3983, 3985]
```

7. Recommend by influence on the Facebook data.

7.1 Task definition:

Here we will execute the recommend by influence algorithm on Facebook data.

7.2 Algorithm definition:

We have implemented the top_10_influence_fb() method. It uses the recommend_by_influence method. We will find the top 10 friend recommendation of user id 1000,2000,3000 and 4000 using algorithm-2

```
top_10_influence_fb()
```

```
1000 [1308, 1090, 1366, 1236, 1709, 1634, 1141, 1472, 1785, 1415]
3000 [2782, 2994, 2755, 2773, 2974, 2864, 3426, 2750, 2920, 2940]
2000 [1973, 2447, 1980, 2006, 1991, 2004, 2583, 2640, 2378, 1919]
4000 [4014, 3982, 3988, 3993, 4004, 4019, 4023, 3997, 3983, 3985]
```

8. How good are the recommendations for the Romeo-Juliet and Facebook data.

8.1 Task definition:

Here we will check the accuracy of both the recommendation algorithms and find which one is better for our data by computing the average rank of both the algorithms on the Facebook and Romeo-Juliet graph.

8.2 Algorithm definition:

We will test the two recommendation systems in the following way:

1. Randomly choose a real friend connection; call the two friends F1 and F2.
2. Remove their friendship from the graph.
3. Compute friend recommendations for F1 and F2.
4. Determine the rank of F1 in F2's list of recommended friends.
5. Determine the rank of F2 in F1's list of recommended friends.
6. If either of these does not exist (e.g., F1 is not recommended as one of F2's friends), discard the F1-F2 pair from the experiment.
7. Otherwise, average these two numbers.

8. The "rank" is also known as the "index" or "position". It starts counting at 1, not 0.
9. Put their friendship back in the graph.

We will repeat the above algorithm for 100 times on the Romeo-Juliet graph for both the recommendation system algorithms and compute the average rank. Based on the average rank we will decide which recommendation algorithm is better.

Similarly we will compute the average rank for both the algorithms on the Facebook data as well. The logic behind using 100 random friendship edges is to avoid the possibility of getting an incorrect result because it could be possible that for some edges algorithm 1 is better but overall it is worse than algorithm 2 or vice versa. It is very unlikely that even after randomly choosing the friendship edges we would still suffer from the above error.

We have used the following methods to compute the rank:

1) get_friend_rank : This method calculates the rank of F1 in F2's list and vice versa
2) average_rank_calc : This method is used to calculate the average rank for a pair of friends
3) common_rank_calc : This method calls average_rank_calc method to calculate rank for common friends algorithm
4) influence_rank_calc : This method calls average_rank_calc method to calculate rank for influence friends algorithm
5) evaluate_recommender : This method computes the rank for 100 pairs of friendship for both the algorithms using the above methods and then decides which algorithm out of common friends and influence is better.

```
import random
import sys
evaluate_recommender(rj)
evaluate_recommender(fb_graph)
```

```
Average rank of influence method: 2.0259740259740258
Average rank of number of friends in common method: 2.383116883116883
Influence method is better
Average rank of influence method: 44.80102040816327
Average rank of number of friends in common method: 53.255102040816325
Influence method is better
```

9. Conclusion:

After testing our recommendation algorithms on both the dataset we can see that Influence method is better for both the Romeo-Juliet graph and Facebook graph than the common friends algorithm since the average computed rank is less for the influence method than the number of common friends algorithm. To further enhance our friend recommendation system we can also use the time at which the friendship was formed between the two users on Facebook, this is the future scope of our project.