Part 1. Select data and create network

I'm choosing the first option: treat papers as nodes and citation relationships as edges.

First, I'm using all the datasets, 4 json file and 4 csv files, and combine them together. The shape of data is: (3079007, 8) from json files, and (25166994,2) from csv files, which is a lot. Merge these two datasets to make sure references is not a list type.
Then, to reduce the dimension of data, I want to select the articles that are widely used. I select data for recent 4 years because I want to choose recent research. Also, I select data posted in venue, articles in venue might me more professional. Then, select n_citations and number of references >= 20, a widely used article should be cited by other papers, and use other's related works.

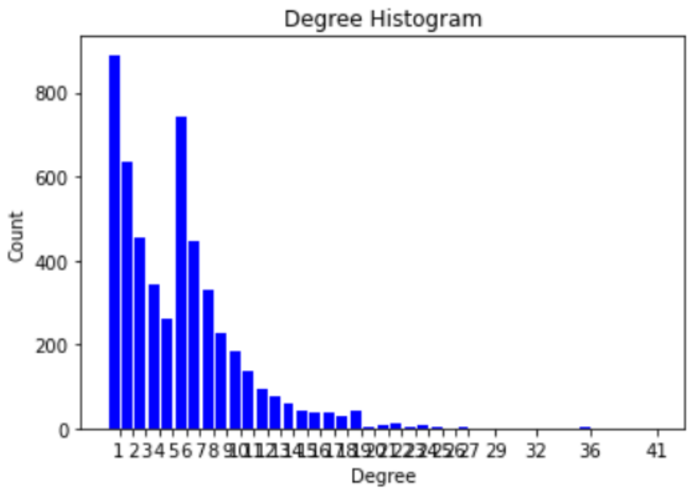
My dataset still has 164065 rows. By subset paper in references column, with occurrence between 5 and 20, my final dataset has 14609 rows. I choose occurrence greater than 5 because the cited paper should also be popular. Choose occurrence smaller than 20, for reduce the high computational cost.

Part 2 network analysis, reporting basic statistics

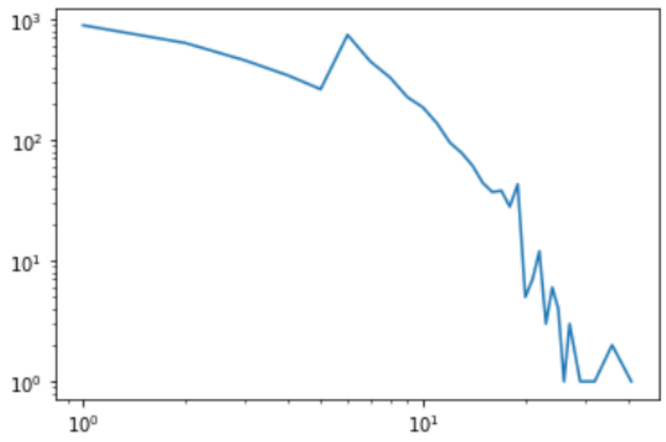
1) number of nodes in the network: 5123
2) number of edges in the network: 14579
3) average degree of node: 5.6916
4) radius of the network: 10
5) diameter of the network: 18
6) density of the network: 0.001

My network has a low number of radius, because I select a subset of reference paper's occurrence < 20, which reduce computational cost.

Histogram



It has a clear distribution of degree. The log-log plot shows degree distribution by another viewpoint.
(by library matplotlib.pyplot)



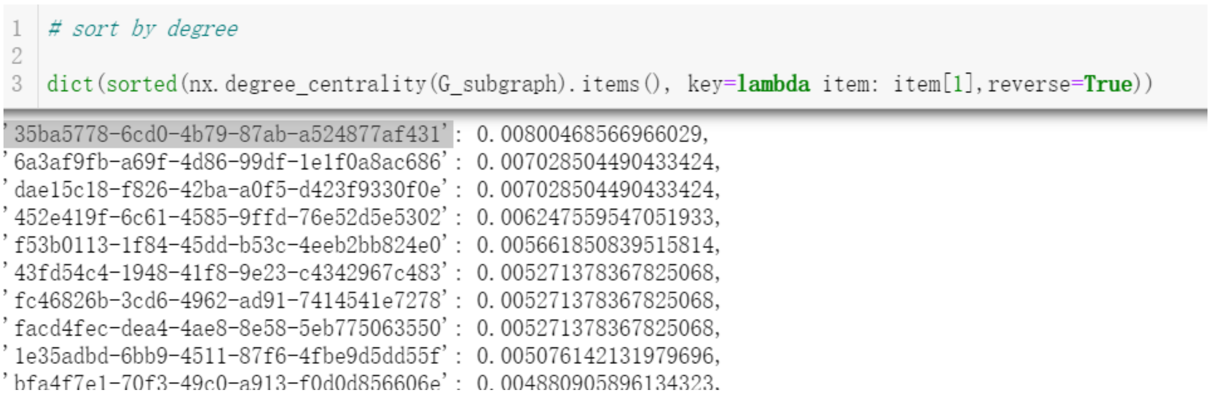
Log–log plot is a two-dimensional graph of numerical data that uses logarithmic scales on both the horizontal and vertical axes. This graph helps show other point's degree. When degree = 1, the value is large. Both Log-log plot and histogram shows a decreasing trend of degree distribution, when number of degree increases.

Part3 node centrality analysis

Degree Centrality:
Degree centrality assigns an importance score based simply on the number of links held by each node. I want to find the individuals, who can quickly connect with the wider network.

The average of degree centrality is 0.0011.

By sort the average of degree centrality according to different nodes, the paper has id of '35ba5778-6cd0-4b79-87ab-a524877af431' is the best, which has 0.0080.

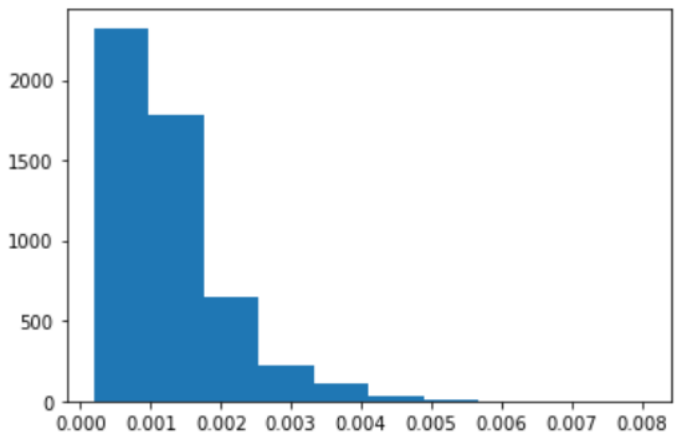


```
# sort by degree

dict(sorted(nx.degree_centrality(G_subgraph).items(), key=lambda item: item[1],reverse=True))
```

```
'35ba5778-6cd0-4b79-87ab-a524877af431': 0.008004685669660 29,
'6a3af9fb-a69f-4d86-99df-1e1f0a8ac686': 0.007028504490433424,
'dae15c18-f826-42ba-a0f5-d423f9330f0e': 0.007028504490433424,
'452e419f-6c61-4585-9ffd-76e52d5e5302': 0.006247559547051933,
'f53b0113-1f84-45dd-b53c-4eeb2bb824e0': 0.005661850839515814,
'43fd54c4-1948-41f8-9e23-c4342967c483': 0.005271378367825068,
'fc46826b-3cd6-4962-ad91-7414541e7278': 0.005271378367825068,
'facd4fec-dea4-4ae8-8e58-5eb775063550': 0.005271378367825068,
'1e35adbd-6bb9-4511-87f6-4fbe9d5dd55f': 0.005076142131979696,
'bfa4f7e1-70f3-49c0-a913-f0d0d856606e': 0.004880905896134323,
```

I'm using histogram to check the centrality value works in my network:



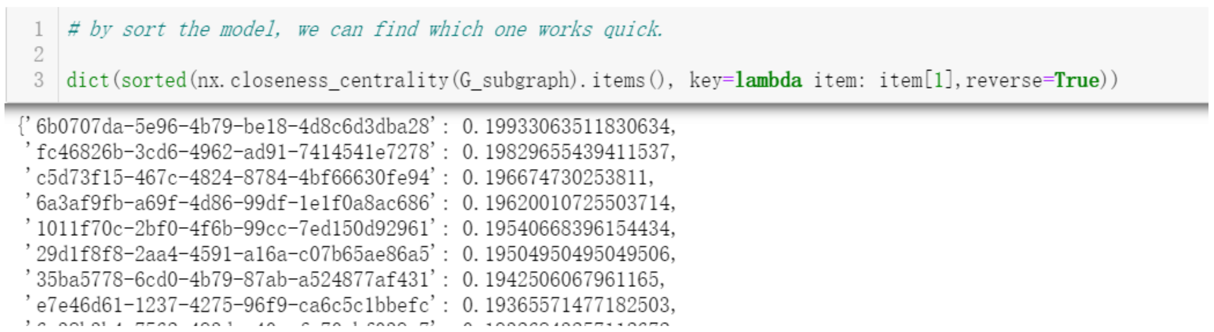
It shows most points has a low degree centrality. So, my network is not enough central. By the sorted degree certainty， I found the individuals who can quickly connect with the wider network.

Closeness centrality

Closeness centrality scores each node based on their 'closeness' to all other nodes in the network. It helps me to find individual who works quickly.

The average of closeness centrality is 0.15147.

By sort the average of closeness centrality according to different nodes, the paper has id of ' 6b0707da-5e96-4b79-be18-4d8c6d3dba28' is the best, which has 0.19933.

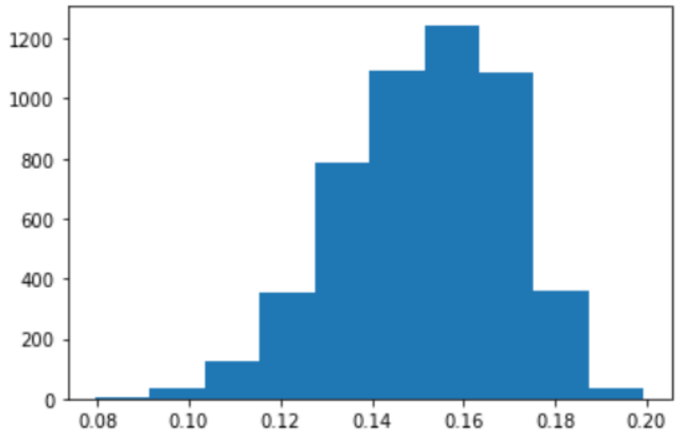


```
# by sort the model, we can find which one works quick.

dict(sorted(nx.closeness_centrality(G_subgraph).items(), key=lambda item: item[1],reverse=True))
```

```
{'6b0707da-5e96-4b79-be18-4d8c6d3dba28': 0.19933063511830634,
 'fc46826b-3cd6-4962-ad91-7414541e7278': 0.19829655439411537,
 'c5d73f15-467c-4824-8784-4bf66630fe94': 0.196674730253811,
 '6a3af9fb-a69f-4d86-99df-1e1f0a8ac686': 0.19620010725503714,
 '1011f70c-2bf0-4f6b-99cc-7ed150d92961': 0.19540668396154434,
 '29d1f8f8-2aa4-4591-a16a-c07b65ae86a5': 0.19504950495049506,
 '35ba5778-6cd0-4b79-87ab-a524877af431': 0.1942506067961165,
 'e7e46d61-1237-4275-96f9-ca6c5c1bbefc': 0.19365571477182503,
```

I'm using histogram to check the centrality value works in my network:



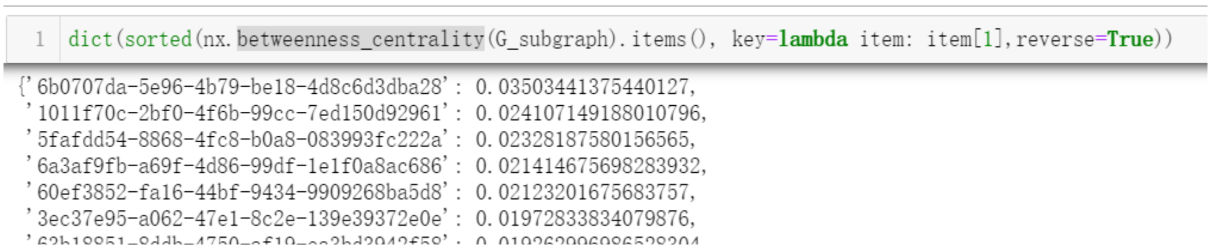
Most nodes have closeness centrality about 0.15. It shows the distribution of the nodes' centrality value, looks like normal distribution. The larger the value it is, the nodes work quicker. I can choose the best nodes which has the best closeness certainties as shown above.

Betweenness centrality

Betweenness is the degree the node lies on the shortest path between two other nodes. By betweenness, we can find which node influence the system in the network.

The average of betweenness centrality is 0.0011.

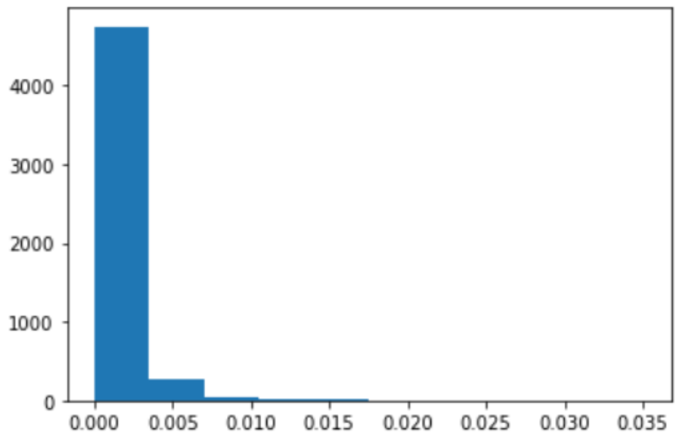
By sort the average of betweenness centrality according to different nodes, the paper has id of ' 6b0707da-5e96-4b79-be18-4d8c6d3dba28' is the best, which has 0.035.



```
dict(sorted(nx.betweenness_centrality(G_subgraph).items(), key=lambda item: item[1],reverse=True))
```

```
{'6b0707da-5e96-4b79-be18-4d8c6d3dba28': 0.03503441375440127,
 '1011f70c-2bf0-4f6b-99cc-7ed150d92961': 0.024107149188010796,
 '5fafdd54-8868-4fc8-b0a8-083993fc222a': 0.02328187580156565,
 '6a3af9fb-a69f-4d86-99df-1e1f0a8ac686': 0.021414675698283932,
 '60ef3852-fa16-44bf-9434-9909268ba5d8': 0.02123201675683757,
 '3ec37e95-a062-47e1-8c2e-139e39372e0e': 0.01972833834079876,
 '63b18851-8ddb-4750-af19-ee3bd3942f58': 0.01926299698652830
```

I'm using histogram to check the centrality value works in my network:



Most nodes have a low value for betweenness, which means most of the nodes does not influence the system enough. By the sorted centrality, I found the best nodes which has great influence to the network.

Overall, closeness has a relatively higher number, but degree and betweenness are low. Which means key player tied to important players, and probably multiple paths in the network, ego is near many people, but there is many others.

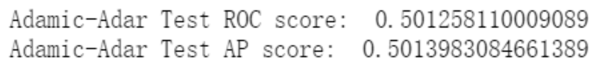
Part 4 Link predictions

For the score methods, I'm choosing to use ROC-AUC score and AP score. ROC-AUC score is a classification-based evaluation metric, given by true positive rate divide by true negative rate. AP, Average Precision, is a measure that combines recall and precision for ranked retrieval results, which is a rank-based evaluation metric.

Adamic-Adar (unsupervised method)

The idea for Adamic-Adar is common elements with large neighborhoods are less significant when predicting a connection between two nodes compared to elements shared between a small number of nodes. It's defined as the sum of the inverse logarithmic degree centrality of the neighbours shared by the two nodes.

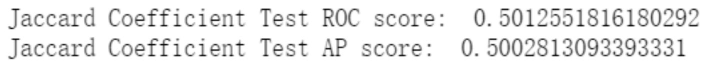
Score methods: ROC and AP



```
Adamic-Adar Test ROC score:  0.501258110009089
Adamic-Adar Test AP score:  0.5013983084661389
```

Jaccard Coefficient: (unsupervised method)
The Jaccard coefficient measures similarity between finite sample sets and is defined as the size of the intersection divided by the size of the union of the sample sets.

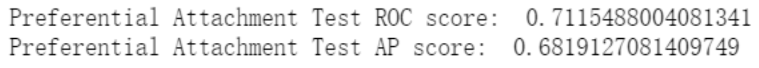
Score methods: ROC and AP



```
Jaccard Coefficient Test ROC score:  0.5012551816180292
Jaccard Coefficient Test AP score:  0.5002813093393331
```

Preferential Attachment (unsupervised)
A preferential attachment process is a class of processes in which some quantity is distributed among a few objects according to how much they already have.

Score methods: ROC and AP

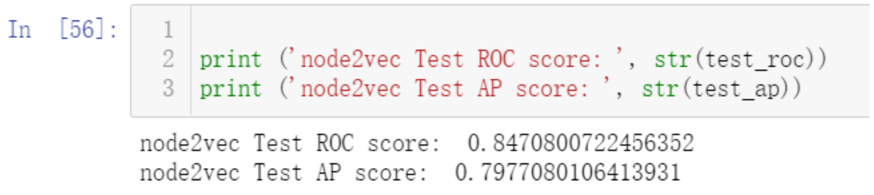


```
Preferential Attachment Test ROC score:  0.7115488004081341
Preferential Attachment Test AP score:  0.6819127081409749
```

By the score methods, I found that Preferential Attachment is the best for my link prediction method in unsupervised learning methods.

For supervised learning methods, I'm choosing to use different supervised learning models to do the link prediction. The result is below:

Logistic regression:

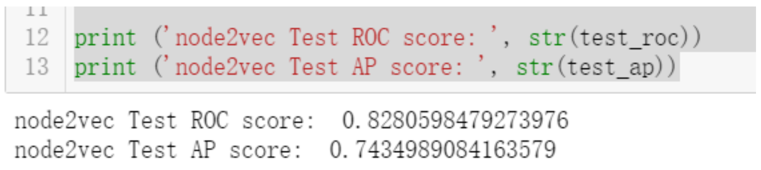


```
In [56]:

print ('node2vec Test ROC score: ', str(test_roc))
print ('node2vec Test AP score: ', str(test_ap))
```

```
node2vec Test ROC score:  0.8470800722456352
node2vec Test AP score:  0.7977080106413931
```

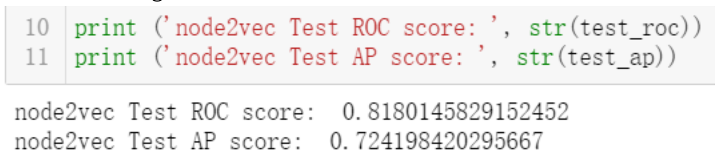
XGboost Classifier:



```
print ('node2vec Test ROC score: ', str(test_roc))
print ('node2vec Test AP score: ', str(test_ap))
```

```
node2vec Test ROC score:  0.8280598479273976
node2vec Test AP score:  0.7434989084163579
```

K nearest neighbour:



```
print ('node2vec Test ROC score: ', str(test_roc))
print ('node2vec Test AP score: ', str(test_ap))
```

```
node2vec Test ROC score:  0.8180145829152452
node2vec Test AP score:  0.724198420295667
```

I found that Logistic regression did the best for the default setting with both score methods in the supervised learning models.

Among all methods, Logistic Regression did the best, which has ROC score of 0.847, AP score of 0.798.