

```
#Importing Libraries
from google.colab import drive
drive.mount('/drive', force_remount=True);

import warnings
warnings.filterwarnings("ignore")

import csv
import pandas as pd#pandas to create small dataframes
import datetime #Convert to unix time
import time #Convert to unix time
# if numpy is not installed already : pip3 install numpy
import numpy as np#Do arithmetic operations on arrays
# matplotlib: used to plot graphs
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns#Plots
from matplotlib import rcParams#Size of plots
from sklearn.cluster import MiniBatchKMeans, KMeans#Clustering
import math
import pickle
import os
# to install xgboost: pip3 install xgboost
import xgboost as xgb

import warnings
import networkx as nx
import pdb
import pickle

from scipy.sparse.linalg import svds, eigs
import gc

from pandas import HDFStore,DataFrame
from pandas import read_hdf
from scipy.sparse.linalg import svds, eigs
import gc
from tqdm import tqdm
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score

Mounted at /drive
```

```
#reading graph
if not os.path.isfile('/drive/My Drive/Colab Notebooks/Assignment/data/train_woh
    traincsv = pd.read_csv('/drive/My Drive/Colab Notebooks/Assignment/data/trai
    print(traincsv[traincsv.idsna().any(1)])
    print(traincsv.info())
    print("Number of duplicate entries: ",sum(traincsv.duplicated()))
    traincsv.to_csv('/drive/My Drive/Colab Notebooks/Assignment/data/train_wohea
    print("saved the graph into file")
else:
    g=nx.read_edgelist('/drive/My Drive/Colab Notebooks/Assignment/data/train_wo
    print(nx.info(g))
```

DiGraph with 1862220 nodes and 9437519 edges

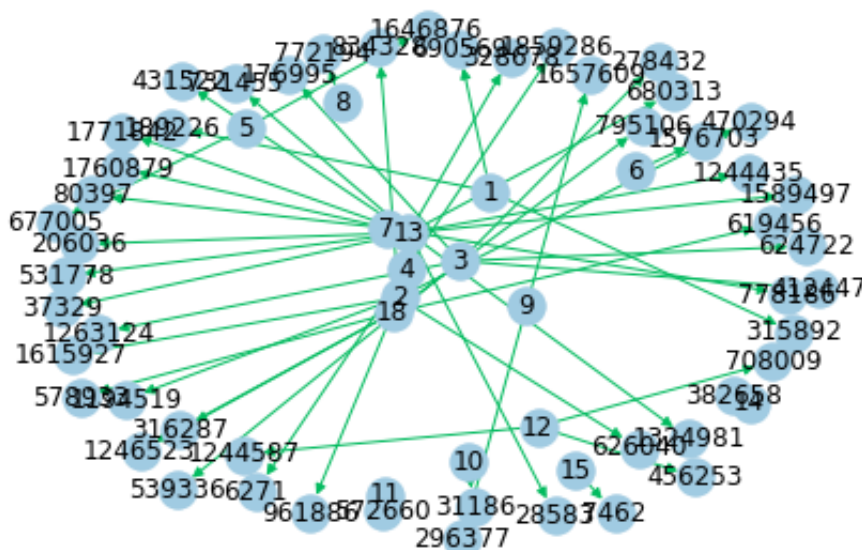
## ▼ Displaying a sub graph

```
if not os.path.isfile('train_woheader_sample.csv'):
    pd.read_csv('/drive/My Drive/Colab Notebooks/Assignment/data/train.csv', nro

subgraph=nx.read_edgelist('train_woheader_sample.csv',delimiter=',',create_using

pos=nx.spring_layout(subgraph)
nx.draw(subgraph,pos,node_color='#A0CBE2',edge_color='#00bb5e',width=1,edge_cmap
plt.savefig("graph_sample.pdf")
print(nx.info(subgraph))
```

DiGraph with 66 nodes and 50 edges



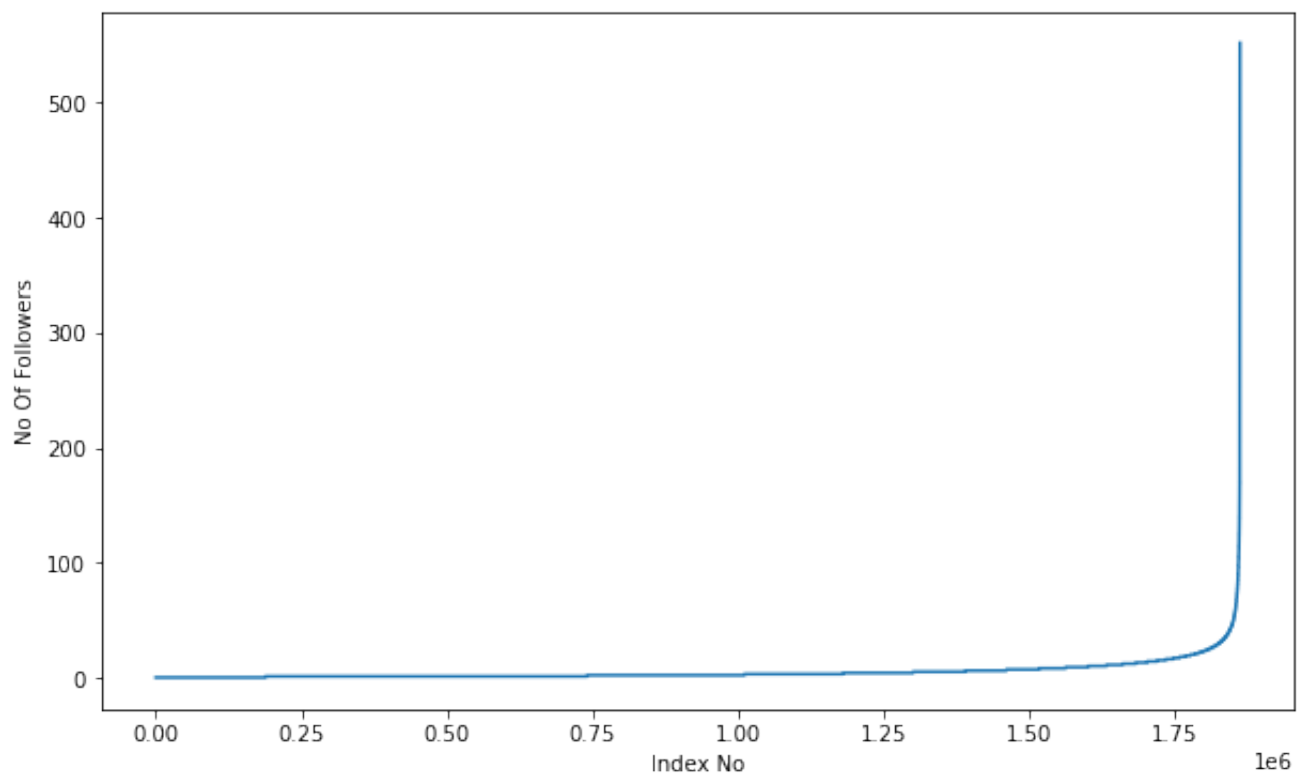
## ▼ 1. Exploratory Data Analysis

```
# No of Unique persons  
print("The number of unique persons",len(g.nodes()))
```

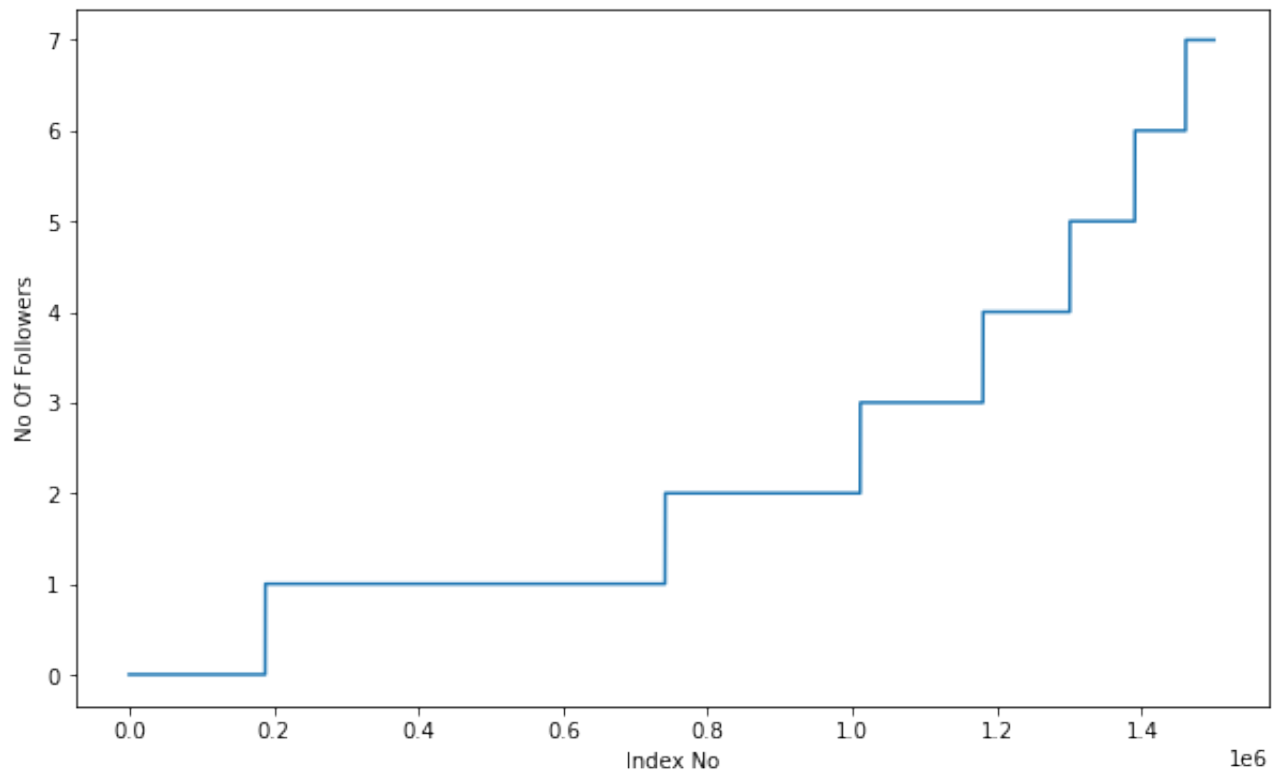
The number of unique persons 1862220

### ▼ 1.1 No of followers for each person

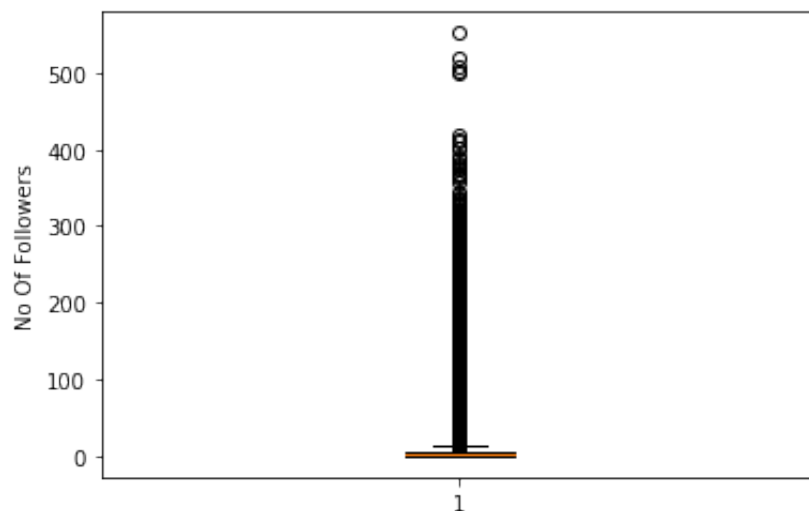
```
indegree_dist = list(dict(g.in_degree()).values())  
indegree_dist.sort()  
plt.figure(figsize=(10,6))  
plt.plot(indegree_dist)  
plt.xlabel('Index No')  
plt.ylabel('No Of Followers')  
plt.show()
```



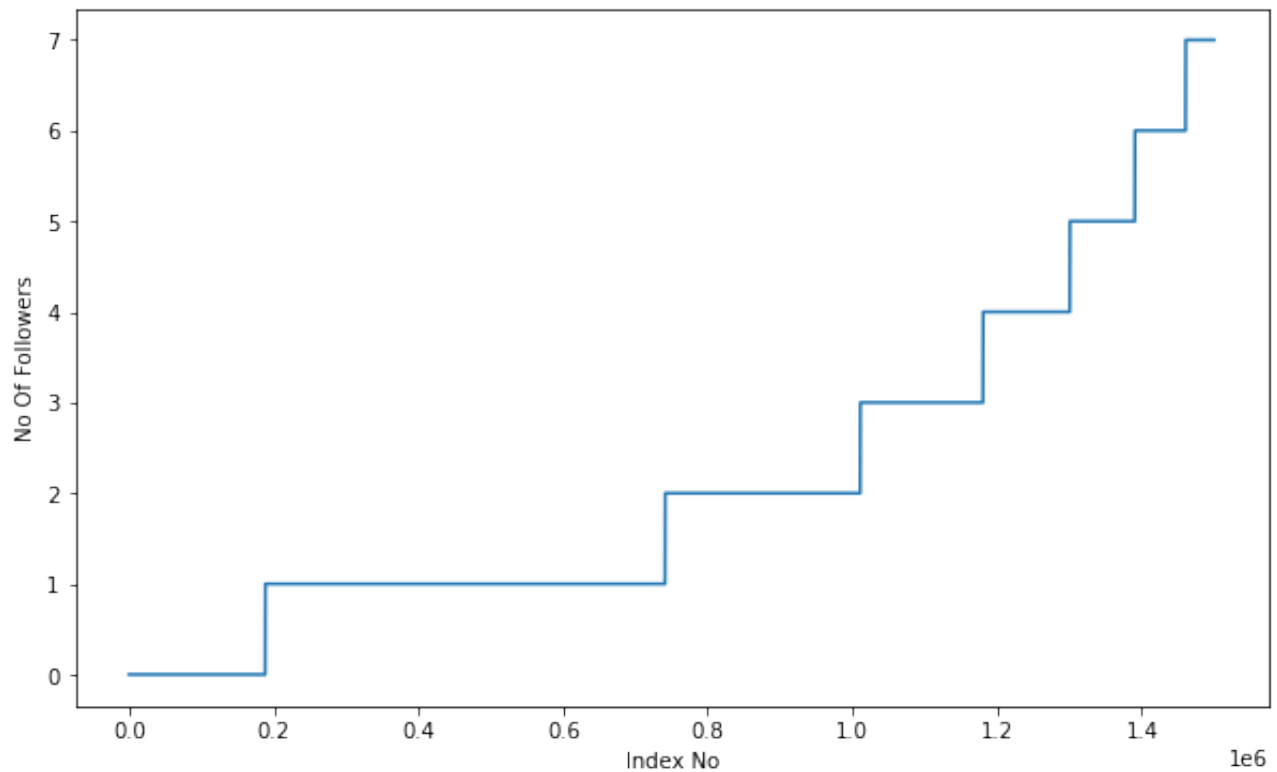
```
indegree_dist = list(dict(g.in_degree()).values())
indegree_dist.sort()
plt.figure(figsize=(10,6))
plt.plot(indegree_dist[0:1500000])
plt.xlabel('Index No')
plt.ylabel('No Of Followers')
plt.show()
```



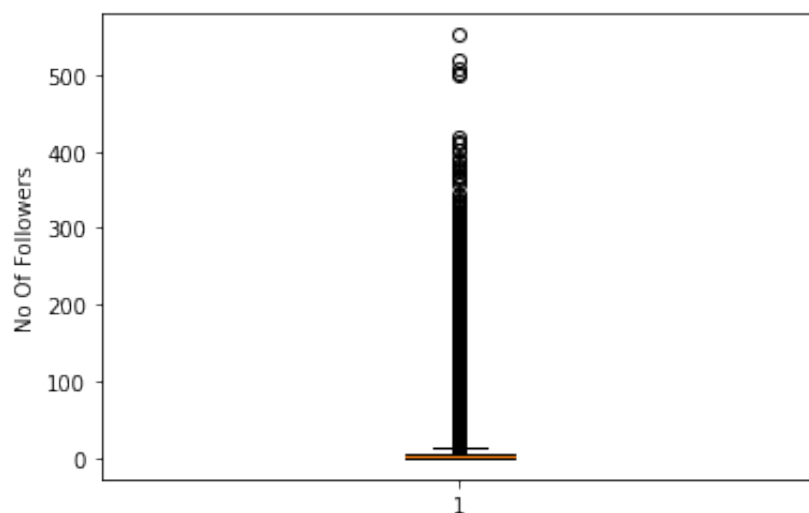
```
plt.boxplot(indegree_dist)
plt.ylabel('No Of Followers')
plt.show()
```



```
indegree_dist = list(dict(g.in_degree()).values())
indegree_dist.sort()
plt.figure(figsize=(10,6))
plt.plot(indegree_dist[0:1500000])
plt.xlabel('Index No')
plt.ylabel('No Of Followers')
plt.show()
```



```
plt.boxplot(indegree_dist)
plt.ylabel('No Of Followers')
plt.show()
```



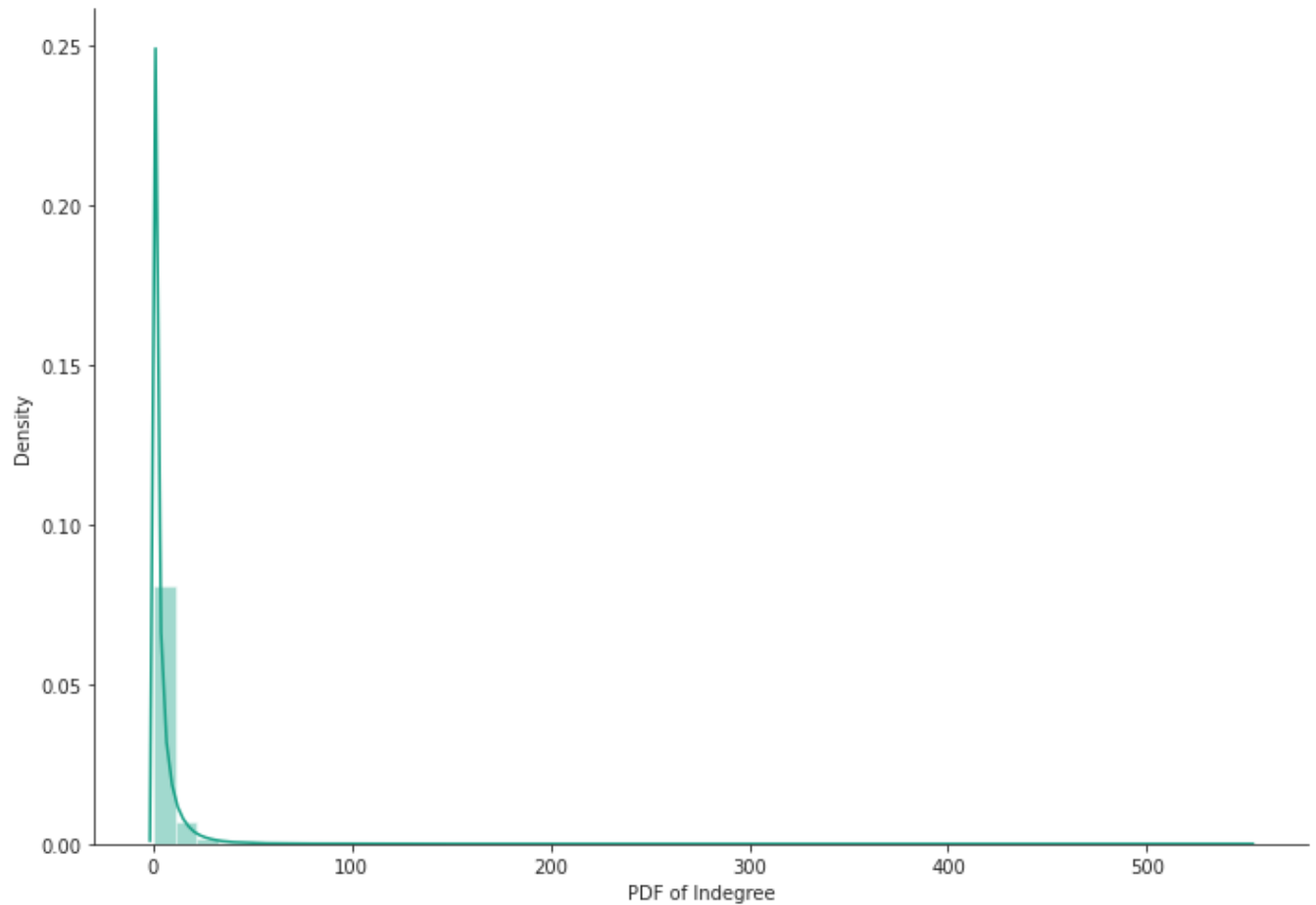
```
### 90-100 percentile
for i in range(0,11):
    print(90+i,'percentile value is',np.percentile(indegree_dist,90+i))

    90 percentile value is 12.0
    91 percentile value is 13.0
    92 percentile value is 14.0
    93 percentile value is 15.0
    94 percentile value is 17.0
    95 percentile value is 19.0
    96 percentile value is 21.0
    97 percentile value is 24.0
    98 percentile value is 29.0
    99 percentile value is 40.0
    100 percentile value is 552.0

### 99-100 percentile
for i in range(10,110,10):
    print(99+(i/100),'percentile value is',np.percentile(indegree_dist,99+(i/100))

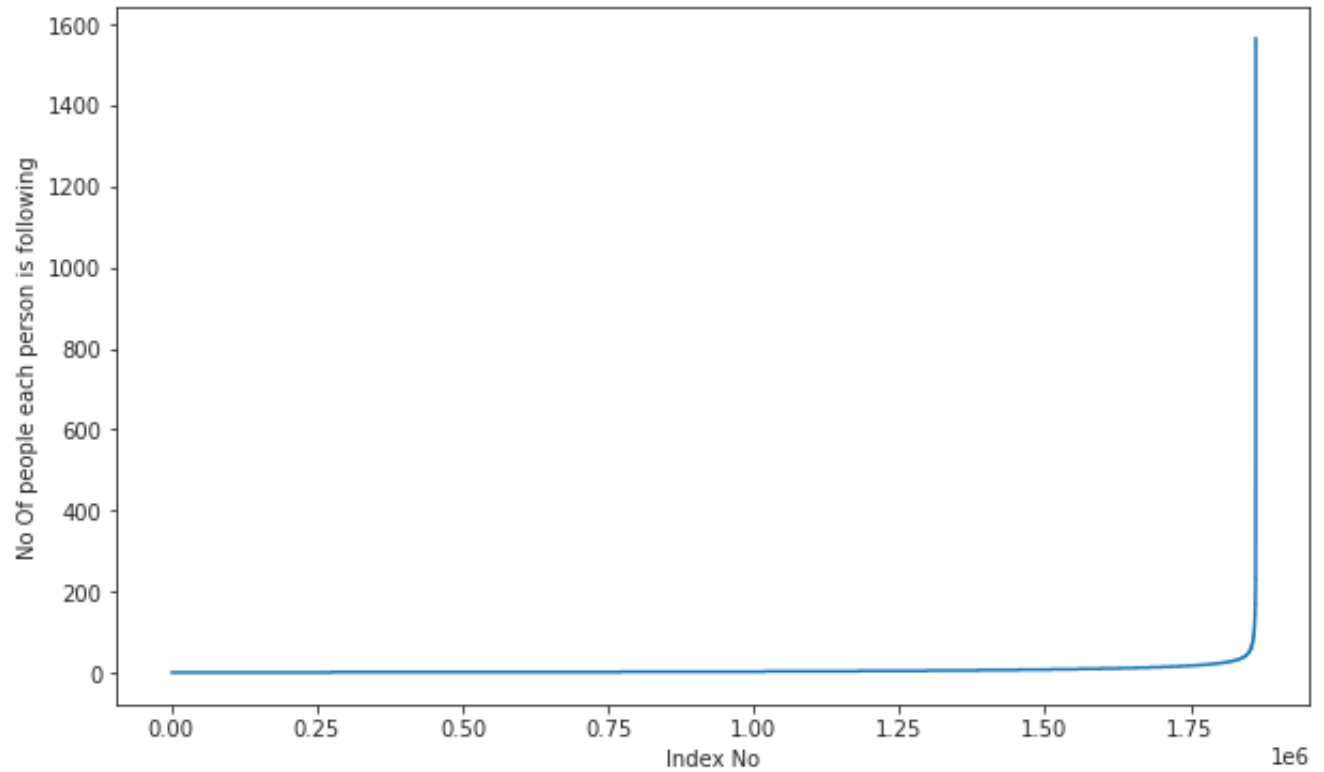
    99.1 percentile value is 42.0
    99.2 percentile value is 44.0
    99.3 percentile value is 47.0
    99.4 percentile value is 50.0
    99.5 percentile value is 55.0
    99.6 percentile value is 61.0
    99.7 percentile value is 70.0
    99.8 percentile value is 84.0
    99.9 percentile value is 112.0
    100.0 percentile value is 552.0
```

```
%matplotlib inline
sns.set_style('ticks')
fig, ax = plt.subplots()
fig.set_size_inches(11.7, 8.27)
sns.distplot(indegree_dist, color='#16A085')
plt.xlabel('PDF of Indegree')
sns.despine()
#plt.show()
```



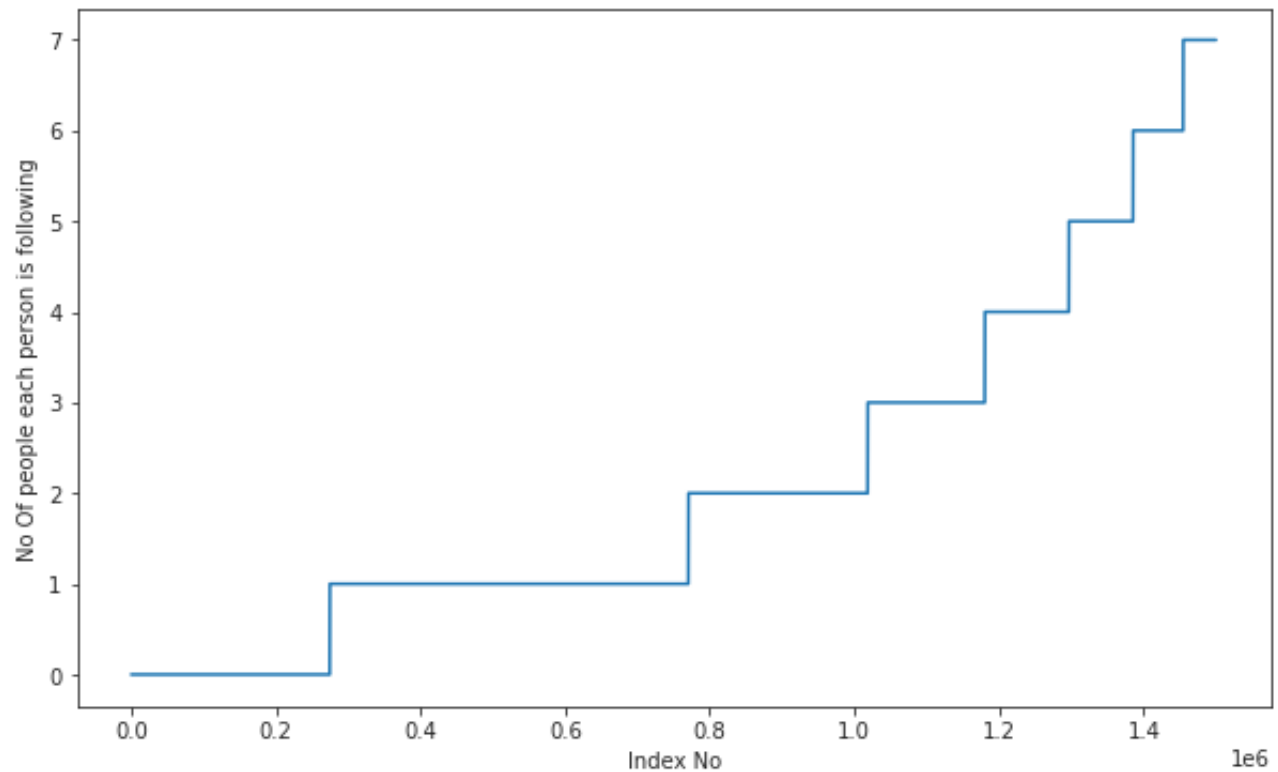
## ▼ 1.2 No of people each person is following

```
outdegree_dist = list(dict(g.out_degree()).values())
outdegree_dist.sort()
plt.figure(figsize=(10,6))
plt.plot(outdegree_dist)
plt.xlabel('Index No')
plt.ylabel('No Of people each person is following')
plt.show()
```

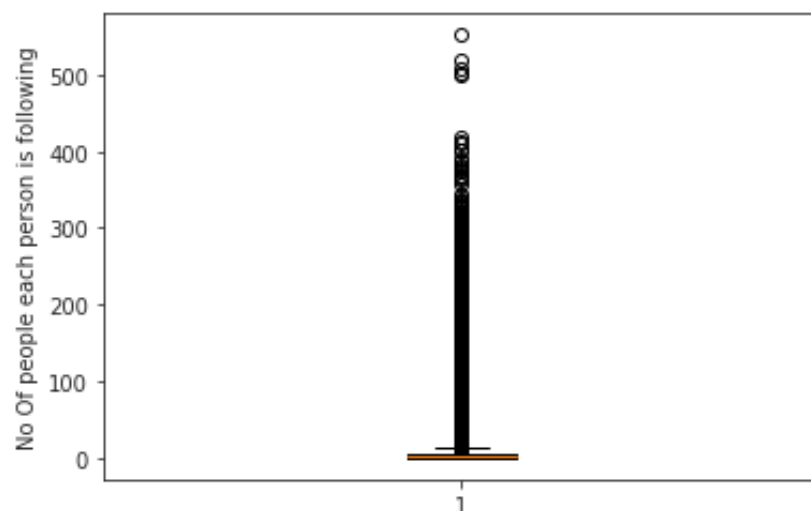




```
indegree_dist = list(dict(g.in_degree()).values())
indegree_dist.sort()
plt.figure(figsize=(10,6))
plt.plot(outdegree_dist[0:1500000])
plt.xlabel('Index No')
plt.ylabel('No Of people each person is following')
plt.show()
```



```
plt.boxplot(indegree_dist)
plt.ylabel('No Of people each person is following')
plt.show()
```



```
### 90-100 percentile
for i in range(0,11):
    print(90+i,'percentile value is',np.percentile(outdegree_dist,90+i))

    90 percentile value is 12.0
    91 percentile value is 13.0
    92 percentile value is 14.0
    93 percentile value is 15.0
    94 percentile value is 17.0
    95 percentile value is 19.0
    96 percentile value is 21.0
    97 percentile value is 24.0
    98 percentile value is 29.0
    99 percentile value is 40.0
    100 percentile value is 1566.0

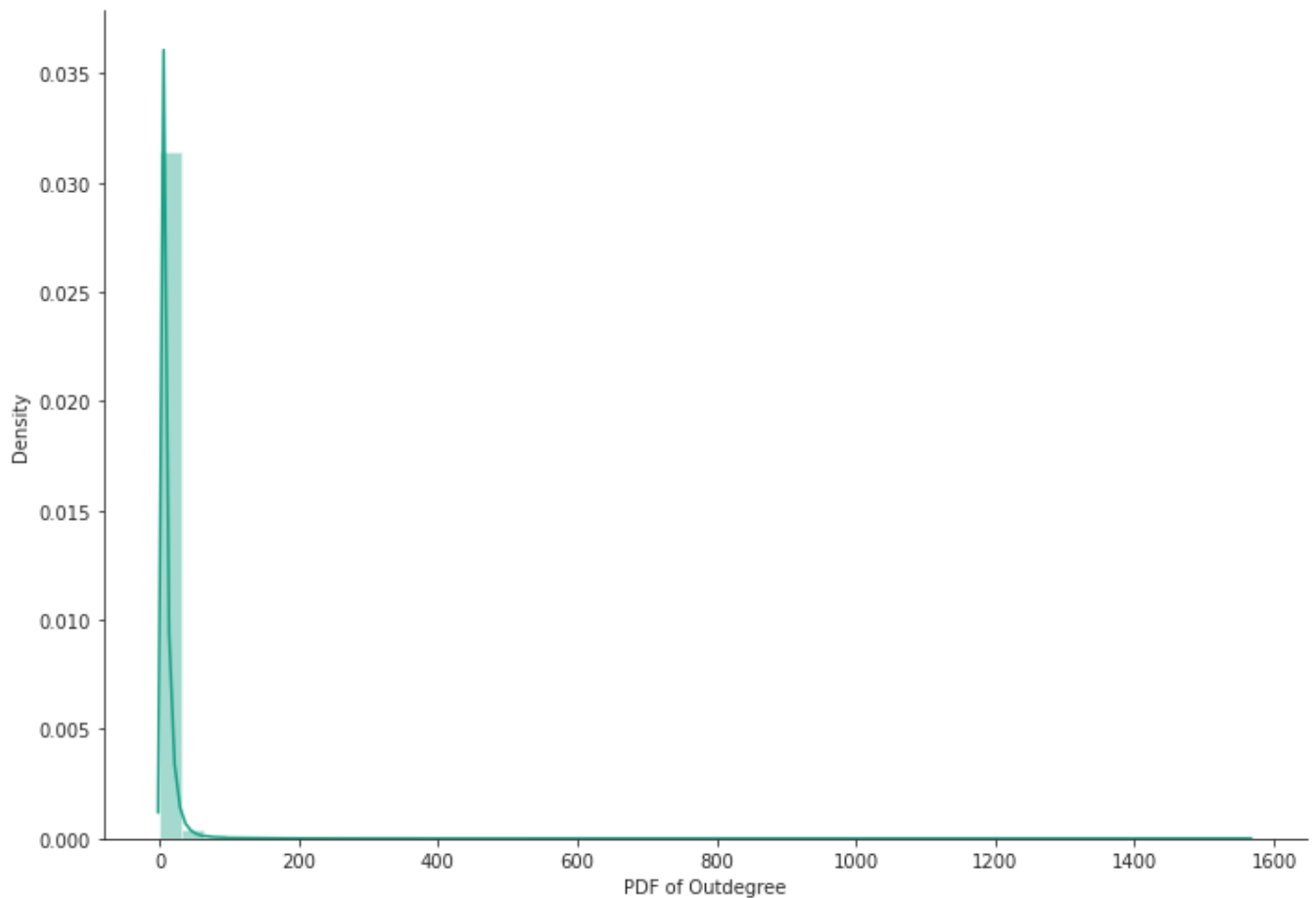
### 99-100 percentile
for i in range(10,110,10):
    print(99+(i/100),'percentile value is',np.percentile(outdegree_dist,99+(i/100)))

    99.1 percentile value is 42.0
    99.2 percentile value is 45.0
    99.3 percentile value is 48.0
    99.4 percentile value is 52.0
    99.5 percentile value is 56.0
    99.6 percentile value is 63.0
    99.7 percentile value is 73.0
    99.8 percentile value is 90.0
    99.9 percentile value is 123.0
    100.0 percentile value is 1566.0
```

```

sns.set_style('ticks')
fig, ax = plt.subplots()
fig.set_size_inches(11.7, 8.27)
sns.distplot(outdegree_dist, color='#19A085')
plt.xlabel('PDF of Outdegree')
sns.despine()

```



```

print('No of persons those are not following anyone are' ,sum(np.array(outdegree
sum(np.array(outdegree_dist)==0)*100/len(outdegr

```

No of persons those are not following anyone are 274512 and % is 14.7411154

```

print('No of persons having zero followers are' ,sum(np.array(indegree_dist)==0)
sum(np.array(indegree_dist)==0)*100/len(indegree

```

No of persons having zero followers are 188043 and % is 10.097786512871734

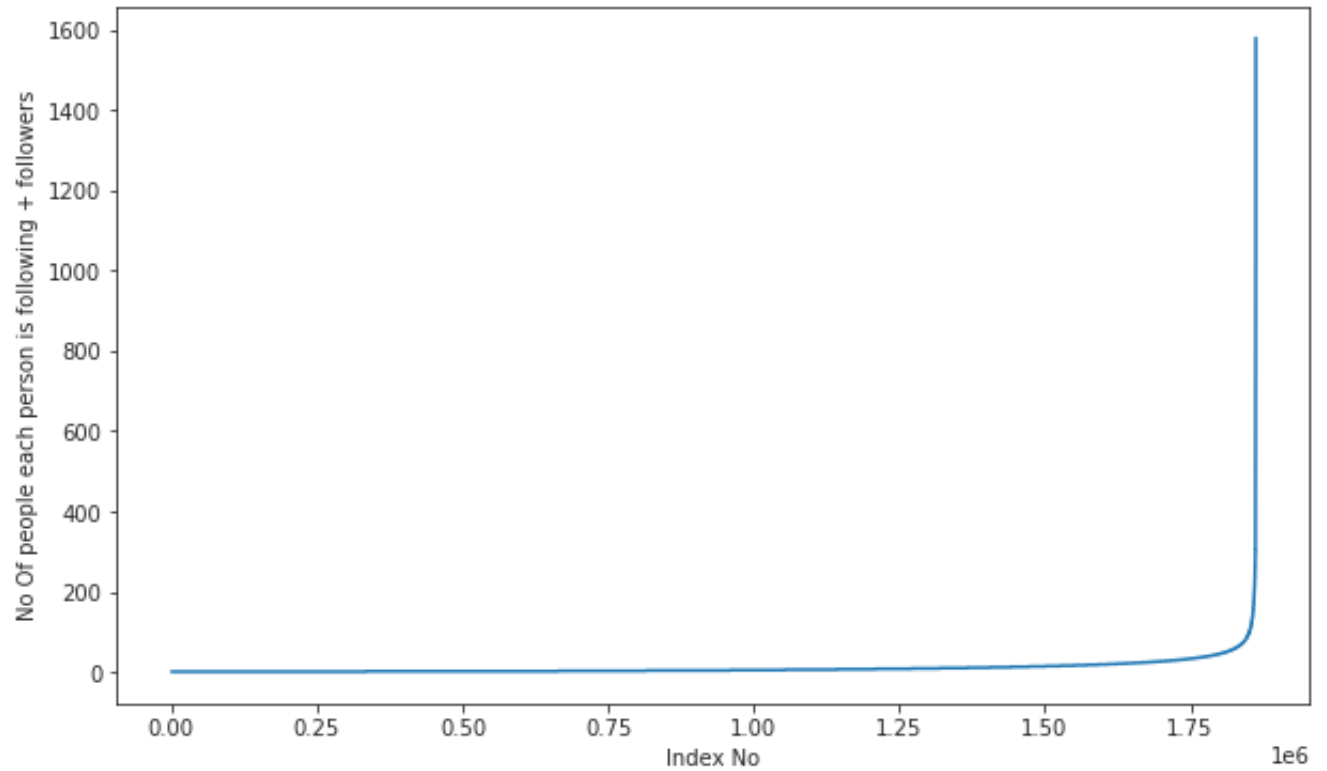
```
count=0
for i in g.nodes():
    if len(list(g.predecessors(i)))==0 :
        if len(list(g.successors(i)))==0:
            count+=1
print('No of persons those are not not following anyone and also not having any

    No of persons those are not not following anyone and also not having any fo
```

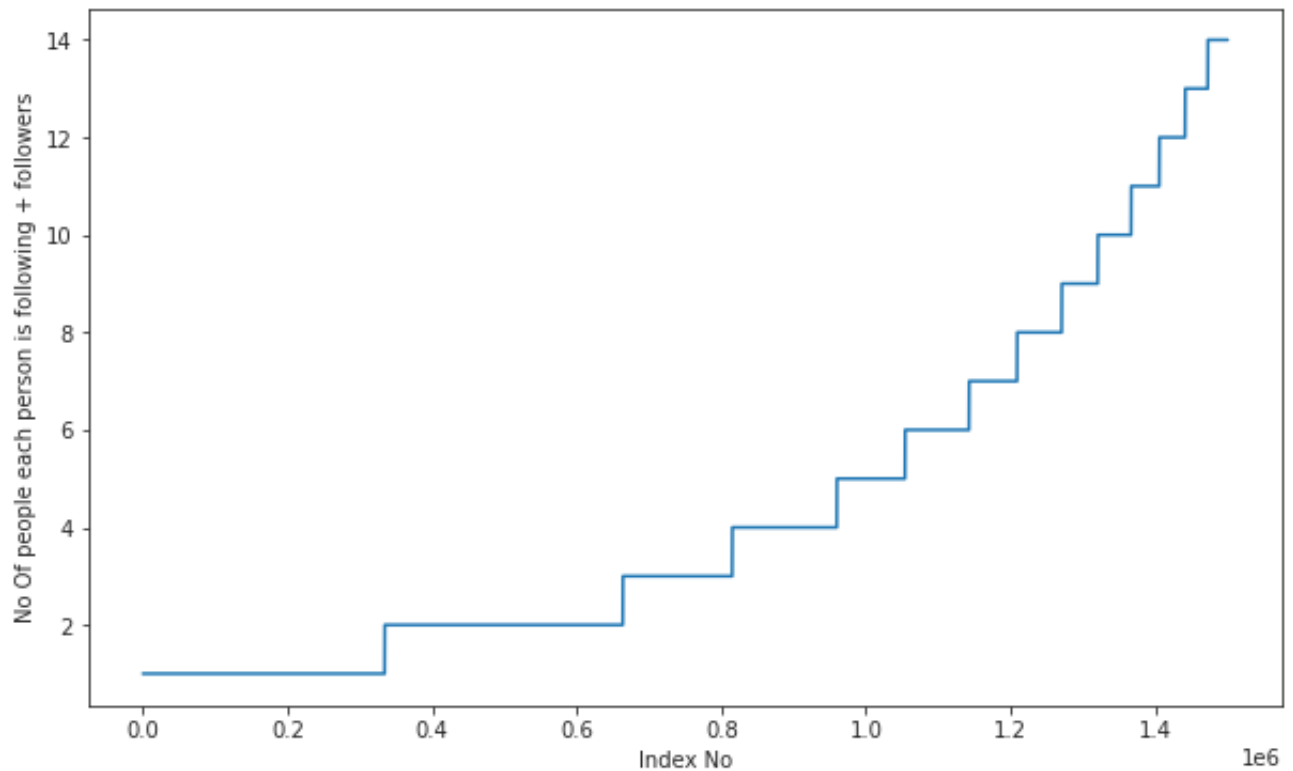
### ▼ 1.3 both followers + following

```
from collections import Counter
dict_in = dict(g.in_degree())
dict_out = dict(g.out_degree())
d = Counter(dict_in) + Counter(dict_out)
in_out_degree = np.array(list(d.values()))
```

```
in_out_degree_sort = sorted(in_out_degree)
plt.figure(figsize=(10,6))
plt.plot(in_out_degree_sort)
plt.xlabel('Index No')
plt.ylabel('No Of people each person is following + followers')
plt.show()
```



```
in_out_degree_sort = sorted(in_out_degree)
plt.figure(figsize=(10,6))
plt.plot(in_out_degree_sort[0:1500000])
plt.xlabel('Index No')
plt.ylabel('No Of people each person is following + followers')
plt.show()
```



```
### 90-100 percentile
for i in range(0,11):
    print(90+i,'percentile value is',np.percentile(in_out_degree_sort,90+i))

90 percentile value is 24.0
91 percentile value is 26.0
92 percentile value is 28.0
93 percentile value is 31.0
94 percentile value is 33.0
95 percentile value is 37.0
96 percentile value is 41.0
97 percentile value is 48.0
98 percentile value is 58.0
99 percentile value is 79.0
100 percentile value is 1579.0
```

```

### 99-100 percentile
for i in range(10,110,10):
    print(99+(i/100),'percentile value is',np.percentile(in_out_degree_sort,99+(

99.1 percentile value is 83.0
99.2 percentile value is 87.0
99.3 percentile value is 93.0
99.4 percentile value is 99.0
99.5 percentile value is 108.0
99.6 percentile value is 120.0
99.7 percentile value is 138.0
99.8 percentile value is 168.0
99.9 percentile value is 221.0
100.0 percentile value is 1579.0

print('Min of no of followers + following is',in_out_degree.min())
print(np.sum(in_out_degree==in_out_degree.min()),' persons having minimum no of

Min of no of followers + following is 1
334291 persons having minimum no of followers + following

print('Max of no of followers + following is',in_out_degree.max())
print(np.sum(in_out_degree==in_out_degree.max()),' persons having maximum no of

Max of no of followers + following is 1579
1 persons having maximum no of followers + following

print('No of persons having followers + following less than 10 are',np.sum(in_ou

No of persons having followers + following less than 10 are 1320326

print('No of weakly connected components',len(list(nx.weakly_connected_component
count=0
for i in list(nx.weakly_connected_components(g)):
    if len(i)==2:
        count+=1
print('weakly connected components wit 2 nodes',count)

No of weakly connected components 45558
weakly connected components wit 2 nodes 32195

```

## ▼ 2. Posing a problem as classification problem

## 2.1 Generating some edges which are not present in graph for supervised learning

Generated Bad links from graph which are not in graph and whose shortest path is greater than 2.

```
%%time
###generating bad edges from given graph
import random
if not os.path.isfile('/drive/My Drive/Colab Notebooks/Assignment/data/after_ed
#getting all set of edges
r = csv.reader(open('/drive/My Drive/Colab Notebooks/Assignment/data/after_e
edges = dict()
for edge in r:
    edges[(edge[0], edge[1])] = 1

missing_edges = set([])
while (len(missing_edges)<9437519):
    a=random.randint(1, 1862220)
    b=random.randint(1, 1862220)
    tmp = edges.get((a,b),-1)
    if tmp == -1 and a!=b:
        try:
            if nx.shortest_path_length(g,source=a,target=b) > 2:

                missing_edges.add((a,b))
            else:
                continue
        except:
            missing_edges.add((a,b))
    else:
        continue
pickle.dump(missing_edges,open('/drive/My Drive/Colab Notebooks/Assignment/d
else:
    missing_edges = pickle.load(open('/drive/My Drive/Colab Notebooks/Assignment

CPU times: user 3.21 s, sys: 1.93 s, total: 5.14 s
Wall time: 5.19 s

len(missing_edges)

9437519
```



## ▼ 2.2 Training and Test data split:

Removed edges from Graph and used as test data and after removing used that graph for creating features for Train and test data

```
from sklearn.model_selection import train_test_split
if (not os.path.isfile('/drive/My Drive/Colab Notebooks/Assignment/data/after_ed
#reading total data df
df_pos = pd.read_csv('/drive/My Drive/Colab Notebooks/Assignment/data/train.
df_neg = pd.DataFrame(list(missing_edges), columns=['source_node', 'destinat

print("Number of nodes in the graph with edges", df_pos.shape[0])
print("Number of nodes in the graph without edges", df_neg.shape[0])

#Trian test split
#Spiltted data into 80-20
#positive links and negative links seperatly because we need positive traini
#and for feature generation
X_train_pos, X_test_pos, y_train_pos, y_test_pos = train_test_split(df_pos,
X_train_neg, X_test_neg, y_train_neg, y_test_neg = train_test_split(df_neg,

print('='*60)
print("Number of nodes in the train data graph with edges", X_train_pos.shap
print("Number of nodes in the train data graph without edges", X_train_neg.s
print('='*60)
print("Number of nodes in the test data graph with edges", X_test_pos.shape[
print("Number of nodes in the test data graph without edges", X_test_neg.sha

#removing header and saving
X_train_pos.to_csv('/drive/My Drive/Colab Notebooks/Assignment/data/after_ed
X_test_pos.to_csv('/drive/My Drive/Colab Notebooks/Assignment/data/after_eda
X_train_neg.to_csv('/drive/My Drive/Colab Notebooks/Assignment/data/after_ed
X_test_neg.to_csv('/drive/My Drive/Colab Notebooks/Assignment/data/after_eda
else:
#Graph from Traing data only
del missing_edges

Number of nodes in the graph with edges 9437519
Number of nodes in the graph without edges 9437519
=====
Number of nodes in the train data graph with edges 7550015 = 7550015
Number of nodes in the train data graph without edges 7550015 = 7550015
=====
Number of nodes in the test data graph with edges 1887504 = 1887504
Number of nodes in the test data graph without edges 1887504 = 1887504
```

```
if (os.path.isfile('/drive/My Drive/Colab Notebooks/Assignment/data/after_eda/tr
train_graph=nx.read_edgelist('/drive/My Drive/Colab Notebooks/Assignment/dat
test_graph=nx.read_edgelist('/drive/My Drive/Colab Notebooks/Assignment/data
print(nx.info(train_graph))
print(nx.info(test_graph))

# finding the unique nodes in the both train and test graphs
train_nodes_pos = set(train_graph.nodes())
test_nodes_pos = set(test_graph.nodes())

trY_teY = len(train_nodes_pos.intersection(test_nodes_pos))
trY_teN = len(train_nodes_pos - test_nodes_pos)
teY_trN = len(test_nodes_pos - train_nodes_pos)

print('no of people common in train and test -- ',trY_teY)
print('no of people present in train but not present in test -- ',trY_teN)

print('no of people present in test but not present in train -- ',teY_trN)
print(' % of people not there in Train but exist in Test in total Test data

DiGraph with 1780722 nodes and 7550015 edges
DiGraph with 1144623 nodes and 1887504 edges
no of people common in train and test -- 1063125
no of people present in train but not present in test -- 717597
no of people present in test but not present in train -- 81498
% of people not there in Train but exist in Test in total Test data are 7.
```

we have a cold start problem here

```
#final train and test data sets
```

```
X_train_pos = pd.read_csv('/drive/My Drive/Colab Notebooks/Assignment/data/after_eda/train_pos.csv')
X_test_pos = pd.read_csv('/drive/My Drive/Colab Notebooks/Assignment/data/after_eda/test_pos.csv')
X_train_neg = pd.read_csv('/drive/My Drive/Colab Notebooks/Assignment/data/after_eda/train_neg.csv')
X_test_neg = pd.read_csv('/drive/My Drive/Colab Notebooks/Assignment/data/after_eda/test_neg.csv')

y_train_pos = np.ones(len(X_train_pos))
y_train_neg = np.zeros(len(X_train_neg))
y_test_pos = np.ones(len(X_test_pos))
y_test_neg = np.zeros(len(X_test_neg))
print('='*60)
print("Number of nodes in the train data graph with edges", X_train_pos.shape[0])
print("Number of nodes in the train data graph without edges", X_train_neg.shape[0])
print('='*60)
print("Number of nodes in the test data graph with edges", X_test_pos.shape[0])
print("Number of nodes in the test data graph without edges", X_test_neg.shape[0])

X_train = X_train_pos.append(X_train_neg, ignore_index=True)
y_train = np.concatenate((y_train_pos, y_train_neg))
X_test = X_test_pos.append(X_test_neg, ignore_index=True)
y_test = np.concatenate((y_test_pos, y_test_neg))

X_train.to_csv('/drive/My Drive/Colab Notebooks/Assignment/data/after_eda/train_data.csv')
X_test.to_csv('/drive/My Drive/Colab Notebooks/Assignment/data/after_eda/test_data.csv')
pd.DataFrame(y_train.astype(int)).to_csv('/drive/My Drive/Colab Notebooks/Assignment/data/after_eda/train_target.csv')
pd.DataFrame(y_test.astype(int)).to_csv('/drive/My Drive/Colab Notebooks/Assignment/data/after_eda/test_target.csv')

=====
Number of nodes in the train data graph with edges 7550015
Number of nodes in the train data graph without edges 7550015
=====
Number of nodes in the test data graph with edges 1887504
Number of nodes in the test data graph without edges 1887504

print("Data points in train data", X_train.shape)
print("Data points in test data", X_test.shape)
print("Shape of target variable in train", y_train.shape)
print("Shape of target variable in test", y_test.shape)

Data points in train data (15100030, 2)
Data points in test data (3775008, 2)
Shape of target variable in train (15100030,)
Shape of target variable in test (3775008,)
```

```

import networkx as nx
import os

if os.path.isfile('/drive/My Drive/Colab Notebooks/Assignment/data/after_eda/train_graph'):
    train_graph=nx.read_edgelist('/drive/My Drive/Colab Notebooks/Assignment/data/train_graph.edgelist')
    print(nx.info(train_graph))
else:
    print("please run the FB_EDA.ipynb or download the files from drive")

    DiGraph with 1780722 nodes and 7550015 edges

```

## ▼ 2. Similarity measures

### ▼ 2.1 Jaccard Distance:

<http://www.statisticshowto.com/jaccard-index/>

$$j = \frac{|X \cap Y|}{|X \cup Y|}$$

```

#for followees
def jaccard_for_followees(a,b):
    try:
        if len(set(train_graph.successors(a))) == 0 | len(set(train_graph.successors(b))) == 0:
            return 0
        sim = (len(set(train_graph.successors(a)).intersection(set(train_graph.successors(b)))) /
                (len(set(train_graph.successors(a)).union(set(train_graph.successors(b))))))
    except:
        return 0
    return sim

#one test case
print(jaccard_for_followees(273084,1505602))

0.0

```

```

#node 1635354 not in graph
print(jaccard_for_followees(273084,1505602))

0.0

#for followers
def jaccard_for_followers(a,b):
    try:
        if len(set(train_graph.predecessors(a))) == 0 | len(set(g.predecessors(
            return 0
        sim = (len(set(train_graph.predecessors(a)).intersection(set(train_graph
            (len(set(train_graph.predecessors(a)).union(set
        return sim
    except:
        return 0

print(jaccard_for_followers(273084,470294))

0

#node 1635354 not in graph
print(jaccard_for_followees(669354,1635354))

0

```

## ▼ 2.2 Cosine distance

$$\text{CosineDistance} = \frac{|X \cap Y|}{\sqrt{|X| \cdot |Y|}}$$

```

#for followees
def cosine_for_followees(a,b):
    try:
        if len(set(train_graph.successors(a))) == 0 | len(set(train_graph.succe
            return 0
        sim = (len(set(train_graph.successors(a)).intersection(set(train_graph.s
            (math.sqrt(len(set(train_graph.successors(a)
        return sim
    except:
        return 0

```

```
print(cosine_for_followees(273084,1505602))
```

```
0
```

```
print(cosine_for_followees(273084,1635354))
```

```
0
```

```
def cosine_for_followers(a,b):
```

```
    try:
```

```
        if len(set(train_graph.predecessors(a))) == 0 | len(set(train_graph.predecessors(b))) == 0:
```

```
            return 0
```

```
        sim = (len(set(train_graph.predecessors(a)).intersection(set(train_graph.predecessors(b)))) /
```

```
                (math.sqrt(len(set(train_graph.predecessors(a)))) * math.sqrt(len(set(train_graph.predecessors(b))))))
```

```
        return sim
```

```
    except:
```

```
        return 0
```

```
print(cosine_for_followers(2,470294))
```

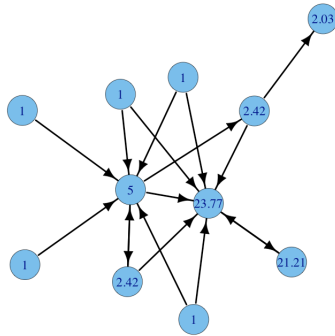
```
0
```

```
print(cosine_for_followers(669354,1635354))
```

```
0
```

### ▼ 3. Ranking Measures

PageRank computes a ranking of the nodes in the graph  $G$  based on the structure of the incoming links.



Mathematical PageRanks for a simple network, expressed as percentages. (Google uses a logarithmic scale.) Page C has a higher PageRank than Page E, even though there are fewer links to C; the one link to C comes from an important page and hence is of high value. If web surfers who start on a random page have an 85% likelihood of choosing a random link from the page they are currently visiting, and a 15% likelihood of jumping to a page chosen at random from the entire web, they will reach Page E 8.1% of the time. **(The 15% likelihood of jumping to an arbitrary page corresponds to a damping factor of 85%.) Without damping, all web surfers would eventually end up on Pages A, B, or C, and all other pages would have PageRank zero. In the presence of damping, Page A effectively links to all pages in the web, even though it has no outgoing links of its own.**

### ▼ 3.1 Page Ranking

<https://en.wikipedia.org/wiki/PageRank>

```
import pickle
import os
```

```

if not os.path.isfile('/drive/My Drive/Colab Notebooks/Assignment/data/fea_sampl
    pr = nx.pagerank(train_graph, alpha=0.85)
    pickle.dump(pr,open('/drive/My Drive/Colab Notebooks/Assignment/data/fea_sam
else:
    pr = pickle.load(open('/drive/My Drive/Colab Notebooks/Assignment/data/fea_s

print('min',pr[min(pr, key=pr.get)])
print('max',pr[max(pr, key=pr.get)])
print('mean',float(sum(pr.values())) / len(pr))

min 1.6556497245737814e-07
max 2.709825134193587e-05
mean 5.615699699389075e-07

#for imputing to nodes which are not there in Train data
mean_pr = float(sum(pr.values())) / len(pr)
print(mean_pr)

```

```
5.615699699389075e-07
```

## ▼ 4. Other Graph Features

getting shortest path between two nodes , if nodes have direct path i.e directly connected then we are removing that edge and calculate path.

```

#if has direct edge then deleting that edge and calculating shortest path
def compute_shortest_path_length(a,b):
    p=-1
    try:
        if train_graph.has_edge(a,b):
            train_graph.remove_edge(a,b)
            p= nx.shortest_path_length(train_graph,source=a,target=b)
            train_graph.add_edge(a,b)
        else:
            p= nx.shortest_path_length(train_graph,source=a,target=b)
        return p
    except:
        return -1

```



```
#testing
compute_shortest_path_length(77697, 826021)

10

#testing
compute_shortest_path_length(669354,1635354)

-1

#getting weekly connected edges from graph
wcc=list(nx.weakly_connected_components(train_graph))
def belongs_to_same_wcc(a,b):
    index = []
    if train_graph.has_edge(b,a):
        return 1
    if train_graph.has_edge(a,b):
        for i in wcc:
            if a in i:
                index= i
                break
        if (b in index):
            train_graph.remove_edge(a,b)
            if compute_shortest_path_length(a,b)==-1:
                train_graph.add_edge(a,b)
                return 0
            else:
                train_graph.add_edge(a,b)
                return 1
        else:
            return 0
    else:
        for i in wcc:
            if a in i:
                index= i
                break
        if(b in index):
            return 1
        else:
            return 0

belongs_to_same_wcc(861, 1659750)

0
```

```
belongs_to_same_wcc(669354, 1635354)
```

```
0
```

### ▼ 4.3 Adamic/Adar Index:

Adamic/Adar measures is defined as inverted sum of degrees of common neighbours for given two vertices.

$$A(x, y) = \sum_{u \in N(x) \cap N(y)} \frac{1}{\log(|N(u)|)}$$

```
#adar index
def calc_adar_in(a,b):
    sum=0
    try:
        n=list(set(train_graph.successors(a)).intersection(set(train_graph.succe
        if len(n)!=0:
            for i in n:
                sum=sum+(1/np.log10(len(list(train_graph.predecessors(i)))))
            return sum
        else:
            return 0
    except:
        return 0
```

```
calc_adar_in(1,189226)
```

```
0
```

```
calc_adar_in(669354,1635354)
```

```
0
```

### ▼ 4.4 Is person was following back:

```
def follows_back(a,b):
    if train_graph.has_edge(b,a):
        return 1
    else:
        return 0
```

```
follows_back(1,189226)
```

```
1
```

```
follows_back(669354,1635354)
```

```
0
```

## ▼ 4.5 Katz Centrality:

[https://en.wikipedia.org/wiki/Katz\\_centrality](https://en.wikipedia.org/wiki/Katz_centrality)

<https://www.geeksforgeeks.org/katz-centrality-centrality-measure/> Katz centrality computes the centrality for a node based on the centrality of its neighbors. It is a generalization of the eigenvector centrality. The Katz centrality for node  $i$  is

$$x_i = \alpha \sum_j A_{ij} x_j + \beta,$$

where  $A$  is the adjacency matrix of the graph  $G$  with eigenvalues

$\lambda$

.

The parameter

$\beta$

controls the initial centrality and

$$\alpha < \frac{1}{\lambda_{max}}.$$

```
if not os.path.isfile('/drive/My Drive/Colab Notebooks/Assignment/data/fea_sampl
    katz = nx.katz.katz_centrality(train_graph,alpha=0.005,beta=1)
    pickle.dump(katz,open('/drive/My Drive/Colab Notebooks/Assignment/data/fea_s
else:
    katz = pickle.load(open('/drive/My Drive/Colab Notebooks/Assignment/data/fea
```

```
print('min',katz[min(katz, key=katz.get)])
print('max',katz[max(katz, key=katz.get)])
print('mean',float(sum(katz.values())) / len(katz))
```

```
min 0.0007313532484055976
max 0.0033945549816945084
mean 0.0007483800935553942
```

```
mean_katz = float(sum(katz.values())) / len(katz)
print(mean_katz)
```

```
0.0007483800935553942
```

## ▼ 4.6 Hits Score

The HITS algorithm computes two numbers for a node. Authorities estimates the node value based on the incoming links. Hubs estimates the node value based on outgoing links.

[https://en.wikipedia.org/wiki/HITS\\_algorithm](https://en.wikipedia.org/wiki/HITS_algorithm)

```
import os
```

```
if not os.path.isfile('/drive/My Drive/Colab Notebooks/Assignment/data/fea_samp1
    hits = nx.hits(train_graph, max_iter=100, tol=1e-08, nstart=None, normalized
    pickle.dump(hits,open('/drive/My Drive/Colab Notebooks/Assignment/data/fea_s
else:
```

```
    hits = pickle.load(open('/drive/My Drive/Colab Notebooks/Assignment/data/fea
```

```
print('min',hits[0][min(hits[0], key=hits[0].get)])
print('max',hits[0][max(hits[0], key=hits[0].get)])
print('mean',float(sum(hits[0].values())) / len(hits[0]))
```

```
min -3.533801190926718e-19
max 0.004868653379539048
mean 5.615699699308677e-07
```

## ▼ 5. Featurization

```

import random
if os.path.isfile('/drive/My Drive/Colab Notebooks/Assignment/data/after_eda/train_
    filename = "/drive/My Drive/Colab Notebooks/Assignment/data/after_eda/train_
    # you uncomment this line, if you dont know the lentgh of the file name
    # here we have hardcoded the number of lines as 15100030
    # n_train = sum(1 for line in open(filename)) #number of records in file (ex
    n_train = 15100028
    s = 100000 #desired sample size
    skip_train = sorted(random.sample(range(1,n_train+1),n_train-s))
    #https://stackoverflow.com/a/22259008/4084039

if os.path.isfile('/drive/My Drive/Colab Notebooks/Assignment/data/after_eda/test_a
    filename = "/drive/My Drive/Colab Notebooks/Assignment/data/after_eda/test_a
    # you uncomment this line, if you dont know the lentgh of the file name
    # here we have hardcoded the number of lines as 3775008
    # n_test = sum(1 for line in open(filename)) #number of records in file (exc
    n_test = 3775006
    s = 50000 #desired sample size
    skip_test = sorted(random.sample(range(1,n_test+1),n_test-s))
    #https://stackoverflow.com/a/22259008/4084039

print("Number of rows in the train data file:", n_train)
print("Number of rows we are going to elimiate in train data are",len(skip_train)
print("Number of rows in the test data file:", n_test)
print("Number of rows we are going to elimiate in test data are",len(skip_test))

    Number of rows in the train data file: 15100028
    Number of rows we are going to elimiate in train data are 15000028
    Number of rows in the test data file: 3775006
    Number of rows we are going to elimiate in test data are 3725006

import pandas as pd

```

```

df_final_train = pd.read_csv('/drive/My Drive/Colab Notebooks/Assignment/data/af
df_final_train['indicator_link'] = pd.read_csv('/drive/My Drive/Colab Notebooks/
print("Our train matrix size ",df_final_train.shape)
df_final_train.head(2)

```


Our train matrix size (100002, 3)

	<b>source_node</b>	<b>destination_node</b>	<b>indicator_link</b>
<b>0</b>	273084	1505602	1
<b>1</b>	1722833	544361	1



```
df_final_test = pd.read_csv('/drive/My Drive/Colab Notebooks/Assignment/data/aft
df_final_test['indicator_link'] = pd.read_csv('/drive/My Drive/Colab Notebooks/A
print("Our test matrix size ",df_final_test.shape)
df_final_test.head(2)
```

Our test matrix size (50002, 3)

	source_node	destination_node	indicator_link	
0	848424	784690	1	
1	875380	1394902	1	

## ▼ 5.2 Adding a set of features

**we will create these each of these features for both train and test data points**

1. jaccard\_followers
2. jaccard\_followees
3. cosine\_followers
4. cosine\_followees
5. num\_followers\_s
6. num\_followees\_s
7. num\_followers\_d
8. num\_followees\_d
9. inter\_followers
10. inter\_followees

```
if not os.path.isfile('/drive/My Drive/Colab Notebooks/Assignment/data/fea_sampl
#mapping jaccrd followers to train and test data
df_final_train['jaccard_followers'] = df_final_train.apply(lambda row:
                                                             jaccard_for_followers(row['source_no
df_final_test['jaccard_followers'] = df_final_test.apply(lambda row:
                                                            jaccard_for_followers(row['source_no

#mapping jaccrd followees to train and test data
df_final_train['jaccard_followees'] = df_final_train.apply(lambda row:
                                                            jaccard_for_followees(row['source_no
df_final_test['jaccard_followees'] = df_final_test.apply(lambda row:
                                                            jaccard_for_followees(row['source_no

    #mapping jaccrd followers to train and test data
df_final_train['cosine_followers'] = df_final_train.apply(lambda row:
                                                            cosine_for_followers(row['source_nod
df_final_test['cosine_followers'] = df_final_test.apply(lambda row:
                                                            cosine_for_followers(row['source_nod

#mapping jaccrd followees to train and test data
df_final_train['cosine_followees'] = df_final_train.apply(lambda row:
                                                            cosine_for_followees(row['source_nod
df_final_test['cosine_followees'] = df_final_test.apply(lambda row:
                                                            cosine_for_followees(row['source_nod
```

```
def compute_features_stage1(df_final):
    #calculating no of followers followees for source and destination
    #calculating intersection of followers and followees for source and destination
    num_followers_s=[]
    num_followees_s=[]
    num_followers_d=[]
    num_followees_d=[]
    inter_followers=[]
    inter_followees=[]
    for i,row in df_final.iterrows():
        try:
            s1=set(train_graph.predecessors(row['source_node']))
            s2=set(train_graph.successors(row['source_node']))
        except:
            s1 = set()
            s2 = set()
        try:
            d1=set(train_graph.predecessors(row['destination_node']))
            d2=set(train_graph.successors(row['destination_node']))
        except:
            d1 = set()
            d2 = set()
        num_followers_s.append(len(s1))
        num_followees_s.append(len(s2))

        num_followers_d.append(len(d1))
        num_followees_d.append(len(d2))

        inter_followers.append(len(s1.intersection(d1)))
        inter_followees.append(len(s2.intersection(d2)))

    return num_followers_s, num_followers_d, num_followees_s, num_followees_d, i

from pandas import HDFStore,DataFrame
```



```

if not os.path.isfile('/drive/My Drive/Colab Notebooks/Assignment/data/fea_sample/feats.h5'):
    df_final_train['num_followers_s'], df_final_train['num_followers_d'], \
    df_final_train['num_followees_s'], df_final_train['num_followees_d'], \
    df_final_train['inter_followers'], df_final_train['inter_followees'] = compute_features(
        df_final_train['source'], df_final_train['target'], df_final_train['source_id'], df_final_train['target_id'],
        df_final_test['source'], df_final_test['target'], df_final_test['source_id'], df_final_test['target_id'],
        df_final_test['inter_followers'], df_final_test['inter_followees'])

    hdf = HDFStore('/drive/My Drive/Colab Notebooks/Assignment/data/fea_sample/feats.h5')
    hdf.put('train_df', df_final_train, format='table', data_columns=True)
    hdf.put('test_df', df_final_test, format='table', data_columns=True)
    hdf.close()
else:
    df_final_train = read_hdf('/drive/My Drive/Colab Notebooks/Assignment/data/feats.h5', 'train_df')
    df_final_test = read_hdf('/drive/My Drive/Colab Notebooks/Assignment/data/feats.h5', 'test_df')

```

## ▼ 5.3 Adding new set of features

**we will create these each of these features for both train and test data points**

1. adar index
2. is following back
3. belongs to same weakly connect components
4. shortest path between source and destination

```

if not os.path.isfile('/drive/My Drive/Colab Notebooks/Assignment/data/fea_sampl
#mapping adar index on train
df_final_train['adar_index'] = df_final_train.apply(lambda row: calc_adar_in
#mapping adar index on test
df_final_test['adar_index'] = df_final_test.apply(lambda row: calc_adar_in(r

#-----
#mapping followback or not on train
df_final_train['follows_back'] = df_final_train.apply(lambda row: follows_ba

#mapping followback or not on test
df_final_test['follows_back'] = df_final_test.apply(lambda row: follows_back

#-----
#mapping same component of wcc or not on train
df_final_train['same_comp'] = df_final_train.apply(lambda row: belongs_to_sa

##mapping same component of wcc or not on train
df_final_test['same_comp'] = df_final_test.apply(lambda row: belongs_to_same

#-----
#mapping shortest path on train
df_final_train['shortest_path'] = df_final_train.apply(lambda row: compute_s
#mapping shortest path on test
df_final_test['shortest_path'] = df_final_test.apply(lambda row: compute_sho

hdf = HDFStore('/drive/My Drive/Colab Notebooks/Assignment/data/fea_sample/s
hdf.put('train_df',df_final_train, format='table', data_columns=True)
hdf.put('test_df',df_final_test, format='table', data_columns=True)
hdf.close()
else:
df_final_train = read_hdf('/drive/My Drive/Colab Notebooks/Assignment/data/f
df_final_test = read_hdf('/drive/My Drive/Colab Notebooks/Assignment/data/fe

```

## ▼ 5.4 Adding new set of features

**we will create these each of these features for both train and test data points**

### 1. Weight Features

- weight of incoming edges
- weight of outgoing edges
- weight of incoming edges + weight of outgoing edges
- weight of incoming edges \* weight of outgoing edges
- 2\*weight of incoming edges + weight of outgoing edges
- weight of incoming edges + 2\*weight of outgoing edges

### 2. Page Ranking of source

### 3. Page Ranking of dest

### 4. katz of source

### 5. katz of dest

### 6. hubs of source

### 7. hubs of dest

### 8. authorities\_s of source

### 9. authorities\_s of dest

## ▼ Weight Features

In order to determine the similarity of nodes, an edge weight value was calculated between nodes. Edge weight decreases as the neighbor count goes up. Intuitively, consider one million people following a celebrity on a social network then chances are most of them never met each other or the celebrity. On the other hand, if a user has 30 contacts in his/her social network, the chances are higher that many of them know each other. credit - Graph-based Features for Supervised Link Prediction William Cukierski, Benjamin Hamner, Bo Yang

$$W = \frac{1}{\sqrt{1 + |X|}}$$

it is directed graph so calculated Weighted in and Weighted out differently

```

from tqdm import tqdm
import numpy as np
from scipy.sparse.linalg import svds, eigs
import gc

```

```

#weight for source and destination of each link

```

```

Weight_in = {}

```

```

Weight_out = {}

```

```

for i in tqdm(train_graph.nodes()):
    s1=set(train_graph.predecessors(i))
    w_in = 1.0/(np.sqrt(1+len(s1)))
    Weight_in[i]=w_in

```

```

    s2=set(train_graph.successors(i))

```

```

    w_out = 1.0/(np.sqrt(1+len(s2)))

```

```

    Weight_out[i]=w_out

```

```

#for imputing with mean

```

```

mean_weight_in = np.mean(list(Weight_in.values()))

```

```

mean_weight_out = np.mean(list(Weight_out.values()))

```

```

100%|██████████| 1780722/1780722 [00:18<00:00, 96071.03it/s]

```

```

if not os.path.isfile('/drive/My Drive/Colab Notebooks/Assignment/data/fea_sampl

```

```

    #mapping to pandas train

```

```

    df_final_train['weight_in'] = df_final_train.destination_node.apply(lambda x

```

```

    df_final_train['weight_out'] = df_final_train.source_node.apply(lambda x: We

```

```

#mapping to pandas test

```

```

df_final_test['weight_in'] = df_final_test.destination_node.apply(lambda x:

```

```

df_final_test['weight_out'] = df_final_test.source_node.apply(lambda x: Weig

```

```

#some features engineerings on the in and out weights

```

```

df_final_train['weight_f1'] = df_final_train.weight_in + df_final_train.weig

```

```

df_final_train['weight_f2'] = df_final_train.weight_in * df_final_train.weig

```

```

df_final_train['weight_f3'] = (2*df_final_train.weight_in + 1*df_final_train

```

```

df_final_train['weight_f4'] = (1*df_final_train.weight_in + 2*df_final_train

```

```

#some features engineerings on the in and out weights

```

```

df_final_test['weight_f1'] = df_final_test.weight_in + df_final_test.weight_

```

```

df_final_test['weight_f2'] = df_final_test.weight_in * df_final_test.weight_

```

```

df_final_test['weight_f3'] = (2*df_final_test.weight_in + 1*df_final_test.we

```

```

df_final_test['weight_f4'] = (1*df_final_test.weight_in + 2*df_final_test.we

```

```

if not os.path.isfile('/drive/My Drive/Colab Notebooks/Assignment/data/fea_sampl

#page rank for source and destination in Train and Test
#if anything not there in train graph then adding mean page rank
df_final_train['page_rank_s'] = df_final_train.source_node.apply(lambda x:pr
df_final_train['page_rank_d'] = df_final_train.destination_node.apply(lambda

df_final_test['page_rank_s'] = df_final_test.source_node.apply(lambda x:pr.g
df_final_test['page_rank_d'] = df_final_test.destination_node.apply(lambda x
#=====

#Katz centrality score for source and destination in Train and test
#if anything not there in train graph then adding mean katz score
df_final_train['katz_s'] = df_final_train.source_node.apply(lambda x: katz.g
df_final_train['katz_d'] = df_final_train.destination_node.apply(lambda x: k

df_final_test['katz_s'] = df_final_test.source_node.apply(lambda x: katz.get
df_final_test['katz_d'] = df_final_test.destination_node.apply(lambda x: kat
#=====

#Hits algorithm score for source and destination in Train and test
#if anything not there in train graph then adding 0
df_final_train['hubs_s'] = df_final_train.source_node.apply(lambda x: hits[0]
df_final_train['hubs_d'] = df_final_train.destination_node.apply(lambda x: h

df_final_test['hubs_s'] = df_final_test.source_node.apply(lambda x: hits[0].
df_final_test['hubs_d'] = df_final_test.destination_node.apply(lambda x: hit
#=====

#Hits algorithm score for source and destination in Train and Test
#if anything not there in train graph then adding 0
df_final_train['authorities_s'] = df_final_train.source_node.apply(lambda x:
df_final_train['authorities_d'] = df_final_train.destination_node.apply(lamb

df_final_test['authorities_s'] = df_final_test.source_node.apply(lambda x: h
df_final_test['authorities_d'] = df_final_test.destination_node.apply(lambda
#=====

hdf = HDFStore('/drive/My Drive/Colab Notebooks/Assignment/data/fea_sample/s
hdf.put('train_df',df_final_train, format='table', data_columns=True)
hdf.put('test_df',df_final_test, format='table', data_columns=True)
hdf.close()
else:
df_final_train = read_hdf('/drive/My Drive/Colab Notebooks/Assignment/data/f
df_final_test = read_hdf('/drive/My Drive/Colab Notebooks/Assignment/data/fe

```

## ▼ 5.5 Adding new set of features

**we will create these each of these features for both train and test data points**

1. SVD features for both source and destination

```
def svd(x, S):
    try:
        z = sadj_dict[x]
        return S[z]
    except:
        return [0,0,0,0,0,0]
```

#for svd features to get feature vector creating a dict node val and index in sv

```
sadj_col = sorted(train_graph.nodes())
sadj_dict = { val:idx for idx,val in enumerate(sadj_col)}
```

Adj = nx.adjacency\_matrix(train\_graph,nodelist=sorted(train\_graph.nodes())).asf

```
U, s, V = svds(Adj, k = 6)
print('Adjacency matrix Shape',Adj.shape)
print('U Shape',U.shape)
print('V Shape',V.shape)
print('s Shape',s.shape)
```

```
Adjacency matrix Shape (1780722, 1780722)
U Shape (1780722, 6)
V Shape (6, 1780722)
s Shape (6,)
```

```

if not os.path.isfile('/drive/My Drive/Colab Notebooks/Assignment/data/fea_sampl
#=====

df_final_train[['svd_u_s_1', 'svd_u_s_2', 'svd_u_s_3', 'svd_u_s_4', 'svd_u_s_
df_final_train.source_node.apply(lambda x: svd(x, U)).apply(pd.Series)

df_final_train[['svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3', 'svd_u_d_4', 'svd_u_d
df_final_train.destination_node.apply(lambda x: svd(x, U)).apply(pd.Series)
#=====

df_final_train[['svd_v_s_1', 'svd_v_s_2', 'svd_v_s_3', 'svd_v_s_4', 'svd_v_s_
df_final_train.source_node.apply(lambda x: svd(x, V.T)).apply(pd.Series)

df_final_train[['svd_v_d_1', 'svd_v_d_2', 'svd_v_d_3', 'svd_v_d_4', 'svd_v_d
df_final_train.destination_node.apply(lambda x: svd(x, V.T)).apply(pd.Series
#=====

df_final_test[['svd_u_s_1', 'svd_u_s_2', 'svd_u_s_3', 'svd_u_s_4', 'svd_u_s_5
df_final_test.source_node.apply(lambda x: svd(x, U)).apply(pd.Series)

df_final_test[['svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3', 'svd_u_d_4', 'svd_u_d
df_final_test.destination_node.apply(lambda x: svd(x, U)).apply(pd.Series)

#=====

df_final_test[['svd_v_s_1', 'svd_v_s_2', 'svd_v_s_3', 'svd_v_s_4', 'svd_v_s_5
df_final_test.source_node.apply(lambda x: svd(x, V.T)).apply(pd.Series)

df_final_test[['svd_v_d_1', 'svd_v_d_2', 'svd_v_d_3', 'svd_v_d_4', 'svd_v_d
df_final_test.destination_node.apply(lambda x: svd(x, V.T)).apply(pd.Series)
#=====

hdf = HDFStore('/drive/My Drive/Colab Notebooks/Assignment/data/fea_sample/s
hdf.put('train_df', df_final_train, format='table', data_columns=True)
hdf.put('test_df', df_final_test, format='table', data_columns=True)
hdf.close()

```

## ▼ Assignments:

1. Add another feature called Preferential Attachment with followers and followees data of vertex. you can check about Preferential Attachment in below link  
<http://be.amazd.com/link-prediction/>
2. Add feature called svd\_dot. you can calculate svd\_dot as Dot product between source node svd and destination node svd features. you can read about this in below pdf  
[https://storage.googleapis.com/kaggle-forum-message-attachments/2594/supervised\\_link\\_prediction.pdf](https://storage.googleapis.com/kaggle-forum-message-attachments/2594/supervised_link_prediction.pdf)
3. Tune hyperparameters for XG boost with all these features and check the error metric.

### ▼ 1. Preferential Attachment

$$Score(x, y) = |x| \cdot |y|$$

```
# for followees
def preferential_attachment_for_followees(a,b):
    try:
        if len(set(train_graph.successors(a))) == 0 | len(set(train_graph.succes
            return 0
        score = (len(set(train_graph.successors(a))) * len(set(train_graph.succe

        return score
    except:
        return 0
```

```
#one test case
print(preferential_attachment_for_followees(273084,1505602))
```

```
#node 1635354 not in graph
print(preferential_attachment_for_followees(273084,1505602))
```

```
120
120
```



```
# for followers
def preferential_attachment_for_followers(a,b):
    try:
        if len(set(train_graph.predecessors(a))) == 0 | len(set(train_graph.pred
            return 0
        score = (len(set(train_graph.predecessors(a))) * len(set(train_graph.pre
        return score
    except:
        return 0
```

```
#one test case
print(preferential_attachment_for_followers(273084,1505602))
```

```
#node 1635354 not in graph
print(preferential_attachment_for_followers(273084,1505602))
```

```
66
66
```

```
# adding above svd_dot into dataframes
#followers
df_final_train["preferential_followers"] = df_final_train.apply(lambda row: pref
df_final_test["preferential_followers"] = df_final_train.apply(lambda row: prefe
#followees
df_final_train["preferential_followees"] = df_final_train.apply(lambda row: pref
df_final_test["preferential_followees"] = df_final_train.apply(lambda row: prefe
```

## ▼ 2 SVD\_dot

Add feature called svd\_dot. you can calculate svd\_dot as Dot product between source node svd and destination node svd features.

```
def svd(x, S):
    try:
        z = sadj_dict[x]
        return S[z]
    except:
        return [0,0,0,0,0,0]
```

```
#for svd features to get feature vector creating a dict node val and index in sv
sadj_col = sorted(train_graph.nodes())
sadj_dict = { val:idx for idx,val in enumerate(sadj_col)}
```

```
Adj = nx.adjacency_matrix(train_graph,nodelist=sorted(train_graph.nodes())).asf
```

```
Adj = Adj.asfptype()
Adj
```

```
<1780722x1780722 sparse matrix of type '<class 'numpy.float64'>'
  with 7542469 stored elements in Compressed Sparse Row format>
```

```
U, s, V = svds(Adj, k = 6)
print('Adjacency matrix Shape',Adj.shape)
print('U Shape',U.shape)
print('V Shape',V.shape)
print('s Shape',s.shape)
```

```
Adjacency matrix Shape (1780722, 1780722)
U Shape (1780722, 6)
V Shape (6, 1780722)
s Shape (6,)
```

```
del V
del s
```

```
# SVD dot product of source and destination vectors
# training data
from tqdm import tqdm
svd_dot_train = []
```

```
for indx, temp_series in tqdm(df_final_train.iterrows(), total=df_final_train.sh
    in_indx = sadj_dict.get(temp_series.destination_node, 'X')
    out_indx = sadj_dict.get(temp_series.source_node, 'X')
    #print(in_indx , out_indx)
    if ( in_indx != 'X' and out_indx != 'X' ):
        #dot product of svd vector of Source and destination
        svd_temp = np.dot(U[in_indx,:],U[out_indx,:])
        svd_dot_train.append(svd_temp)
    else:
        svd_dot_train.append(0)
```

```
100%|██████████| 100002/100002 [00:09<00:00, 10373.05it/s]
```

```
# SVD dot product of source and destination vectors
# test data
from tqdm import tqdm
svd_dot_test = []

for indx, temp_series in tqdm(df_final_test.iterrows(), total=df_final_test.shape):
    in_indx = sadj_dict.get(temp_series.destination_node, 'X')
    out_indx = sadj_dict.get(temp_series.source_node, 'X')
    #print(in_indx , out_indx)
    if ( in_indx != 'X' and out_indx != 'X' ):
        #dot product of svd vector of Source and destination
        svd_temp = np.dot(U[in_indx,:],U[out_indx,:])
        svd_dot_test.append(svd_temp)
    else:
        svd_dot_test.append(0)
```

100%|██████████| 50002/50002 [00:07<00:00, 6530.62it/s]

```
# adding above svd_dot into dataframes
df_final_train["svd_dot"] = svd_dot_train
df_final_test["svd_dot"] = svd_dot_test
```

```
# save the train and test datas
```

```
hdf = HDFStore('/drive/My Drive/Colab Notebooks/Assignment/data/fea_sample/final')
hdf.put('train_df', df_final_train, format='table', data_columns=True)
hdf.put('test_df', df_final_test, format='table', data_columns=True)
hdf.close()
```

## ▼ feature models

```
#reading
from pandas import read_hdf
df_final_train = read_hdf('/drive/My Drive/Colab Notebooks/Assignment/data/fea_s')
df_final_test = read_hdf('/drive/My Drive/Colab Notebooks/Assignment/data/fea_sa')
```

```
df_final_train.columns
```

```
Index(['source_node', 'destination_node', 'indicator_link',
      'jaccard_followers', 'jaccard_followees', 'cosine_followers',
      'cosine_followees', 'num_followers_s', 'num_followers_d',
      'num_followees_s', 'num_followees_d', 'inter_followers',
      'inter_followees', 'adar_index', 'follows_back', 'same_comp',
      'shortest_path', 'weight_in', 'weight_out', 'weight_f1', 'weight_f2',
      'weight_f3', 'weight_f4', 'page_rank_s', 'page_rank_d', 'katz_s',
      'katz_d', 'hubs_s', 'hubs_d', 'authorities_s', 'authorities_d',
      'svd_u_s_1', 'svd_u_s_2', 'svd_u_s_3', 'svd_u_s_4', 'svd_u_s_5',
      'svd_u_s_6', 'svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3', 'svd_u_d_4',
      'svd_u_d_5', 'svd_u_d_6', 'svd_v_s_1', 'svd_v_s_2', 'svd_v_s_3',
      'svd_v_s_4', 'svd_v_s_5', 'svd_v_s_6', 'svd_v_d_1', 'svd_v_d_2',
      'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5', 'svd_v_d_6',
      'preferential_followers', 'preferential_followees', 'svd_dot'],
      dtype='object')
```

```
y_train = df_final_train.indicator_link
y_test = df_final_test.indicator_link
```

## ▼ XG boost

```
from sklearn.metrics import f1_score
from xgboost import XGBClassifier
from sklearn.metrics import f1_score
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint as sp_randint
from scipy.stats import uniform
```

```
param_dist = {"n_estimators": sp_randint(105,125),
              "max_depth": sp_randint(10,15),
              "min_samples_split": sp_randint(110,190),
              "min_samples_leaf": sp_randint(25,65)}

clf = XGBClassifier(random_state=25)\

rf_random = RandomizedSearchCV(clf, param_distributions=param_dist,
                               n_iter=5, cv=10, scoring='f1', random_state=2)

rf_random.fit(df_final_train, y_train)
print('mean test scores', rf_random.cv_results_['mean_test_score'])
print('mean train scores', rf_random.cv_results_['mean_train_score'])

print(rf_random.best_estimator_)

clf = XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                    colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
                    max_depth=11, min_child_weight=1, min_samples_leaf=56,
                    min_samples_split=179, missing=None, n_estimators=106, n_jobs=1,
                    nthread=None, objective='binary:logistic', random_state=25,
                    reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
                    silent=True, subsample=1)

from sklearn.metrics import f1_score
print('Train f1 score', f1_score(y_train, y_train_pred))
print('Test f1 score', f1_score(y_test, y_test_pred))
```

```
from sklearn.metrics import confusion_matrix
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)

    A = (((C.T)/(C.sum(axis=1))).T)

    B = (C/C.sum(axis=0))
    plt.figure(figsize=(20,4))

    labels = [0,1]
    # representing A in heatmap format
    cmap=sns.light_palette("blue")
    plt.subplot(1, 3, 1)
    sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, ytickl
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Confusion matrix")

    plt.subplot(1, 3, 2)
    sns.heatmap(B, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, ytickl
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Precision matrix")

    plt.subplot(1, 3, 3)
    # representing B in heatmap format
    sns.heatmap(A, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, ytickl
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()

print('Train confusion_matrix')
plot_confusion_matrix(y_train,y_train_pred)
print('Test confusion_matrix')
plot_confusion_matrix(y_test,y_test_pred)
```

```
from sklearn.metrics import roc_curve, auc
fpr,tpr,ths = roc_curve(y_test,y_test_pred)
auc_sc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='navy',label='ROC curve (area = %0.2f)' % auc_sc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic with test data')
plt.legend()
plt.show()

features = df_final_train.columns
importances = clf.feature_importances_
indices = (np.argsort(importances))[-25:]
plt.figure(figsize=(10,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='r', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```

- ▼ step by step procedure you followed to solve this case study

- first we import some Libraries
- using networkx library load the data without header
- we observe that the data set contains only two features those are 'source\_node' and 'destination\_node'
- we Map this problem into supervised learning problem
- we use performance metrics for this problem are f1 score and confusion matrix
- this data set contains only directed edges
- we do some EDA on graph type features
- Type: DiGraph
- Number of nodes: 1862220
- Number of edges: 9437519
- Average in degree: 5.0679
- Average out degree: 5.0679



# EDA

## No of followers for each person

- we observe that very few have more connections
- 99% of nodes are have just less than 40 connections
- 99.9 percentile is 112

## No of people each person is following

- we observe that very few have more connections
- 99% of nodes are have just less than 40 following
- 99.9 percentile is 123
- No of persons those are not following anyone are 274512 and % is 14.741115442858524
- No of persons having zero followers are 188043 and % is 10.097786512871734

## both followers + following

- we observe that very few have more connections
- 99% of nodes are have just less than 79
- 99.9 percentile is 221
- Min of no of followers + following is 1334291 persons having minimum no of followers + following
- Max of no of followers + following is 15791 persons having maximum no of followers + following
- No of persons having followers + following less than 10 are 1320326
- No of weakly connected components 45558 weakly connected components with 2 nodes 32195

## Posing a problem as classification problem

Generating some edges which are not present in graph for supervised learning

- Generated Bad links from graph which are not in graph and whose shortest path is greater than 2.

Training and Test data split

- Removed edges from Graph and used as test data and after removing used that graph for creating features for Train and test data
- we done 80:20 split as train and test data
- Data points in train data (15100030, 2)
- Data points in test data (3775008, 2)
- Shape of target variable in train (15100030,)
- Shape of target variable in test (3775008,)

### ▼ Featurization

## Similarity measures

- Jucard Distance

$$j = \frac{|X \cap Y|}{|X \cup Y|}$$

- Cosine distance

$$\text{CosineDistance} = \frac{|X \cap Y|}{|X| \cdot |Y|}$$

## Ranking Measures

- PageRank computes a ranking of the nodes in the graph G based on the structure of the incoming links.
- Page Ranking
- 

## Other Graph Features

- Shortest path
- checking for same community
- adamic/Adar index

$$A(x, y) = \sum_{u \in N(x) \cap N(y)} \frac{1}{\log(|N(u)|)}$$

- Is person was following back
- Katz Centrality
- Hits Score

## ▼ Adding new set of features

**we will create these each of these features for both train and test data points**

### 1. Weight Features

- weight of incoming edges
- weight of outgoing edges
- weight of incoming edges + weight of outgoing edges
- weight of incoming edges \* weight of outgoing edges
- $2 \times$  weight of incoming edges + weight of outgoing edges
- weight of incoming edges +  $2 \times$  weight of outgoing edges

### 2. Page Ranking of source

### 3. Page Ranking of dest

### 4. katz of source

### 5. katz of dest

### 6. hubs of source

### 7. hubs of dest

### 8. authorities\_s of source

### 9. authorities\_s of dest

- SVD features for both source and destination
- Add another feature called Preferential Attachment with followers and followees data of vertex. you can check about Preferential Attachment in below link  
<http://be.amazd.com/link-prediction/>
- Add feature called svd\_dot. you can calculate svd\_dot as Dot product between source node svd and destination node svd features. you can read about this in below pdf  
[https://storage.googleapis.com/kaggle-forum-message-attachments/2594/supervised\\_link\\_prediction.pdf](https://storage.googleapis.com/kaggle-forum-message-attachments/2594/supervised_link_prediction.pdf)

## ▼ Modeling

- we are hyperparameter tuning for XG boost with all these features
- we get best train and test accureces
- Train f1 score 0.9892563978754224
- Test f1 score 0.9256446597423477
- we check the error metric using confusion matrices

##### importent features

- hubs\_s
- page\_rank\_d
- svd\_dot
- katz-d
- katz-s
- so...on