Social network Graph Link Prediction - Facebook Challenge

Problem statement:

Given a directed social graph, have to predict missing links to recommend users (Link Prediction in graph)

Data Overview

Taken data from facebook's recruting challenge on kaggle https://www.kaggle.com/c/FacebookRecruiting
data contains two columns source and destination eac edge in graph - Data columns (total 2 columns):
- source_node int64
- destination_node int64

Mapping the problem into supervised learning problem:

- Generated training samples of good and bad links from given directed graph and for each link got some featu page rank, katz score, adar index, some svd fetures of adj matrix, some weight features etc. and trained ml m
- Some reference papers and videos :
    - https://www.cs.cornell.edu/home/kleinber/link-pred.pdf
    - https://www3.nd.edu/~dial/publications/lichtenwalter2010new.pdf
    - https://kaggle2.blob.core.windows.net/forum-message-attachments/2594/supervised_link_prediction.
    - https://www.youtube.com/watch?v=2M77Hgy17cg

Business objectives and constraints:

- No low-latency requirement.
- Probability of prediction is useful to recommend ighest probability links

Performance metric for supervised learning:

- Both precision and recall is important so F1 score is good choice
- Confusion matrix

```
from google.colab import drive
drive.mount('/gdrive')
%cd /gdrive
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=9473189

Enter your authorization code:
··········
Mounted at /gdrive
/gdrive
```

```
#Importing Libraries
```

```
# please do go through this python notebook:
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 aritmetic operations on arrays
# matplotlib: used to plot graphs
import matplotlib
import matplotlib.pylab 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
```

```
#reading graph
if not os.path.isfile('./My Drive/Facebook/data/after_eda/train_woheader.csv'):
    traincsv = pd.read_csv('./My Drive/Facebook/data/train.csv')
    print(traincsv[traincsv.isna().any(1)])
    print(traincsv.info())
    print("Number of diplicate entries: ",sum(traincsv.duplicated()))
    traincsv.to_csv('./My Drive/Facebook/data/after_eda/train_woheader.csv',header=False,index=False
    print("saved the graph into file")
else:
    g=nx.read_edgelist('./My Drive/Facebook/data/after_eda/train_woheader.csv',delimiter=',',create_
    print(nx.info(g))
```

```
Name:
Type: DiGraph
Number of nodes: 1862220
Number of edges: 9437519
Average in degree:   5.0679
Average out degree:   5.0679
```

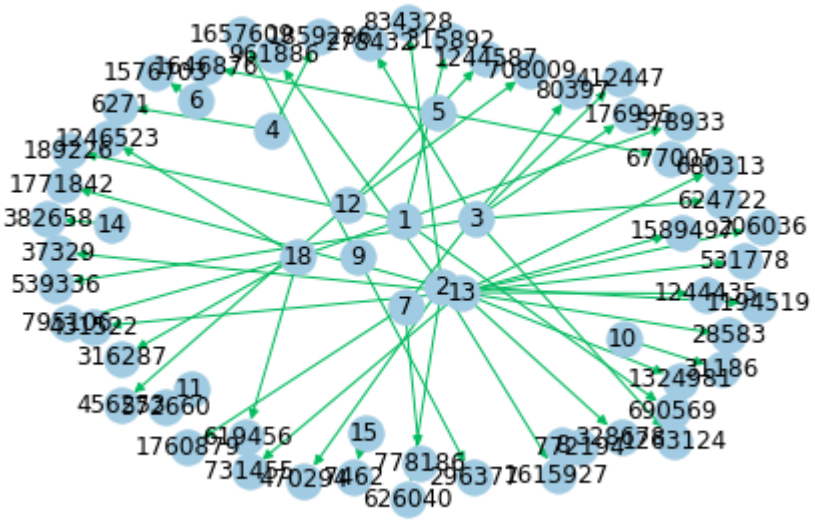
Displaying a sub graph

```
if not os.path.isfile('./My Drive/Facebook/data/train_woheader_sample.csv'):
    df = pd.read_csv('./My Drive/Facebook/data/train.csv', nrows=50)
```

```
df.head(2)
```

```
if not os.path.isfile('./My Drive/Facebook1/data/train_woheader_sample.csv'):
    pd.read_csv('./My Drive/Facebook1/data/train.csv', nrows=50).to_csv('./My Drive/Facebook1/data/t

subgraph=nx.read_edgelist('./My Drive/Facebook1/data/train_woheader_sample.csv',delimiter=',',create
# https://stackoverflow.com/questions/9402255/drawing-a-huge-graph-with-networkx-and-matplotlib

pos=nx.spring_layout(subgraph)
nx.draw(subgraph,pos,node_color='#A0CBE2',edge_color='#00bb5e',width=1,edge_cmap=plt.cm.Blues,with_l
plt.savefig("./My Drive/Facebook1/data/graph_sample.pdf")
print(nx.info(subgraph))
```

```
Name:
Type: DiGraph
Number of nodes: 66
Number of edges: 50
Average in degree:   0.7576
Average out degree:   0.7576
```



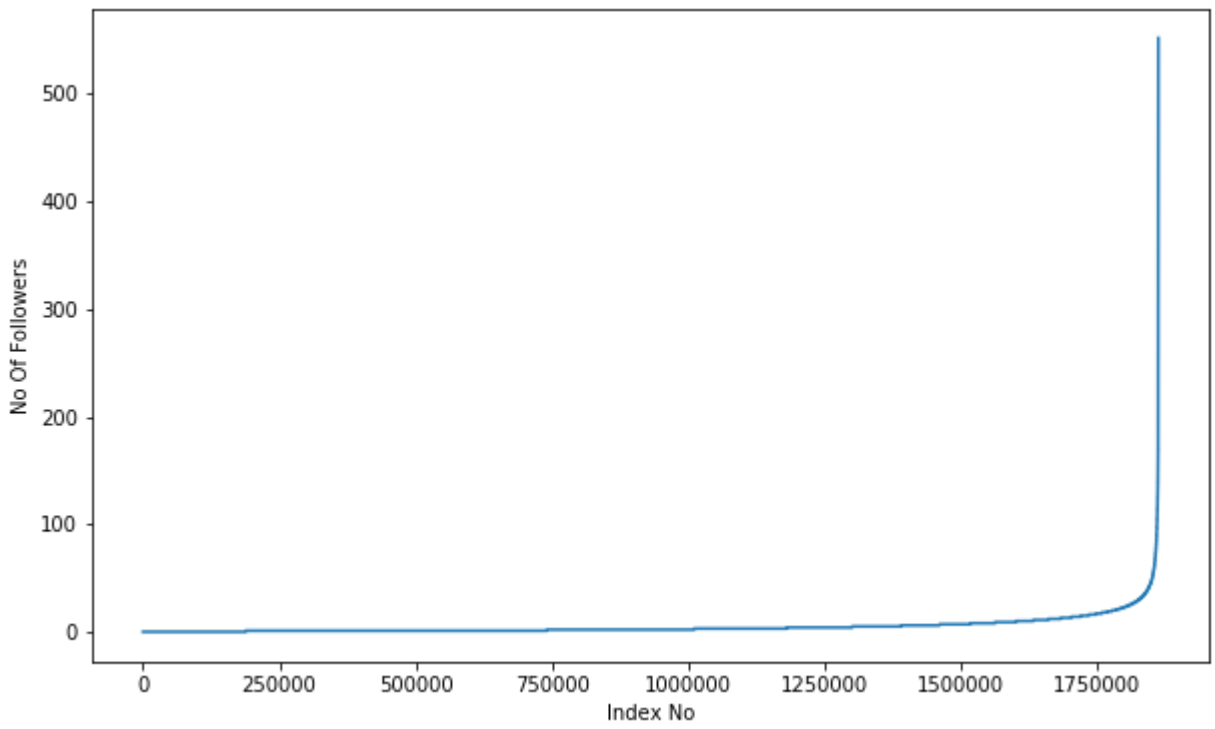
1. Exploratory Data Analysis

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

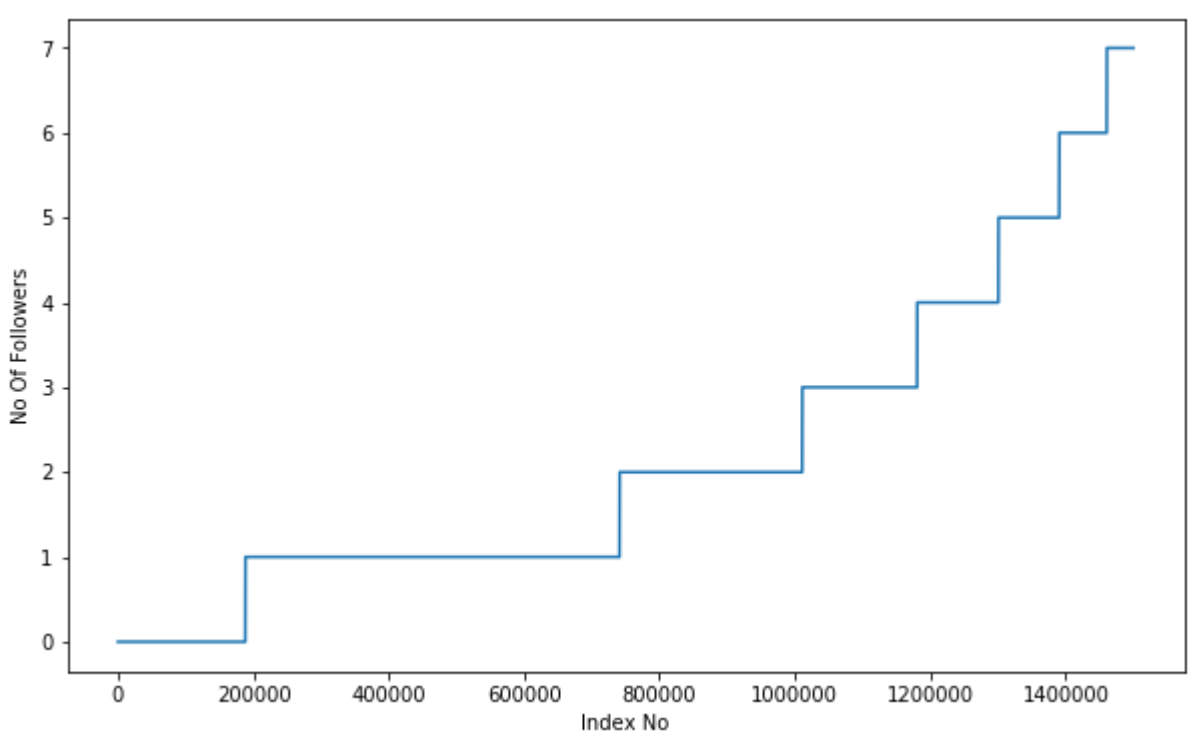
```
The number of unique persons 1862220
```

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()
```

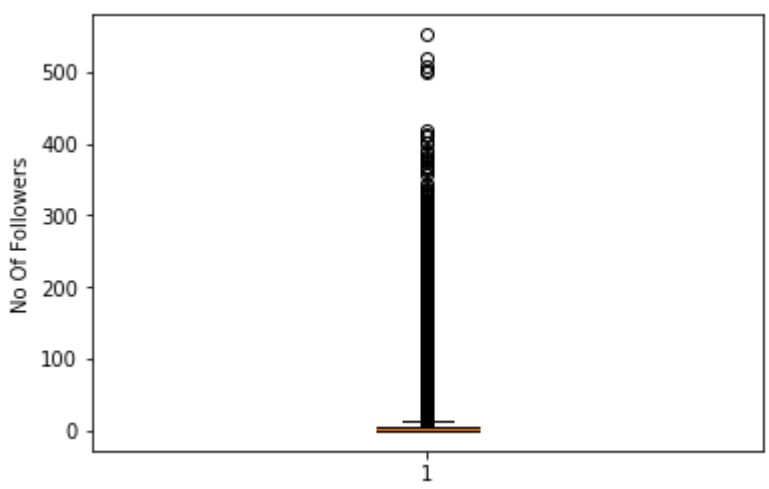



```
indegree_dist = list(dict(g.in_degree()).values())
indegree_dist.sort()
plt.figure(figsize=(10,6))
plt.plot(indegree_dist[0:1500000])
plt.xlabel('Index No')
plt.ylabel('No Of Followers')
plt.show()
```




```
plt.boxplot(indegree_dist)
```

```
plt.ylabel('No Of Followers')

plt.show()
```




```
### 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
```

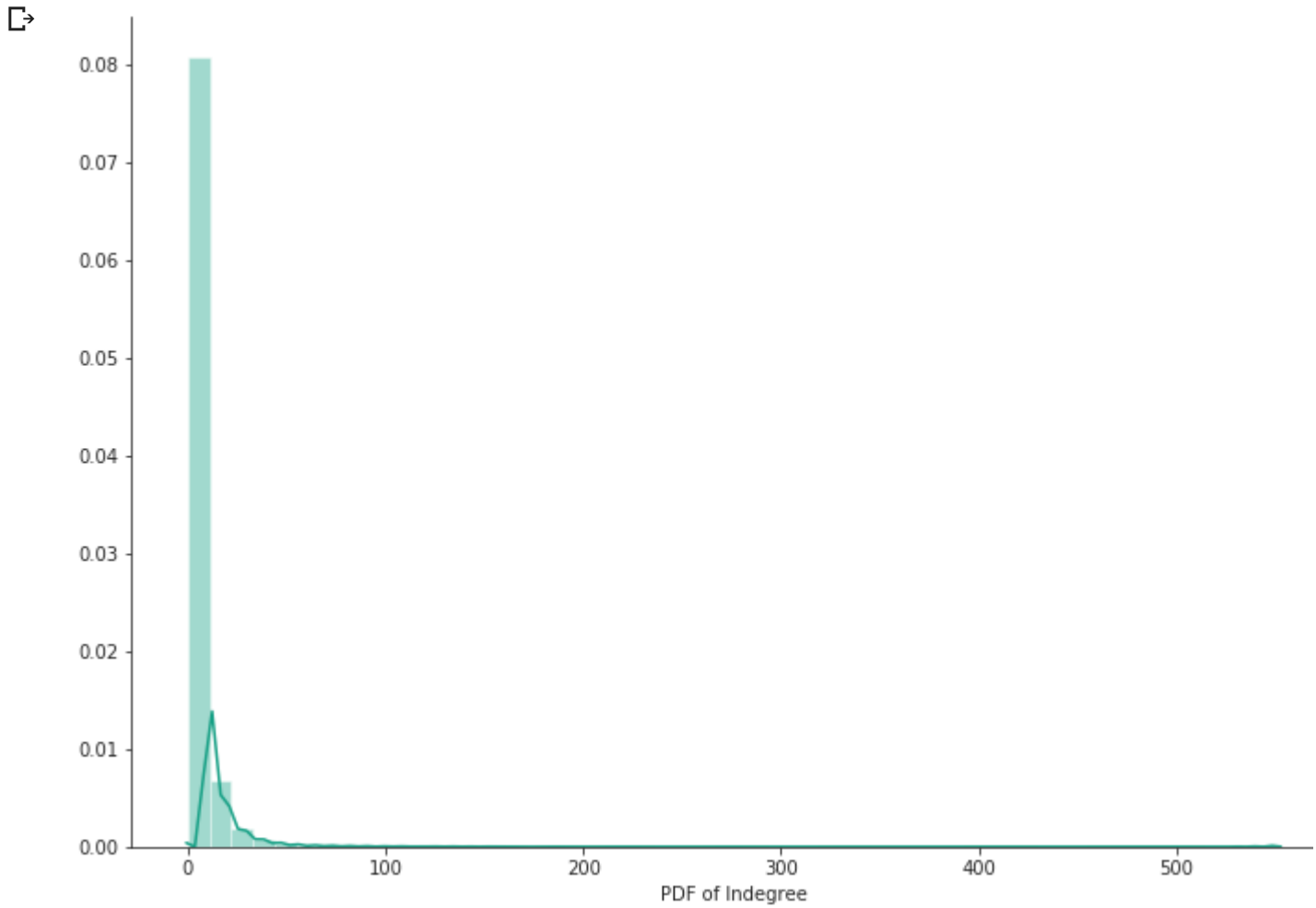
99% of data having followers of 40 only.

```
### 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
```

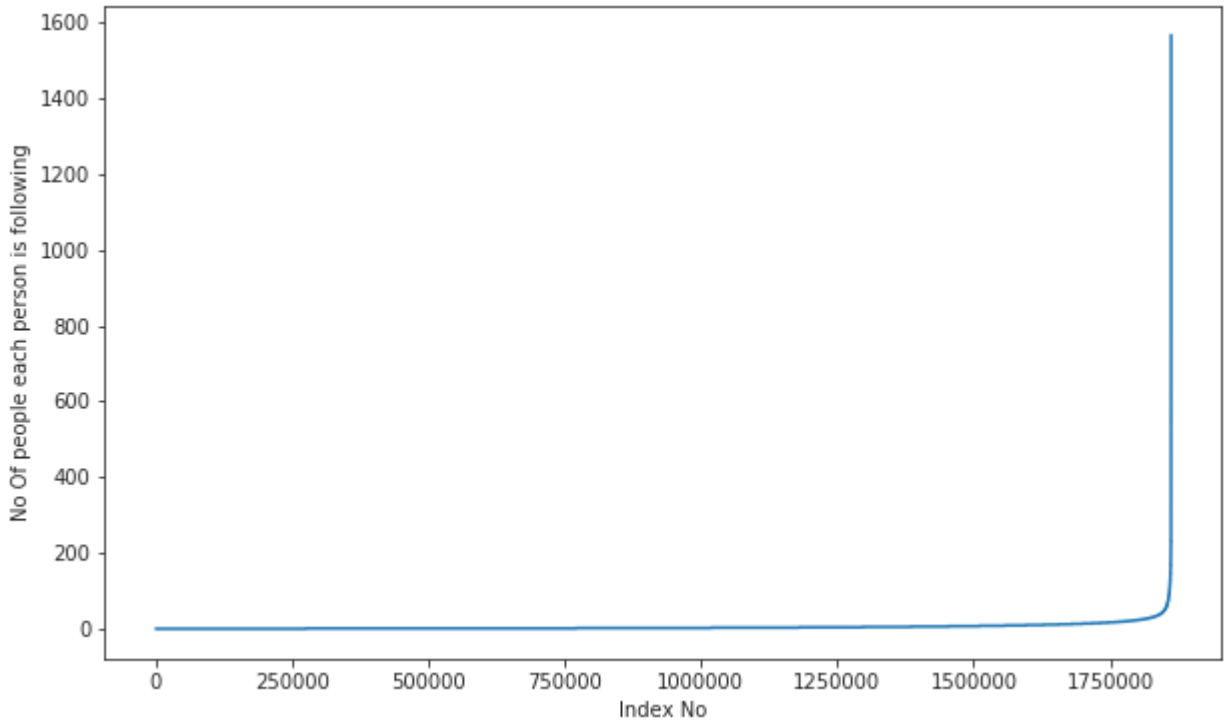
```
%matplotlib inline
sns.set_style('ticks')
```

```
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()
```

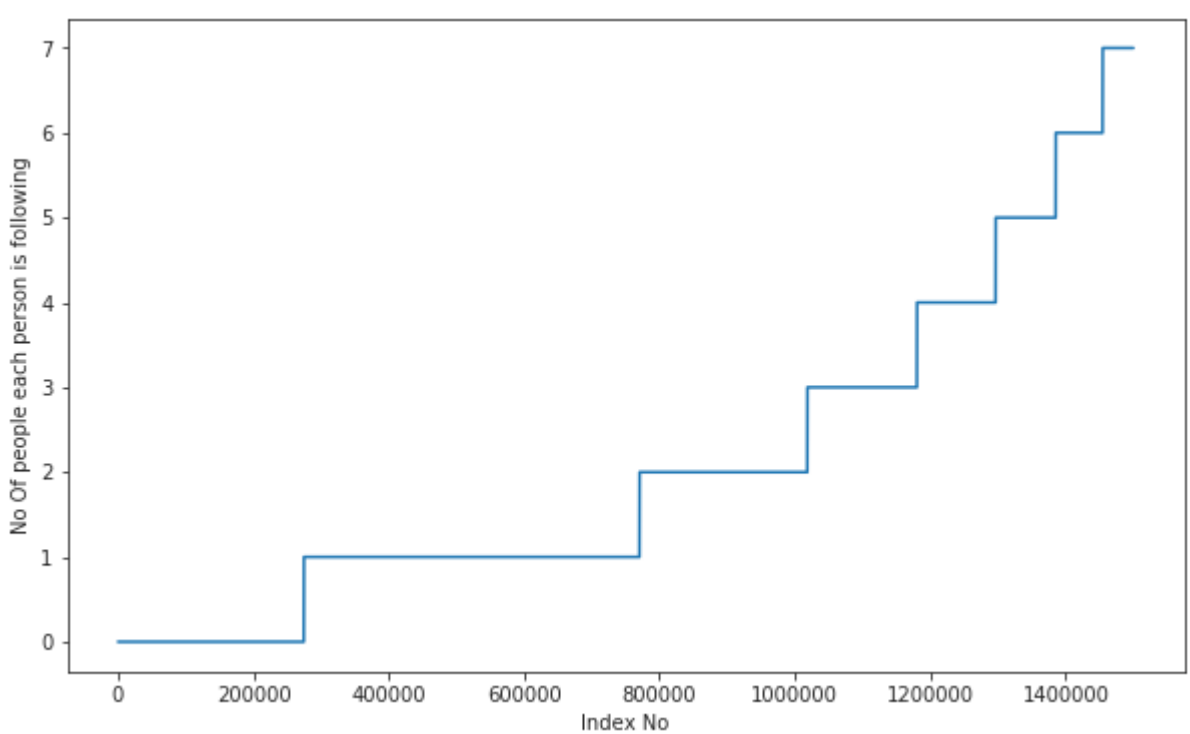


1.2 No of people each person is following

```
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()
```

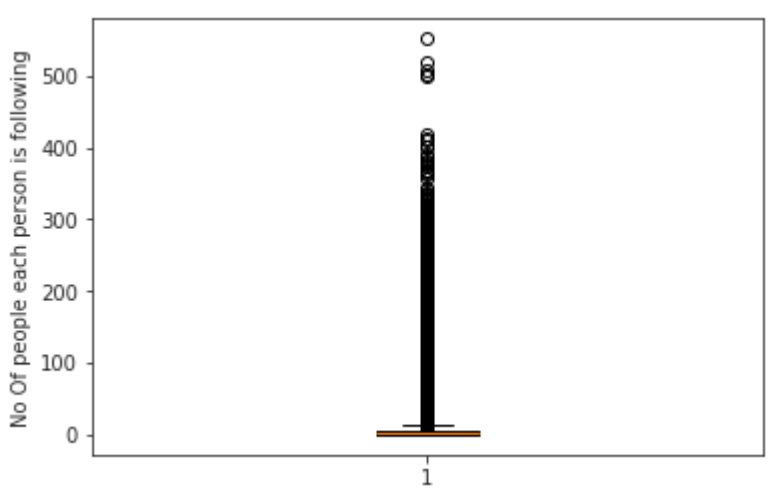



```
indegree_dist = list(dict(g.in_degree()).values())
indegree_dist.sort()
plt.figure(figsize=(10,6))
plt.plot(outdegree_dist[0:1500000])
plt.xlabel('Index No')
plt.ylabel('No Of people each person is following')
plt.show()
```




```
plt.boxplot(indegree_dist)
```

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



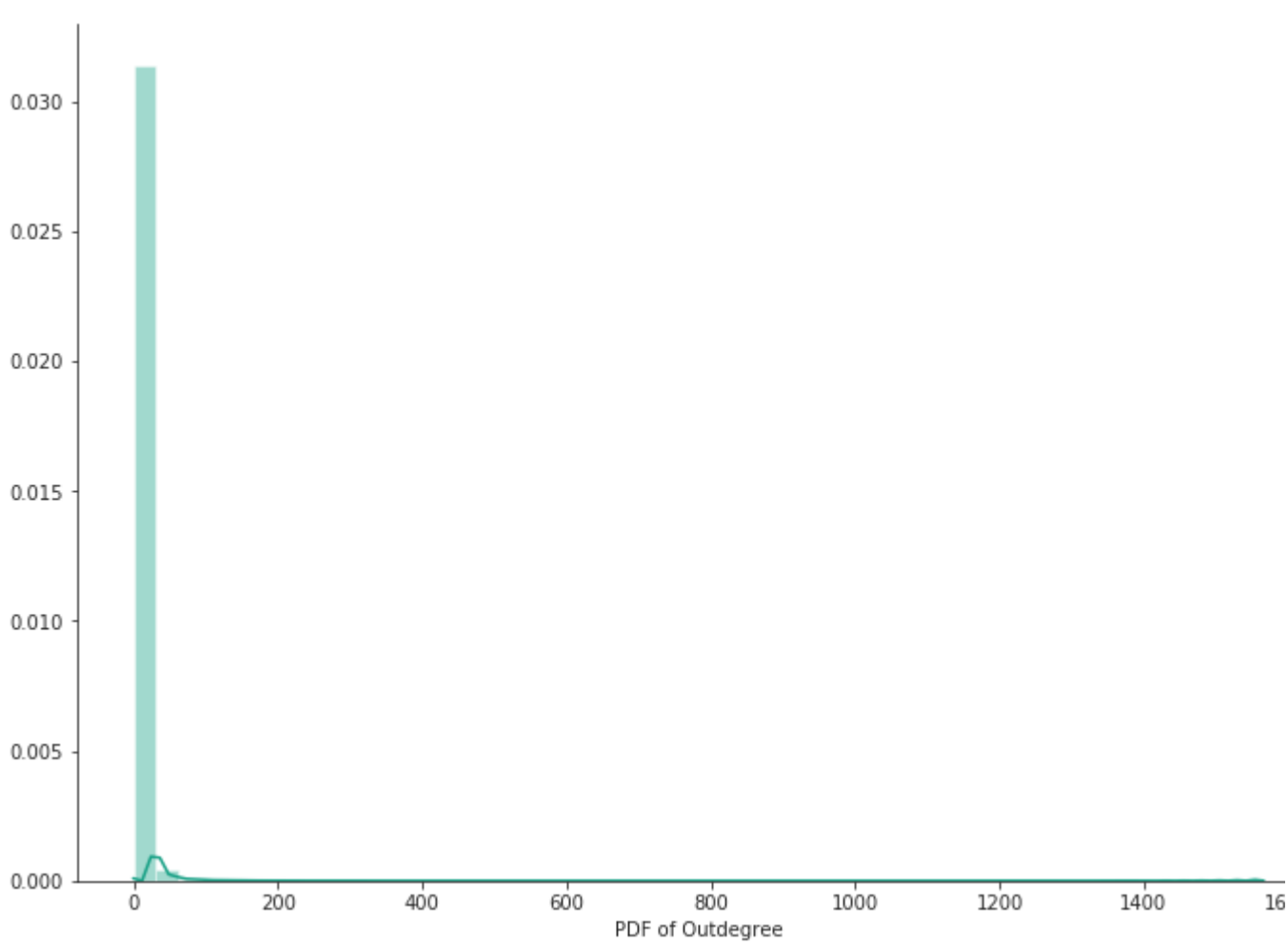

```
### 90-100 percentile
for i in range(0,11):
    print(90+i,'percentile value is',np.percentile(outdegree_dist,90+i))
```

```
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='#16A085')
plt.xlabel('PDF of Outdegree')
sns.despine()
```




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

```
No of persons those are not following anyone are 274512 and % is 14.741115442858524
```

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

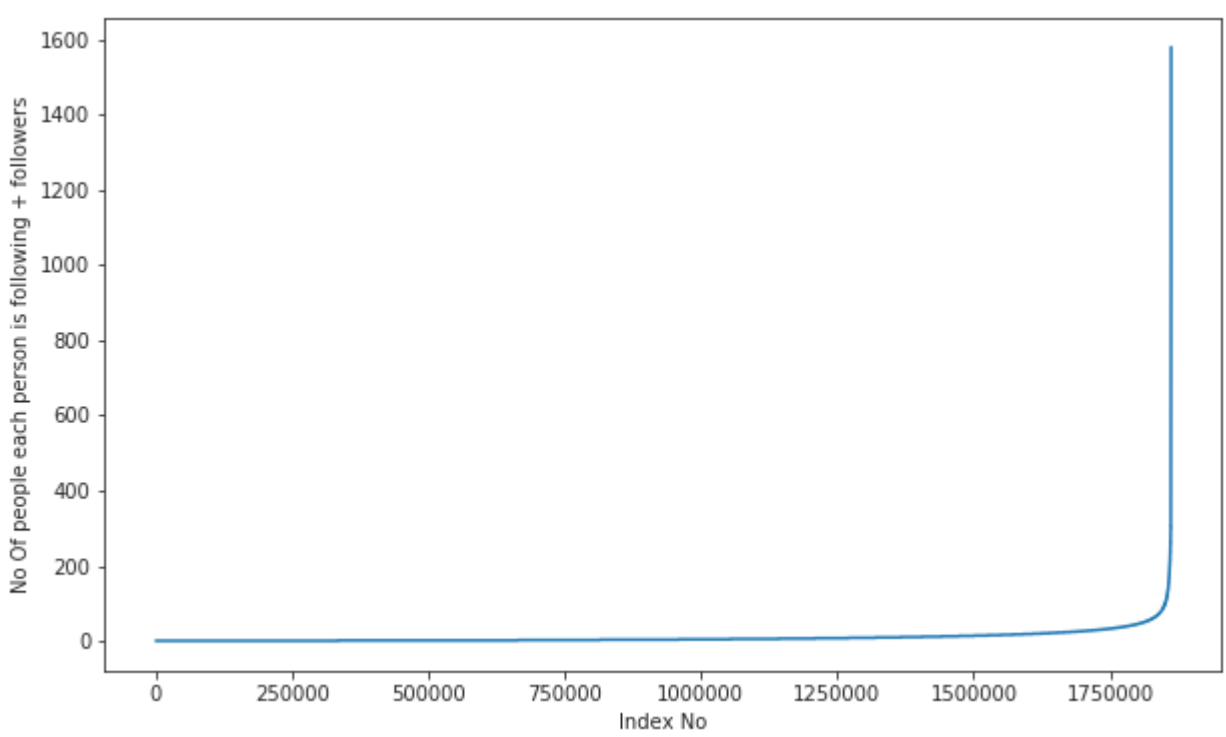
```
No of persons having zero followers are 188043 and % is 10.097786512871734
```

```
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 followers are',count
```

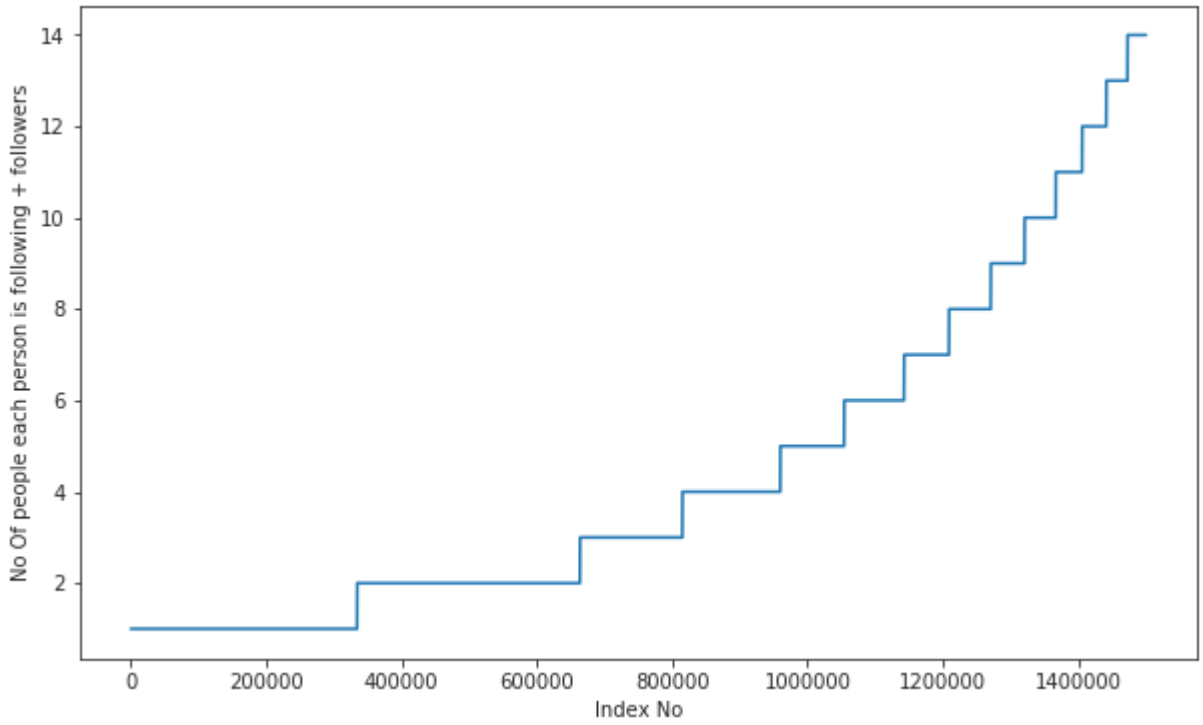
```
No of persons those are not not following anyone and also not having any followers are 0
```

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+(i/100)))
```

```
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 followers + followin
```

```
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 followers + followin
```

```
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_out_degree<10))
```

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

```
print('No of weakly connected components',len(list(nx.weakly_connected_components(g))))
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 le

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('./My Drive/Facebook1/data/after_eda/missing_edges_final.p'):
    #getting all set of edges
    r = csv.reader(open('./My Drive/Facebook1/data/after_eda/train_woheader.csv','r'))
    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
```

```
                    continue
            except:
                    missing_edges.add((a,b))
        else:
            continue
    pickle.dump(missing_edges,open('./My Drive/Facebook1/data/after_eda/missing_edges_final.p','wb')
else:
    missing_edges = pickle.load(open('./My Drive/Facebook1/data/after_eda/missing_edges_final.p','rb
```

```
CPU times: user 2.13 s, sys: 809 ms, total: 2.93 s
Wall time: 10.8 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 Trair

```
from sklearn.model_selection import train_test_split
if (not os.path.isfile('./My Drive/Facebook1/data/after_eda/train_pos_after_eda.csv')) and (not os.p
    #reading total data df
    df_pos = pd.read_csv('data/train.csv')
    df_neg = pd.DataFrame(list(missing_edges), columns=['source_node', 'destination_node'])

    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 training data only for cre
    #and for feature generation
    X_train_pos, X_test_pos, y_train_pos, y_test_pos  = train_test_split(df_pos,np.ones(len(df_pos))
    X_train_neg, X_test_neg, y_train_neg, y_test_neg  = train_test_split(df_neg,np.zeros(len(df_neg)

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

    #removing header and saving
    X_train_pos.to_csv('./My Drive/Facebook1/data/after_eda/train_pos_after_eda.csv',header=False, i
    X_test_pos.to_csv('./My Drive/Facebook1/data/after_eda/test_pos_after_eda.csv',header=False, ind
    X_train_neg.to_csv('./My Drive/Facebook1/data/after_eda/train_neg_after_eda.csv',header=False, i
    X_test_neg.to_csv('./My Drive/Facebook1/data/after_eda/test_neg_after_eda.csv',header=False, ind
else:
    #Graph from Traing data only
    del missing_edges
```

```
if (os.path.isfile('./My Drive/Facebook1/data/after_eda/train_pos_after_eda.csv')) and (os.path.isfi
    train_graph=nx.read_edgelist('./My Drive/Facebook1/data/after_eda/train_pos_after_eda.csv',delim
    test_graph=nx.read_edgelist('./My Drive/Facebook1/data/after_eda/test_pos_after_eda.csv',delimit
    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 are {} %'.format(teY
```

```
Name:
Type: DiGraph
Number of nodes: 1780722
Number of edges: 7550015
Average in degree:   4.2399
Average out degree:   4.2399
Name:
Type: DiGraph
Number of nodes: 1144623
Number of edges: 1887504
Average in degree:   1.6490
Average out degree:   1.6490
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 7.1200735962845
```

we have a cold start problem here

```
#final train and test data sets
if (not os.path.isfile('./My Drive/Facebook1/data/after_eda/train_after_eda.csv')) and \
(not os.path.isfile('./My Drive/Facebook1/data/after_eda/test_after_eda.csv')) and \
(not os.path.isfile('./My Drive/Facebook1/data/train_y.csv')) and \
(not os.path.isfile('./My Drive/Facebook1/data/test_y.csv')) and \
(os.path.isfile('./My Drive/Facebook1/data/after_eda/train_pos_after_eda.csv')) and \
(os.path.isfile('./My Drive/Facebook1/data/after_eda/test_pos_after_eda.csv')) and \
(os.path.isfile('./My Drive/Facebook1/data/after_eda/train_neg_after_eda.csv')) and \
(os.path.isfile('./My Drive/Facebook1/data/after_eda/test_neg_after_eda.csv')):

    X_train_pos = pd.read_csv('./My Drive/Facebook1/data/after_eda/train_pos_after_eda.csv', names=[
    X_test_pos = pd.read_csv('./My Drive/Facebook1/data/after_eda/test_pos_after_eda.csv', names=['s
    X_train_neg = pd.read_csv('./My Drive/Facebook1/data/after_eda/train_neg_after_eda.csv', names=[
    X_test_neg = pd.read_csv('./My Drive/Facebook1/data/after_eda/test_neg_after_eda.csv', names=['s

    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('./My Drive/Facebook1/data/after_eda/train_after_eda.csv',header=False,index=Fals
    X_test.to_csv('./My Drive/Facebook1/data/after_eda/test_after_eda.csv',header=False,index=False)
    pd.DataFrame(y_train.astype(int)).to_csv('./My Drive/Facebook1/data/train_y.csv',header=False,in
    pd.DataFrame(y_test.astype(int)).to_csv('./My Drive/Facebook1/data/test_y.csv',header=False,inde
```

```
X_train = pd.read_csv('./My Drive/Facebook1/data/after_eda/train_after_eda.csv', names=['source_node
X_test = pd.read_csv('./My Drive/Facebook1/data/after_eda/test_after_eda.csv', names=['source_node',
y_train = pd.read_csv('./My Drive/Facebook1/data/train_y.csv')
y_test = pd.read_csv('./My Drive/Facebook1/data/test_y.csv')
```

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

```
Data points in train data (15100030, 2)
Data points in test data (3775008, 2)
Shape of traget variable in train (15100029, 1)
Shape of traget variable in test (3775007, 1)
```

Featurization

```
#Importing Libraries
# please do go through this python notebook:
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 aritmetic operations on arrays
# matplotlib: used to plot graphs
import matplotlib
import matplotlib.pylab 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 pandas import HDFStore,DataFrame
from pandas import read_hdf
from scipy.sparse.linalg import svds, eigs
import gc
from tqdm import tqdm
```

1. Reading Data

```
if os.path.isfile('./My Drive/Facebook1/data/after_eda/train_pos_after_eda.csv'):
    train_graph=nx.read_edgelist('./My Drive/Facebook1/data/after_eda/train_pos_after_eda.csv',delim
```

```
    print(nx.info(train_graph))
else:
    print("please run the FB_EDA.ipynb or download the files from drive")
```

```
Name:
Type: DiGraph
Number of nodes: 1780722
Number of edges: 7550015
Average in degree:   4.2399
Average out degree:   4.2399
```

2. Similarity measures

2.1 Jaccard Distance

```
#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.succes
    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(b))) == 0:
            return 0
        sim = (len(set(train_graph.predecessors(a)).intersection(set(train_graph.predecessors(b)))))
                                    (len(set(train_graph.predecessors(a)).union(set(train_graph.predece
        return sim
    except:
        return 0
```

```
print(jaccard_for_followers(273084,470294))
```

```
0.0
```

```
#node 1635354 not in graph
print(jaccard_for_followers(669354,1635354))
```

```
0
```

2.2 Cosine Distance

```
#for followees
def cosine_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)))))/\
                                    (math.sqrt(len(set(train_graph.successors(a))))*len((set(train_gr
        return sim
    except:
        return 0
```

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

```
0.0
```

```
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))))*(len(set(trai
        return sim
    except:
        return 0
```

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

```
0.02886751345948129
```

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

```
0
```

Ranking Measures

https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.algorithms.link_analysis.p

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
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
85% likelihood of choosing a random link from the page they are currently visiting, and a 15% likelihood of jumping t
they will reach Page E 8.1% of the time. **(The 15% likelihood of jumping to an arbitrary page corresponds to a damp
surfers would eventually end up on Pages A, B, or C, and all other pages would have PageRank zero. In the presen
pages in the web, even though it has no outgoing links of its own.**

3.1 Page Ranking

```
if not os.path.isfile('./My Drive/Facebook1/data/fea_sample/page_rank.p'):
    pr = nx.pagerank(train_graph, alpha=0.85)
    pickle.dump(pr,open('./My Drive/Facebook1/data/fea_sample/page_rank.p','wb'))
else:
    pr = pickle.load(open('./My Drive/Facebook1/data/fea_sample/page_rank.p','rb'))
```

```
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 2.7098251341935827e-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

4.1 Shortest path:

Getting Shortest path between twoo nodes, if nodes have direct path i.e directly connected then we are removing tha

```
#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
```

4.2 Checking for same community

```
#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.successors(b))))
        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 persion 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://www.geeksforgeeks.org/katz-centrality-centrality-measure/ Katz centrality computes the centrality for a nod 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('./My Drive/Facebook1/data/fea_sample/katz.p'):
    katz = nx.katz.katz_centrality(train_graph,alpha=0.005,beta=1)
    pickle.dump(katz,open('./My Drive/Facebook1/data/fea_sample/katz.p','wb'))
else:
    katz = pickle.load(open('./My Drive/Facebook1/data/fea_sample/katz.p','rb'))
```

```
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 0.0007313532484065916
max 0.003394554981699122
mean 0.0007483800935562018
```

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

```
0.0007483800935562018
```

4.6 Hits Score

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

https://en.wikipedia.org/wiki/HITS_algorithm

```
if not os.path.isfile('./My Drive/Facebook1/data/fea_sample/hits.p'):
    hits = nx.hits(train_graph, max_iter=100, tol=1e-08, nstart=None, normalized=True)
    pickle.dump(hits,open('./My Drive/Facebook1/data/fea_sample/hits.p','wb'))
else:
    hits = pickle.load(open('./My Drive/Facebook1/data/fea_sample/hits.p','rb'))
```

```
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 0.0
max 0.004868653378780953
mean 5.615699699344123e-07
```

5. Featurization

5. 1 Reading a sample of Data from both train and test

```
import random
if os.path.isfile('./My Drive/Facebook1/data/after_eda/train_after_eda.csv'):
    filename = "./My Drive/Facebook1/data/after_eda/train_after_eda.csv"
    # 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 (excludes header)
    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('./My Drive/Facebook1/data/after_eda/train_after_eda.csv'):
    filename = "./My Drive/Facebook1/data/after_eda/test_after_eda.csv"
    # 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 (excludes header)
    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
```

```
df_final_train = pd.read_csv('./My Drive/Facebook1/data/after_eda/train_after_eda.csv', skiprows=ski
df_final_train['indicator_link'] = pd.read_csv('./My Drive/Facebook1/data/train_y.csv', skiprows=ski
print("Our train matrix size ",df_final_train.shape)
df_final_train.head(2)
```

```
Our train matrix size  (100002, 3)
```

	source_node	destination_node	indicator_link
0	273084	1505602	1
1	1814022	1791177	1

```
df_final_test = pd.read_csv('./My Drive/Facebook1/data/after_eda/test_after_eda.csv', skiprows=skip_
df_final_test['indicator_link'] = pd.read_csv('./My Drive/Facebook1/data/test_y.csv', skiprows=skip_
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	169499	1465659	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('./My Drive/Facebook1/data/fea_sample/storage_sample_stage1.h5'):
    #mapping jaccrd followers to train and test data
    df_final_train['jaccard_followers'] = df_final_train.apply(lambda row:
                                            jaccard_for_followers(row['source_node'],row['destinatio
    df_final_test['jaccard_followers'] = df_final_test.apply(lambda row:
                                            jaccard_for_followers(row['source_node'],row['destinatio

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


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

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

```
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, inter_followers, inte


if not os.path.isfile('./My Drive/Facebook1/data/fea_sample/storage_sample_stage1.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_stage1(df

    df_final_test['num_followers_s'], df_final_test['num_followers_d'], \
    df_final_test['num_followees_s'], df_final_test['num_followees_d'], \
    df_final_test['inter_followers'], df_final_test['inter_followees']= compute_features_stage1(df_f

    hdf = HDFStore('./My Drive/Facebook1/data/fea_sample/storage_sample_stage1.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('./My Drive/Facebook1/data/fea_sample/storage_sample_stage1.h5', 'trai
    df_final_test = read_hdf('./My Drive/Facebook1/data/fea_sample/storage_sample_stage1.h5', 'test_


df_final_train.head(2)
```

	indicator_link	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees
	1	0	0.000000	0.000000	0.000000
	1	0	0.187135	0.028382	0.343828

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('./My Drive/Facebook1/data/fea_sample/storage_sample_stage2.h5'):
    #mapping adar index on train
    df_final_train['adar_index'] = df_final_train.apply(lambda row: calc_adar_in(row['source_node'],
    #mapping adar index on test
    df_final_test['adar_index'] = df_final_test.apply(lambda row: calc_adar_in(row['source_node'],ro

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

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

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

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

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

    hdf = HDFStore('./My Drive/Facebook1/data/fea_sample/storage_sample_stage2.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('./My Drive/Facebook1/data/fea_sample/storage_sample_stage2.h5', 'trai
    df_final_test = read_hdf('./My Drive/Facebook1/data/fea_sample/storage_sample_stage2.h5', 'test_
```

```
df_final_train.head(2)
```

:s	cosine_followers	cosine_followees	num_followers_s	num_followees_s	num_followees_d
)0	0.000000	0.000000	11	15	8
35	0.028382	0.343828	17	61	142

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 deci consider one million people following a celebrity on a social network then chances are most of them never met each user has 30 contacts in his/her social network, the chances are higher that many of them know each other. `credit` · 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

```
#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 [00:17<00:00, 103168.10it/s]
```

```
if not os.path.isfile('./My Drive/Facebook1/data/fea_sample/storage_sample_stage3.h5'):
    #mapping to pandas train
    df_final_train['weight_in'] = df_final_train.destination_node.apply(lambda x: Weight_in.get(x,me
    df_final_train['weight_out'] = df_final_train.source_node.apply(lambda x: Weight_out.get(x,mean_

    #mapping to pandas test
    df_final_test['weight_in'] = df_final_test.destination_node.apply(lambda x: Weight_in.get(x,mean
    df_final_test['weight_out'] = df_final_test.source_node.apply(lambda x: Weight_out.get(x,mean_we
```

```
    #some features engineerings on the in and out weights
    df_final_train['weight_f1'] = df_final_train.weight_in + df_final_train.weight_out
    df_final_train['weight_f2'] = df_final_train.weight_in * df_final_train.weight_out
    df_final_train['weight_f3'] = (2*df_final_train.weight_in + 1*df_final_train.weight_out)
    df_final_train['weight_f4'] = (1*df_final_train.weight_in + 2*df_final_train.weight_out)

    #some features engineerings on the in and out weights
    df_final_test['weight_f1'] = df_final_test.weight_in + df_final_test.weight_out
    df_final_test['weight_f2'] = df_final_test.weight_in * df_final_test.weight_out
    df_final_test['weight_f3'] = (2*df_final_test.weight_in + 1*df_final_test.weight_out)
    df_final_test['weight_f4'] = (1*df_final_test.weight_in + 2*df_final_test.weight_out)


if not os.path.isfile('./My Drive/Facebook1/data/fea_sample/storage_sample_stage3.h5'):

    #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.get(x,mean_pr))
    df_final_train['page_rank_d'] = df_final_train.destination_node.apply(lambda x:pr.get(x,mean_pr)

    df_final_test['page_rank_s'] = df_final_test.source_node.apply(lambda x:pr.get(x,mean_pr))
    df_final_test['page_rank_d'] = df_final_test.destination_node.apply(lambda x:pr.get(x,mean_pr))
    #========================================================================

    #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.get(x,mean_katz))
    df_final_train['katz_d'] = df_final_train.destination_node.apply(lambda x: katz.get(x,mean_katz)

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

    #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].get(x,0))
    df_final_train['hubs_d'] = df_final_train.destination_node.apply(lambda x: hits[0].get(x,0))

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

    #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: hits[1].get(x,0))
    df_final_train['authorities_d'] = df_final_train.destination_node.apply(lambda x: hits[1].get(x,

    df_final_test['authorities_s'] = df_final_test.source_node.apply(lambda x: hits[1].get(x,0))
    df_final_test['authorities_d'] = df_final_test.destination_node.apply(lambda x: hits[1].get(x,0)
    #========================================================================

    hdf = HDFStore('./My Drive/Facebook1/data/fea_sample/storage_sample_stage3.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('./My Drive/Facebook1/data/fea_sample/storage_sample_stage3.h5', 'trai
    df_final_test = read_hdf('./My Drive/Facebook1/data/fea_sample/storage_sample_stage3.h5', 'test_


df_final_train.head(2)
```

ck	same_comp	shortest_path	weight_in	weight_out	weight_f1	weight_f2	weight_f3	weig
0	1	4	0.377964	0.250	0.627964	0.094491	1.005929	0.8
0	1	2	0.102598	0.127	0.229598	0.013030	0.332196	0.3

```
followers_s=np.array(df_final_train['num_followers_s'])
followers_d=np.array(df_final_train['num_followers_d'])
preferential_followers=[]
for i in range(len(followers_s)):
    preferential_followers.append(followers_s[i]*followers_d[i])
df_final_train['prefer_Attach_followers']= preferential_followers
df_final_train.head(2)
```

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees
0	273084	1505602	1	0	0.000000
1	832016	1543415	1	0	0.187135

```
followees_s=np.array(df_final_train['num_followees_s'])
followees_d=np.array(df_final_train['num_followees_d'])
preferential_followees=[]
for i in range(len(followees_s)):
    preferential_followees.append(followees_s[i]*followees_d[i])
df_final_train['prefer_Attach_followees']= preferential_followees
df_final_train.head(2)
```

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees
0	273084	1505602	1	0	0.000000
1	832016	1543415	1	0	0.187135

```
followers_s=np.array(df_final_test['num_followers_s'])
followers_d=np.array(df_final_test['num_followers_d'])
preferential_followers=[]
for i in range(len(followers_s)):
    preferential_followers.append(followers_s[i]*followers_d[i])
df_final_test['prefer_Attach_followers']= preferential_followers
df_final_test.head(2)
```

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees
0	848424	784690	1	0	0.0
1	483294	1255532	1	0	0.0

```
followees_s=np.array(df_final_test['num_followees_s'])
followees_d=np.array(df_final_test['num_followees_d'])
preferential_followees=[]
for i in range(len(followees_s)):
    preferential_followees.append(followees_s[i]*followees_d[i])
df_final_test['prefer_Attach_followees']= preferential_followees
df_final_test.head(2)
```

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees
0	848424	784690	1	0	0.0
1	483294	1255532	1	0	0.0

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 inedx in svd vector
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())).asfptype()
```

```
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('./My Drive/Facebook1/data/fea_sample/storage_sample_stage4.h5'):
    #================================================================================
    df_final_train[['svd_u_s_1', 'svd_u_s_2','svd_u_s_3', 'svd_u_s_4', 'svd_u_s_5', 'svd_u_s_6']] =
    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_5','svd_u_d_6']] =
    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_5', 'svd_v_s_6',]] =
    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_5','svd_v_d_6']] =
    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', 'svd_u_s_6']] = \
    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_5','svd_u_d_6']] = \
    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', 'svd_v_s_6',]] =
    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_5','svd_v_d_6']] = \
    df_final_test.destination_node.apply(lambda x: svd(x, V.T)).apply(pd.Series)
    #================================================================================

    hdf = HDFStore('./My Drive/Facebook1/data/fea_sample/storage_sample_stage4.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()
```

```
df_final_train.head(2)
```

ıch_followers	prefer_Attach_followees	svd_u_s_1	svd_u_s_2	svd_u_s_3	svd_u_s_4	svd
66	120	-1.666333e-13	4.613822e-13	1.043041e-05	6.678030e-13	2.45
1598	8662	7.050643e-13	-8.250578e-11	-1.717841e-10	3.705016e-02	1.03

```
#for train datasets
s1,s2,s3,s4,s5,s6=df_final_train['svd_u_s_1'],df_final_train['svd_u_s_2'],df_final_train['svd_u_s_3'
s7,s8,s9,s10,s11,s12=df_final_train['svd_v_s_1'],df_final_train['svd_v_s_2'],df_final_train['svd_v_s

d1,d2,d3,d4,d5,d6=df_final_train['svd_u_d_1'],df_final_train['svd_u_d_2'],df_final_train['svd_u_d_3'
d7,d8,d9,d10,d11,d12=df_final_train['svd_v_d_1'],df_final_train['svd_v_d_2'],df_final_train['svd_v_d
```

```
svd_dot_u=[]
svd_dot_v=[]
for i in range(len(np.array(s1))):
    a=[]
    b=[]
    c=[]
    d=[]
    a.append(np.array(s1[i]))
    a.append(np.array(s2[i]))
    a.append(np.array(s3[i]))
    a.append(np.array(s4[i]))
    a.append(np.array(s5[i]))
    a.append(np.array(s6[i]))
    c.append(np.array(s7[i]))
    c.append(np.array(s8[i]))
    c.append(np.array(s9[i]))
    c.append(np.array(s10[i]))
    c.append(np.array(s11[i]))
    c.append(np.array(s12[i]))
    b.append(np.array(d1[i]))
    b.append(np.array(d2[i]))
    b.append(np.array(d3[i]))
    b.append(np.array(d4[i]))
    b.append(np.array(d5[i]))
    b.append(np.array(d6[i]))
    d.append(np.array(d7[i]))
    d.append(np.array(d8[i]))
    d.append(np.array(d9[i]))
    d.append(np.array(d10[i]))
    d.append(np.array(d11[i]))
    d.append(np.array(d12[i]))
    svd_dot_u.append(np.dot(a,b))
    svd_dot_v.append(np.dot(c,d))
df_final_train['svd_dot_u']=svd_dot_u
df_final_train['svd_dot_v']=svd_dot_v
```

```
df_final_train.head(2)
```

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees
0	273084	1505602	1	0	0.000000
1	832016	1543415	1	0	0.187135

```
#for test dataset
s1,s2,s3,s4,s5,s6=df_final_test['svd_u_s_1'],df_final_test['svd_u_s_2'],df_final_test['svd_u_s_3'],d
s7,s8,s9,s10,s11,s12=df_final_test['svd_v_s_1'],df_final_test['svd_v_s_2'],df_final_test['svd_v_s_3'

d1,d2,d3,d4,d5,d6=df_final_test['svd_u_d_1'],df_final_test['svd_u_d_2'],df_final_test['svd_u_d_3'],d
d7,d8,d9,d10,d11,d12=df_final_test['svd_v_d_1'],df_final_test['svd_v_d_2'],df_final_test['svd_v_d_3'
```

```
svd_dot_u=[]
svd_dot_v=[]
for i in range(len(np.array(s1))):
    a=[]
    b=[]
    c=[]
```

```
        d=[]
        a.append(np.array(s1[i]))
        a.append(np.array(s2[i]))
        a.append(np.array(s3[i]))
        a.append(np.array(s4[i]))
        a.append(np.array(s5[i]))
        a.append(np.array(s6[i]))
        c.append(np.array(s7[i]))
        c.append(np.array(s8[i]))
        c.append(np.array(s9[i]))
        c.append(np.array(s10[i]))
        c.append(np.array(s11[i]))
        c.append(np.array(s12[i]))
        b.append(np.array(d1[i]))
        b.append(np.array(d2[i]))
        b.append(np.array(d3[i]))
        b.append(np.array(d4[i]))
        b.append(np.array(d5[i]))
        b.append(np.array(d6[i]))
        d.append(np.array(d7[i]))
        d.append(np.array(d8[i]))
        d.append(np.array(d9[i]))
        d.append(np.array(d10[i]))
        d.append(np.array(d11[i]))
        d.append(np.array(d12[i]))
        svd_dot_u.append(np.dot(a,b))
        svd_dot_v.append(np.dot(c,d))
    df_final_test['svd_dot_u']=svd_dot_u
    df_final_test['svd_dot_v']=svd_dot_v
```

```
df_final_test.head(2)
```

_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
e-10	1.166048e-13	2.253356e-11	3.220558e-15	-2.148852e-13	1.883259e-13	5.904813e-11	2.701538e-12	4.34162(
e-08	1.907404e-12	3.797447e-11	4.992965e-14	-4.054500e-13	2.895772e-13	2.545371e-10	2.248602e-14	3.60101(

```
hdf = HDFStore('./My Drive/Facebook1/data/fea_sample/storage_sample_stage5.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()
```

```
#Importing Libraries
# please do go through this python notebook:
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 aritmetic operations on arrays
# matplotlib: used to plot graphs
import matplotlib
import matplotlib.pylab 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 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


#reading
from pandas import read_hdf
X_train = read_hdf('./My Drive/Facebook1/data/fea_sample/storage_sample_stage5.h5', 'train_df',mode=
X_test = read_hdf('./My Drive/Facebook1/data/fea_sample/storage_sample_stage5.h5', 'test_df',mode='r


X_test = X_test[[c for c in X_test if c not in ['prefer_Attach_followers',  'prefer_Attach_followees
       + ['prefer_Attach_followers',  'prefer_Attach_followees']]
X_test.columns
```

```
Index(['jaccard_followers', 'jaccard_followees', 'cosine_followers',
       'cosine_followees', 'num_followers_s', 'num_followees_s',
       'num_followees_d', 'inter_followers', 'inter_followees',
       'num_followers_d', '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', 'svd_dot_u',
       'svd_dot_v', 'prefer_Attach_followers', 'prefer_Attach_followees'],
      dtype='object')
```

```
X_train = X_train[[c for c in X_train if c not in ['prefer_Attach_followers', 'prefer_Attach_followe
       + ['prefer_Attach_followers',  'prefer_Attach_followees']]
X_train.columns
```

```
Index(['jaccard_followers', 'jaccard_followees', 'cosine_followers',
       'cosine_followees', 'num_followers_s', 'num_followees_s',
       'num_followees_d', 'inter_followers', 'inter_followees',
       'num_followers_d', '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', 'svd_dot_u',
       'svd_dot_v', 'prefer_Attach_followers', 'prefer_Attach_followees'],
      dtype='object')
```

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

```
X_train.drop(['source_node', 'destination_node','indicator_link'],axis=1,inplace=True)
X_test.drop(['source_node', 'destination_node','indicator_link'],axis=1,inplace=True)
```

```
X_train.shape
```

```
(100002, 56)
```

```
X_test.shape
```

```
(50002, 56)
```

```
X_train.head(2)
```

	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	num_followe
0	0	0.000000	0.000000	0.000000	
1	0	0.187135	0.028382	0.343828	

```
X_test.head(2)
```

	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	num_followe
0	0	0.0	0.029161	0.0	
1	0	0.0	0.000000	0.0	

```
from sklearn.preprocessing import StandardScaler
for column in X_train.columns:
  scalar = StandardScaler()
  scalar.fit(X_train[column].values.reshape(-1,1))
  print(column + ": ")
  print(f"Mean : {scalar.mean_[0]}, Standard deviation : {np.sqrt(scalar.var_[0])}")
  standardized_train = scalar.transform(X_train[column].values.reshape(-1, 1))
  standardized_test = scalar.transform(X_test[column].values.reshape(-1, 1))
  X_train[column] = pd.DataFrame(standardized_train)
  X_test[column] = pd.DataFrame(standardized_test)
```

```
jaccard_followees:
Mean : 0.04008788823588526, Standard deviation : 0.10414053959329606
cosine_followers:
Mean : 0.02171076978621971, Standard deviation : 0.05304628198970236
cosine_followees:
Mean : 0.06700952576842657, Standard deviation : 0.1531591100734706
num_followers_s:
Mean : 9.832273354532909, Standard deviation : 18.343272545999035
num_followees_s:
Mean : 12.016629667406653, Standard deviation : 26.269752786244528
num_followees_d:
Mean : 9.73007539849203, Standard deviation : 18.445197877654227
inter_followers:
Mean : 1.7824443511129777, Standard deviation : 7.990476308631131
inter_followees:
Mean : 1.7856642867142658, Standard deviation : 8.409164346254286
num_followers_d:
Mean : 11.092068158636827, Standard deviation : 21.844594515770783
adar_index:
Mean : 1.2063680748453576, Standard deviation : 4.54248452851973
follows_back:
Mean : 0.2779244415111698, Standard deviation : 0.44797594390979767
same_comp:
Mean : 0.8551628967420651, Standard deviation : 0.3519365237905625
shortest_path:
Mean : 3.1815363692726146, Standard deviation : 3.906704319595922
weight_in:
Mean : 0.47865244152442815, Standard deviation : 0.24643554357411887
weight_out:
Mean : 0.4815817560100229, Standard deviation : 0.2573517973262048
weight_f1:
Mean : 0.9602341975344509, Standard deviation : 0.4126486059667631
weight_f2:
Mean : 0.25216950694912565, Standard deviation : 0.2069480752210443
weight_f3:
Mean : 1.438886639058879, Standard deviation : 0.629117437923112
weight_f4:
Mean : 1.441815953544474, Standard deviation : 0.6420959134350649
page_rank_s:
Mean : 8.165686116052111e-07, Standard deviation : 8.421881765053727e-07
page_rank_d:
Mean : 8.733031446835729e-07, Standard deviation : 9.004112171359109e-07
katz_s:
Mean : 0.0007756027214001141, Standard deviation : 0.0001084078746825317
katz_d:
Mean : 0.0007816125860544442, Standard deviation : 0.0001264936347845149
hubs_s:
Mean : 8.810298728813284e-06, Standard deviation : 0.00016636910402157155
hubs_d:
Mean : 7.1127812687303475e-06, Standard deviation : 0.0001482718474281663
authorities_s:
Mean : 7.460537970267456e-06, Standard deviation : 0.00015510511100973954
authorities_d:
Mean : 8.488533464229226e-06, Standard deviation : 0.00016908335845234487
svd_u_s_1:
Mean : -7.488580979639513e-05, Standard deviation : 0.0025535336623273507
svd_u_s_2:
Mean : 0.0001287945794861713 2, Standard deviation : 0.0027251138499738608
```

```
Mean : 0.0001287945794861/152, Standard deviation : 0.0027251158499738668
svd_u_s_3:
Mean : 9.851711470910341e-05, Standard deviation : 0.00236850594988815
svd_u_s_4:
Mean : 0.0001171212452032282l, Standard deviation : 0.003210097135647676
svd_u_s_5:
Mean : 0.0001365248807726644 7, Standard deviation : 0.003495486797253162
svd_u_s_6:
Mean : 0.0001625439297680010 3, Standard deviation : 0.003069395124644531
svd_u_d_1:
Mean : -7.70115617785015e-05, Standard deviation : 0.0026066552359584457
svd_u_d_2:
Mean : 9.182921355450136e-05, Standard deviation : 0.0021609705740241497
svd_u_d_3:
Mean : 6.8384928538353e-05, Standard deviation : 0.001898998740593041
svd_u_d_4:
Mean : 7.312580180957063e-05, Standard deviation : 0.0024436210341240933
svd_u_d_5:
Mean : 6.687763030251471e-05, Standard deviation : 0.0022077953997139813
svd_u_d_6:
Mean : 0.0001312259044413958 6, Standard deviation : 0.002735513233027441
svd_v_s_1:
Mean : -7.159273161531357e-05, Standard deviation : 0.0024702106810473 29
svd_v_s_2:
Mean : 8.181855296639534e-05, Standard deviation : 0.0021829360488899966
svd_v_s_3:
Mean : 6.568934661563412e-05, Standard deviation : 0.002421247964736364
svd_v_s_4:
Mean : 9.123722150377589e-05, Standard deviation : 0.00251095576984386
svd_v_s_5:
Mean : 8.386329863653715e-05, Standard deviation : 0.0022218196032326049
svd_v_s_6:
Mean : 0.0001390575283074510 6, Standard deviation : 0.0028910158294473713
svd_v_d_1:
Mean : -7.622936118298057e-05, Standard deviation : 0.0025611969809661676
svd_v_d_2:
Mean : 0.0001551794162074575, Standard deviation : 0.0035775065654940637
svd_v_d_3:
Mean : 0.0001478494542602571 5, Standard deviation : 0.003831898010575655
svd_v_d_4:
Mean : 9.529309112357338e-05, Standard deviation : 0.0025854396886112554
svd_v_d_5:
Mean : 0.000102932525737395, Standard deviation : 0.0025348458742164783
svd_v_d_6:
Mean : 0.0001582184136292250 6, Standard deviation : 0.003151557434823596
svd_dot_u:
Mean : 3.0358595354938315e-05, Standard deviation : 0.0004103143278709034
svd_dot_v:
Mean : 3.0221273865876985e-05, Standard deviation : 0.00040496977645611504
prefer_Attach_followers:
Mean : 308.61588768224635, Standard deviation : 1835.1736291887453
prefer_Attach_followees:
Mean : 333.1835763284734, Standard deviation : 1907.7012485758826
```

```
X_train.head(2)
```

	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	num_followe
0	0.0	-0.384940	-0.409280	-0.437516	0.06
1	0.0	1.412002	0.125759	1.807390	0.39

```
X_test.head(2)
```

	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	num_followe
0	0.0	-0.38494	0.14044	-0.437516	-0.20
1	0.0	-0.38494	-0.40928	-0.437516	-0.42

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os

from collections import Counter
```

```
from xgboost import XGBClassifier
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV

parameters = { 'n_estimators' : [100, 150, 200, 300, 500, 1000], 'max_depth' : [2, 3, 4, 5, 6, 7, 8,
train_auc = []
cv_auc = []

xgb = XGBClassifier(class_weight = 'balanced')
clf = GridSearchCV(xgb, parameters, scoring='roc_auc', return_train_score = True, n_jobs = -1, verbo
clf.fit(X_train, y_train)
```

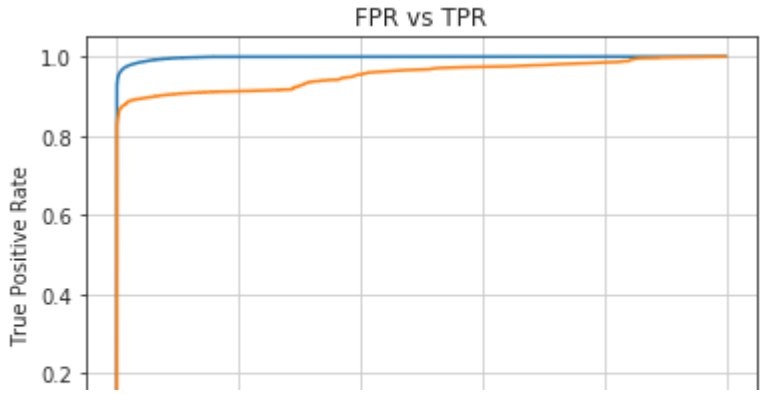
```
clf.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 54 candidates, totalling 162 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed:  1.4min
[Parallel(n_jobs=-1)]: Done  64 tasks      | elapsed: 19.9min
[Parallel(n_jobs=-1)]: Done 154 tasks      | elapsed: 83.0min
[Parallel(n_jobs=-1)]: Done 162 out of 162 | elapsed: 94.6min finished
GridSearchCV(cv='warn', error_score='raise-deprecating',
             estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                     class_weight='balanced',
                                     colsample_bylevel=1, colsample_bynode=1,
                                     colsample_bytree=1, gamma=0,
                                     learning_rate=0.1, max_delta_step=0,
                                     max_depth=3, min_child_weight=1,
                                     missing=None, n_estimators=100, n_jobs=1,
                                     nthread=None, objective='binary:logistic',
                                     random_state=0, reg_alpha=0, reg_lambda=1,
                                     scale_pos_weight=1, seed=None, silent=None,
                                     subsample=1, verbosity=1),
             iid='warn', n_jobs=-1,
             param_grid={'max_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10],
                         'n_estimators': [100, 150, 200, 300, 500, 1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=5)
```

```
print(clf.best_params_)
best_depth = clf.best_params_['max_depth']
best_estimators = clf.best_params_['n_estimators']
```

```
{'max_depth': 10, 'n_estimators': 500}
```

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve

xgb = XGBClassifier(max_depth = best_depth, min_samples_split = best_estimators )
xgb.fit(X_train, y_train)

y_train_pred = xgb.predict_proba( X_train)[:, 1]
y_test_pred = xgb.predict_proba(X_test)[:, 1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("FPR vs TPR")
plt.grid()
plt.show()
```



Double-click (or enter) to edit

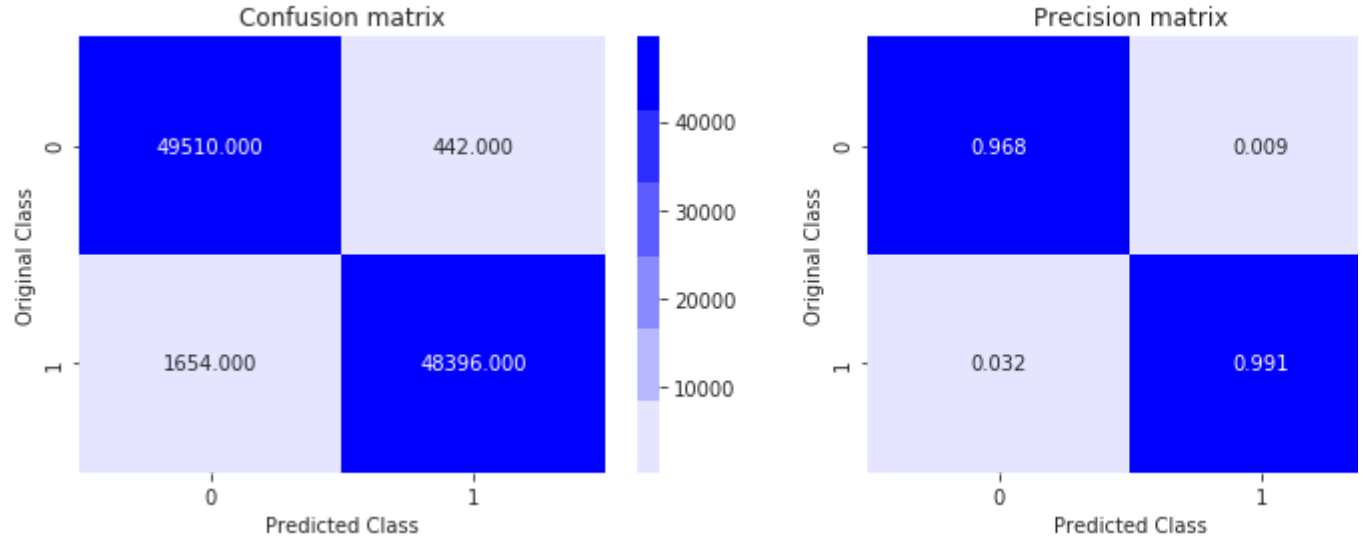
```
from sklearn.metrics import f1_score
print('Train f1 score',f1_score(y_train,y_train_pred.round()))
print('Test f1 score',f1_score(y_test,y_test_pred.round()))
```

```
Train f1 score 0.9788043038589112
Test f1 score 0.9286456015427977
```

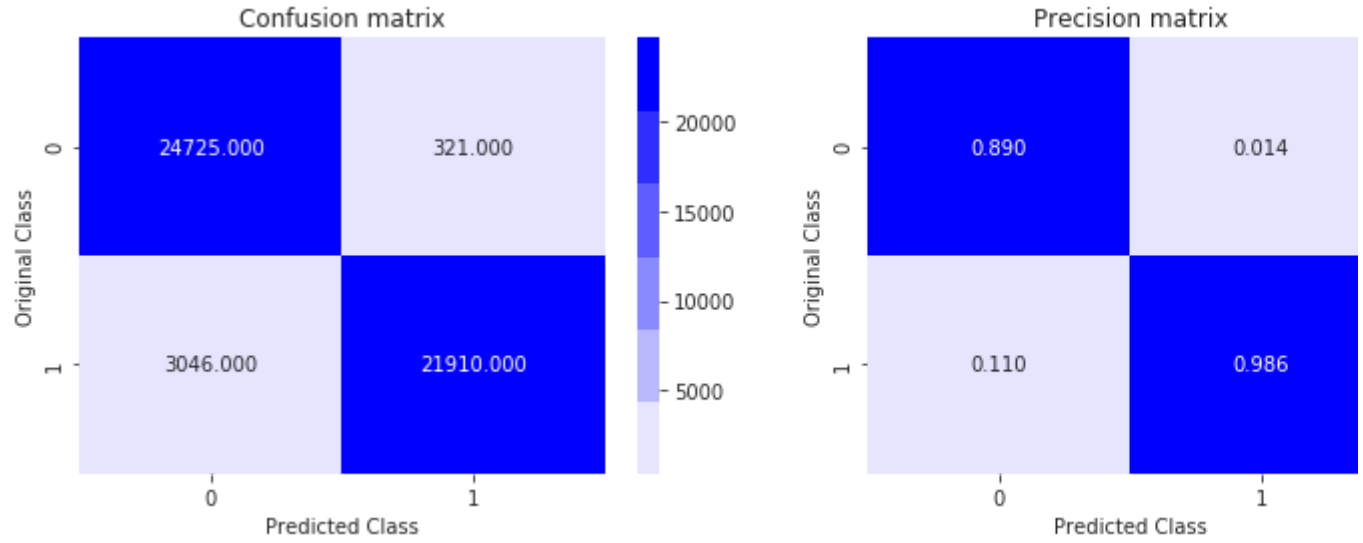
```
from sklearn.metrics import confusion_matrix
def plot_confusion_matrix(y_test, predict_y):
    C = confusion_matrix(y_test, predict_y)

    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, yticklabels=labels)
    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, yticklabels=labels)
    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, yticklabels=labels)
    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.round())
print('Test confusion_matrix')
plot_confusion_matrix(y_test,y_test_pred.round())
```

```
Train confusion_matrix
```




```
Test confusion_matrix
```



SUMMARY

Train confusion matrix

Here the Recall(TPR) = 0.967 i.e., out of total positive points 96.7% are correctly predicted.

Recall = TP/(TP + FN)

Precision = 0.991 i.e., out of total points for which the model predicted as positive class 99.1% are true.
Precision = TP/(TP + FP)

TNR = 0.968 which seems the model is performing well for the negative points.

Test confusion matrix

Here the Recall(TPR) = 0.878 i.e., out of total positive points 87.8% are correctly predicted.

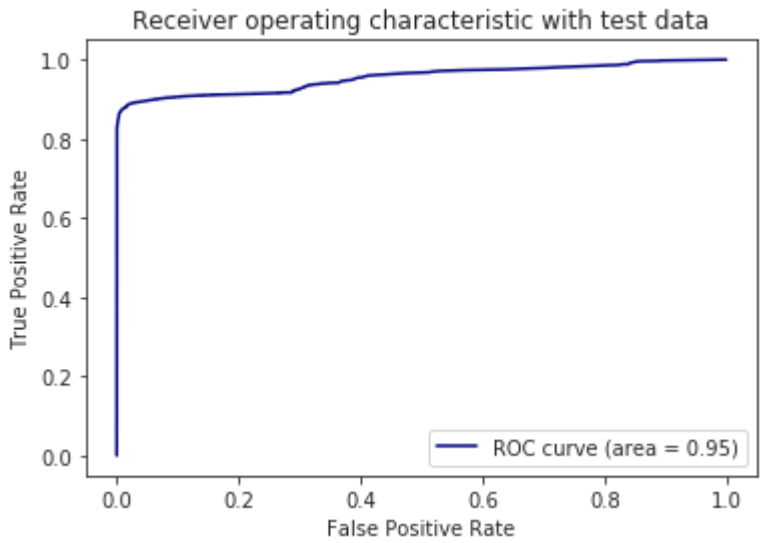
Recall = TP/(TP + FN)

Precision = 0.986 i.e., out of total points for which the model predicted as positive class 98.6% are true.

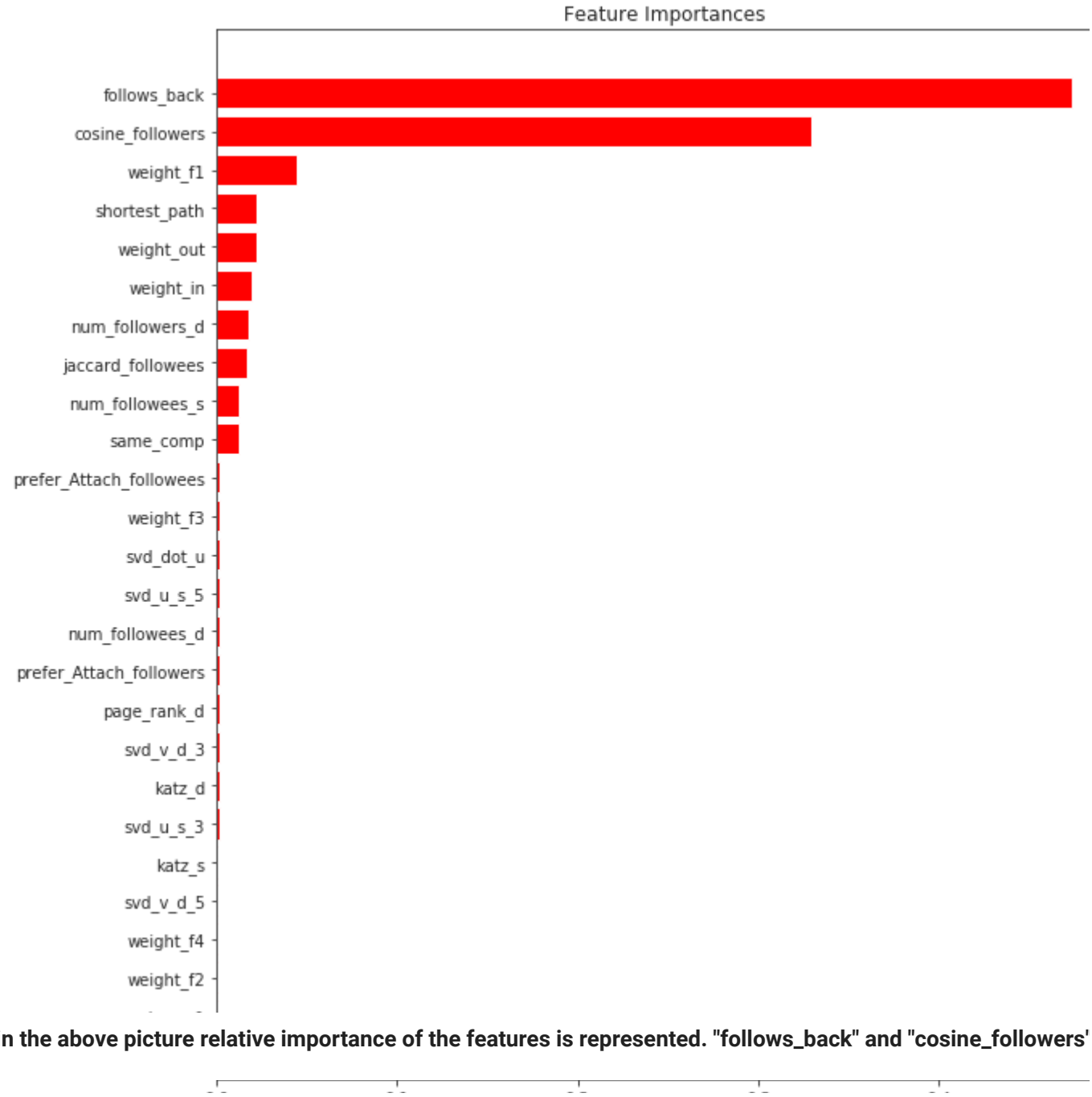
Precision = TP/(TP + FP)

TNR = 0.890 which seems the model is performing well for the negative points.

```
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 = X_train.columns
importances = xgb.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()
```



Here in the above picture relative importance of the features is represented. "follows_back" and "cosine_followers'