Mining Big Datasets

Assignment 1: Recommending Friends

by

Dimitrios-Gerasimos Anastasatos

Athens, June 2018

1) Dataset

The original dataset is the ego-Facebook dataset from the link below: https://snap.stanford.edu/data/egonets-Facebook.html This dataset consists of 'circles' (or “friends” lists') from Facebook. Facebook data was collected from survey participants using this Facebook app. The dataset includes node features (profiles), circles, and ego networks. This dataset consists of 4039 nodes (users) and 88234 edges (friendships between users). The graph is connected and directed (each edge counts as friendship for both nodes). As we need to transform the graph to undirected by adding the missing edges, we do the following steps:

We load the txt file through Jupyter Notebook (Python 3) as 2-column data frame where column 0 is node 1 and column 1 is node 2. In order to transform it to an undirected one, we created a new dataframe, called fd, by swapping the columns of the original one, df. Then we appended the two data frames together.

2) Recommending friends using Common neighbors (friend-of-friend (FoF) method)

For recommending friends using the FoF method, we created a function called fof with three arguments: network (an undirected graph in dataframe format), nodeID (the ID of the node we want to suggest friends to), recommendations (the number of friends we want to suggest). The fof steps are the following:

1. Create a list of friends of the node we want to suggest friends to.
2. Create a list of nodes that are not friends with the above node.
3. Find each non-friend’s node friends and add them in a list.
4. Calculate the length of the intersection of friends of the target node (i.e. node we want to suggest friends to) and every other from the non_friends list.
5. Sort the nodes based on the above length - serves the score metric in the FoF method - and keep the ten with the highest one. In case of ties, prioritize nodes with a smaller ID.

After running the fof function for nodes 107, 1126, 14, and 35, we end up to the bellow friend suggestions (presented in a descending order - from highest score node to lower score node):

target node	**Friends suggestions**									
	1st	**2nd**	**3rd**	**4th**	**5th**	**6th**	**7th**	**8th**	**9th**	**10th**
107	513	400	559	373	492	500	378	436	431	514
1126	916	1238	1750	1230	1004	1791	1530	1172	1570	1597
14	2	17	140	111	137	162	19	333	44	243
35	46	68	99	131	175	177	225	227	278	321

3) Recommending friends using Jaccard coefficient

For recommending friends using the Jaccard coefficient, we created a function called jaccard_coef with three arguments: network (an undirected graph in dataframe format), nodeID (the ID of the node we want to suggest friends to), recommendations (the number of friends we want to suggest). The fof steps are the following:

6. Create a list of friends of the node we want to suggest friends to.
7. Create a list of nodes that are not friends with the above node.
8. Find each non-friend's node friends and add them in a list.
9. Calculate the fraction of the length of the intersection of friends of the target node (i.e. node we want to suggest friends to) and every other from the non_friends list, over the length of the union of target node and every other from the non_friends list.
10. Sort the nodes based on the above fraction - serves the score metric known as the Jaccard coefficient - and keep the ten with the highest one. In case of ties, prioritize nodes with a smaller ID.

After running the jaccard_coef function for nodes 107, 1126, 14, and 35, we end up to the bellow friend suggestions (presented in a descending order - from highest score node to lower score node):

target node	**Friends suggestions**									
	1st	**2nd**	**3rd**	**4th**	**5th**	**6th**	**7th**	**8th**	**9th**	**10th**
107	513	400	492	559	373	378	346	500	431	514
1126	916	1750	1230	1530	1004	1238	1172	1791	1789	1597
14	2	140	17	162	111	333	44	137	19	243
35	321	11	12	15	18	37	43	74	114	209

4) Recommending friends using Adamic and Adar function

For recommending friends using the Jaccard coefficient, we created a function called adamic_adar with three arguments: network (an undirected graph in dataframe format), nodeID (the ID of the node we want to suggest friends to), recommendations (the number of friends we want to suggest). The fof steps are the following:

11. Create a list of friends of the node we want to suggest friends to.
12. Create a list of nodes that are not friends with the above node.
13. Find each non-friend's node friends and add them in a list.
14. Create a graph of the data frame network
15. Calculate the Adamic & Adar scores for each non-friend node. This scoring function evaluate the likelihood for a node A to be linked with another node B, by summing the number of neighbors the two users have in common. Items that are unique to a few nodes are weighted more than commonly occurring items. The weighting scheme uses the inverse log frequency of their occurrence.

16. Sort the nodes based on the above fraction - serves the score metric known as the Jaccard coefficient - and keep the ten with the highest one. In case of ties, prioritize nodes with a smaller ID.

After running the jaccard_coef function for nodes 107, 1126, 14, and 35, we end up to the bellow friend suggestions (presented in a descending order - from highest score node to lower score node):

target node	**Friends suggestions**									
	1st	**2nd**	**3rd**	**4th**	**5th**	**6th**	**7th**	**8th**	**9th**	**10th**
107	513	400	559	500	492	373	378	436	524	514
1126	916	1238	1750	1230	1004	1791	1530	1172	1570	1597
14	2	17	140	111	162	137	133	19	44	243
35	46	68	99	131	175	177	225	227	278	321

5) Evaluation of the Recommendation System

For the first step, in order to evaluate the recommendation system, for each one of the 40 users whose ids are multiple of 100, we compute the similarity percentage between pairs of scoring functions. More specifically for each pair of scoring functions, e.g FoF - Jaccard, we call the fof function that we created previously and the jaccard function correspondly for each user. The output of each function is a list of recommended friends by the method. We then transform the output into sets and take the intersection (common recommended friends) of two sets. Then in a different list we keep for each user the similarity percentage between the two scoring functions, that is the length of the intersection set divided by the total length of nodes (ten as we examine the top ten) and multiplied by 100 to result the p%. The results of the described procedure are following :

The similarity percentage for each one of the 40 users is: [20, 90, 60, 80, 90, 10, 30, 90, 70, 60, 80, 0, 50, 20, 70, 50, 10, 100, 10, 70, 70, 90, 40, 10, 90, 100, 30, 90, 20, 70, 60, 50, 70, 60, 60, 0, 20, 50, 90, 90]%

Moreover, from the previous results we compute the average similarity between FoF and Jaccard methods. The result is:

The average similarity between FoF and Jaccard Coefficient is: 55.5%

For Jaccard - Adamic:

The similarity percentage for each one of the 40 users is: [30, 80, 60, 80, 80, 10, 30, 90, 70, 70, 80, 30, 40, 20, 70, 50, 10, 100, 20,

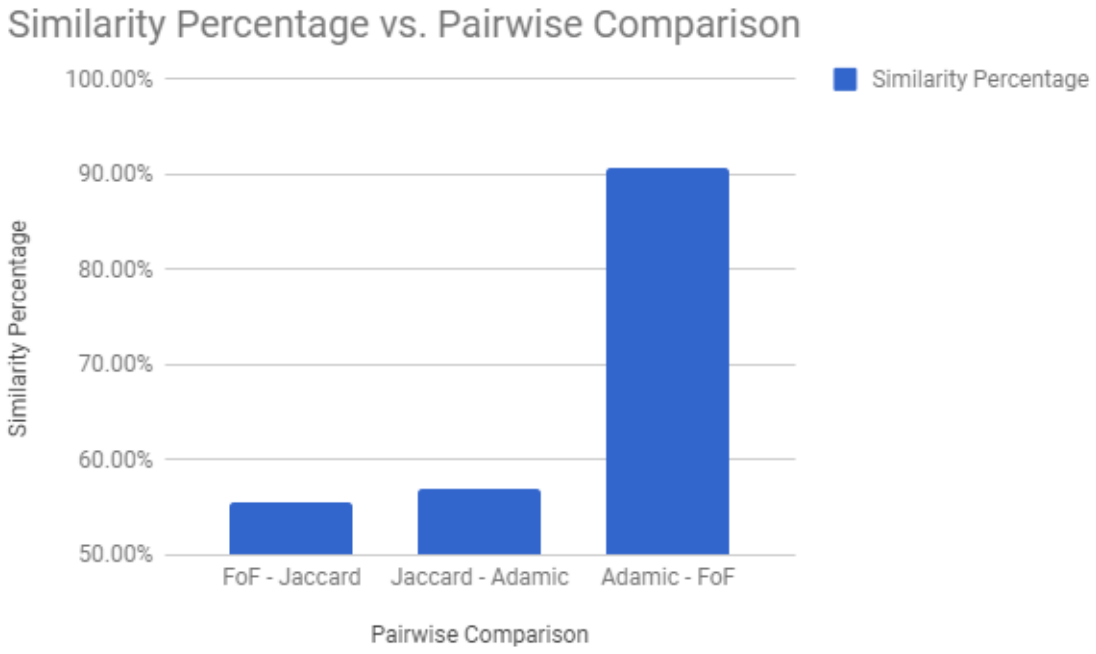
70, 60, 90, 40, 20, 90, 100, 30, 90, 30, 70, 60, 50, 80, 60, 70, 0, 20, 50, 90, 90]%

The average similarity between Jaccard and Adamic is 57.0%

For Adamic - FoF:

The similarity percentage for each one of the 40 users is
[50, 90, 100, 100, 90, 100, 90, 100, 90, 80, 100, 30, 90, 100, 100, 100, 90, 100, 80, 100, 80, 100, 100, 60, 100, 100, 80, 100, 90, 90, 80, 100, 90, 100, 90, 90, 100, 100, 100, 100]%

The average similarity between Adamic and FoF is 90.75%



As we notice from the above graph, Adamic and Friend-of-Friend Methods have the highest similarity between them (over 90%) for the 40 users under examination. In other words, these two methods mostly recommend the same nodes as friends to the top ten recommended list.

Next, as for the second step of the evaluation of the recommendation system, we implement the given algorithm to forecast if the recommendations are going to be accepted from the users. We iteratively run the algorithm until we achieve 100 successfully recommended friendships between F1 and F2. Hence, the average index of F1 and F2's list is for each scoring method:

Scoring Method	Average Index (Ascending order)
Adamic	1.85
Resource Allocation (*)	1.86

FoF	1.94
Jaccard	2.33

(*) Resource allocation method will be discussed later in the Bonus Session.

As we work in Python and indexing starts counting from 0, the indexes in the top ten list are from 0 to 9. In order for the results to be more reflecting of the actual ranking, we add to the average index the number 1 (average function has the linear property) and so we present the average rank (in 1-10 scale) forecasted from each method, meaning that 1 is the most highly ranked node, and 10 the least. The results are following:

Scoring Method	Average Rank (Ascending order)
Adamic	2.85
Resource Allocation (*)	2.86
FoF	2.94
Jaccard	3.33

From the table above we notice that the best forecast for friend recommendation is given by the Adamic (and Resource Allocation) Method because on average it ranks a node (that a priori a friend) to the target node higher than the other methods.

6)

The scoring function we have implemented is called resource allocation*. According to this method each the scoring function evaluates the likelihood for a node u to be linked with another node v, by summing the number of neighbors the two of them have in common. Items that are unique to a few nodes are weighted more than commonly occurring items. The weighting scheme uses the inverse frequency of their occurrence. Resource allocation index of u and v is defined as:

$$\sum_{w \in \Gamma(u) \cap \Gamma(v)} \frac{1}{|\Gamma(w)|}$$

where Γ(u) denotes the set of neighbors of u.

After running the resource_allocation_index for nodes 107, 1126, 14, and 35, we end up to the bellow friend suggestions (presented in a descending order - from highest score node to lower score node):

target node	Friends suggestions									
	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
107	3263	513	400	173	500	3278	630	465	3290	559
1126	916	1238	1750	1230	1004	1791	1172	1570	1530	1597
14	2	17	140	111	162	333	137	19	44	243
35	46	68	99	131	175	177	225	227	278	321

Regarding the evaluation method, we suggest a similar to the proposed one with a variant evaluation metric. Instead of using the average rank a friend of the node has as a non-friend, we count in how many out of 100 cases a

friend of a node is included in the top ten friend recommendations as a non-friend. Specifically, the suggested algorithm works as follows:

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 (10 recommendations).
4. Determine whether F1 is in F2's list of recommended friends. If it is, store the value 1, else store the value 0. Determine whether F2 is in F1's list of recommended friends.If it is, store the value 1, else store the value 0.
5. Put their friendship back in the graph.
6. For each scoring function, perform the above experiment 100 times.
7. Compute the probability of a real friendship to be recommended when we artificially remove it as the proportion of all the times it has occurred (e.g. the summation of 1s) over the iterations (here, 100). Do that for every scoring function.
8. To prevent different random choices from skewing the results, we use the same random choices for all similarity functions.

After running the above evaluation algorithm, we end up with:

Scoring Method	**Friendship probability (Ascending order)**
Resource Allocation (*)	70%
Adamic & Adar	68.5%
FoF	68.5%
Jaccard	65.5%