Written Component

All submissions, regardless of which Option chosen, need to include a written component that covers the following items:

Your details:

○ What is your username on Kaggle and what is your uniqname (if we can't link you to your submission, we have to give you a 0).

Username: Yuan Li

○ Which libraries you used (if any)

I used the networkx libraries.

Your Approach:

○ The description of your final submission's methods and discussion of why you decided on these methods. What algorithm did you implement? What parameters did you use? The description is intended to be human-readable such that another person in class could read it and feasibly understand how you implemented your system and *why* you made the choices you did. Learning to justify your design and choices is a critical part of the discussion and something you will need to do as a practicing data scientist. Your description can include some pseudocode but should not include lots of code.

Link Prediction: (detail description can also see the iphython file)

- I first created a subgraph based on random-sampled nodes from network file (sample node size = 2000). Then I add the nodes from network to make the randomly-selected subgraph can be more representative of the whole network. After the graph is constructed, I computed the distance among each nodes based on Jaccard Coefficients, and finally output the prediction results to a csv file.

Attribute Prediction:

- For attribute prediction, I construct the graph using similar method as what I did in task 1. But instead of randomly selected nodes from network file, I chose the nodes particularly from test data first to construct the network, because I think that's what we actually care about and ultimately want to get the results. However, simply using test node is not enough, since we would want our test node to be around the training nodes, so that we can predict the test node attributes based on training nodes' attributes. So I add the training nodes when they are either in the test-node subgraph or in its neighborhood.

Then I get a list of neighboring nodes and use the "majority vote" intuition to decide which attributes should the target test node adopt.

- I specify the attributes of a target node as adoption from its surrounding nodes. If the attribute type from its neighbors are unique, then the target node would adopt that attribute with value from its neighbor; if the attribute type is not unique, then I would evaluate based on the majority of its values, to decide the specific value for that type of attribute in our target node.

- Finally, output the predictions to file. Once I get all the batches of predictions, I combined all of them locally by copying and pasting in a txt file, and clean it and reformat it as required in the instruction in another ipython script (see "formatting.ipython").

○ What other approaches you tried and how well they did, presented in a Table. You can include a short description of each if you want to comment. Different parameter variations should count in this table. Your work here shows effort and will be counted towards the overall grade.

Besided Jaccard Coefficient, I have also several other methods for link prediction, and their performances are as below:

Jaccard Coefficient	Process took 6.3311 seconds
Preferential Attachment	Process took 0.8533 seconds
Adamic adar index	Process took 7.7301 seconds
Resource Allocation	Process took 7.2603 seconds

Comparing the performances, Preferential Attachment seems to take the shortest time to predict, and Jaccard Coefficient comes after; while Adamic adar and Resource allocation take slightly longer time to get the results.

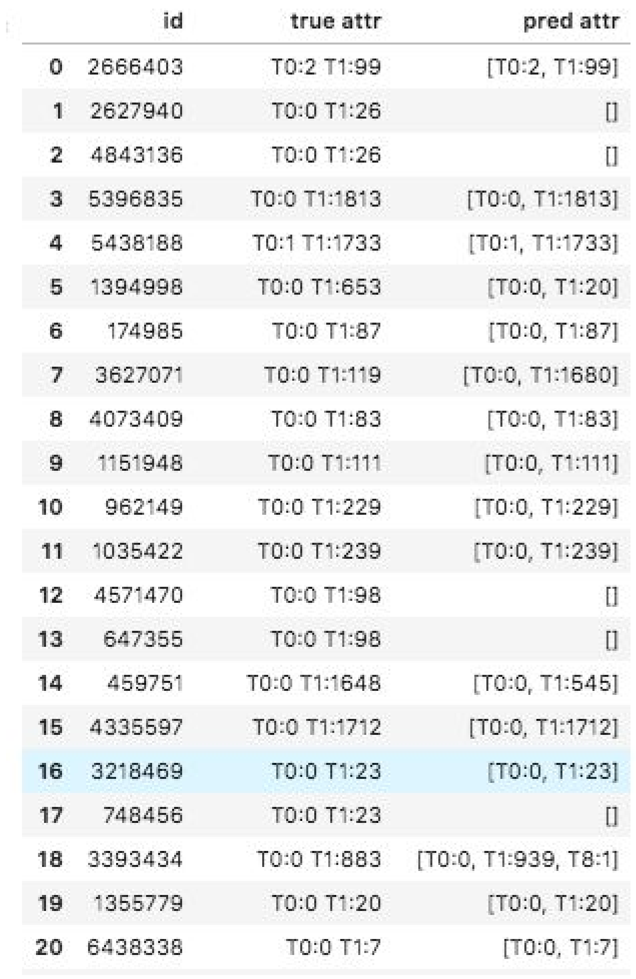
● **Your Analyses:**

○ How much does homophily hold in this dataset? Define a metric to measure how similar are two user's attributes and report on the distribution of how similar all (or a large sample) of users are.

This dataset seem to have quite a lot of nodes share similar attributes, where homophily can be seen. One metric that could measure the similarity of two users' attributes is, to see how many different types of attributes shared among users, and see how close the values of that particular type of attributes are, for example, T1: 171 and T1:175 are closer than T1:171 and T1:455.

○ Which attribute types are hard to predict? Using the development data, perform an error analysis and report your expected error rate for each type.

(detail evaluation can see: Evaluation.ipython)
To conduct an error analysis, I read the dev file, and take the first 50 records as sample to test the error rate. And the result screenshot looks like this:



	id	true attr	pred attr
0	2666403	T0:2 T1:99	[T0:2, T1:99]
1	2627940	T0:0 T1:26	[]
2	4843136	T0:0 T1:26	[]
3	5396835	T0:0 T1:1813	[T0:0, T1:1813]
4	5438188	T0:1 T1:1733	[T0:1, T1:1733]
5	1394998	T0:0 T1:653	[T0:0, T1:20]
6	174985	T0:0 T1:87	[T0:0, T1:87]
7	3627071	T0:0 T1:119	[T0:0, T1:1680]
8	4073409	T0:0 T1:83	[T0:0, T1:83]
9	1151948	T0:0 T1:111	[T0:0, T1:111]
10	962149	T0:0 T1:229	[T0:0, T1:229]
11	1035422	T0:0 T1:239	[T0:0, T1:239]
12	4571470	T0:0 T1:98	[]
13	647355	T0:0 T1:98	[]
14	459751	T0:0 T1:1648	[T0:0, T1:545]
15	4335597	T0:0 T1:1712	[T0:0, T1:1712]
16	3218469	T0:0 T1:23	[T0:0, T1:23]
17	748456	T0:0 T1:23	[]
18	3393434	T0:0 T1:883	[T0:0, T1:939, T8:1]
19	1355779	T0:0 T1:20	[T0:0, T1:20]
20	6438338	T0:0 T1:7	[T0:0, T1:7]

Comparing the true attributes and the predict attributes, I can see that one main problem is that some of the pred attr are missing, simply because when I built the graph, I only include those training nodes that are perceived as neighbors of the dev nodes. This make the prediction not accuracy enough. So one way I can think of to improve, is to build graph based on training nodes, and then add nodes from test data (or dev data). I have tried that, but unfortunately I don't have enough time to fix this problem and to get the results. Also, I notice that some types are especially hard to predict than the others, e.g., T1:26, T1:98, T1:115, etc.