Social network Graph Link Prediction

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 features like no of followers, is he followed back, page rank, katz score, adar index, some svd fetures of adj matrix, some weight features etc. and trained ml model based on these features to predict link.
- 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.pdf
  - 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

```
#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 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
from google.colab import drive
drive.mount('/data')
```

```
Drive already mounted at /data; to attempt to forcibly remount, call drive.mount("/data'
```

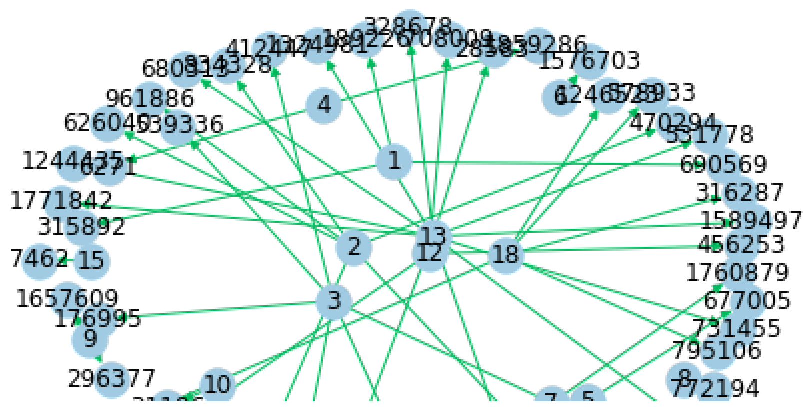
```
#reading graph
if not os.path.isfile('/data/My Drive/data/after_eda/train_woheader.csv'):
    traincsv = pd.read_csv('/data/My Drive/data/train.csv')
    print(traincsv[traincsv.isna().any(1)])
    print(traincsv.info())
    print("Number of diplicate entries: ",sum(traincsv.duplicated()))
    traincsv.to_csv('/data/My Drive/data/after_eda/train_woheader.csv',header=False,index=Fal
    print("saved the graph into file")
else:
    g=nx.read_edgelist('/data/My Drive/data/after_eda/train_woheader.csv',delimiter=',',creat
    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('/data/My Drive/data/train.csv', nrows=50).to_csv('train_woheader_sample.csv'

subgraph=nx.read_edgelist('train_woheader_sample.csv',delimiter=',',create_using=nx.DiGraph()
# https://stackoverflow.com/questions/9402255/drawing-a-huge-graph-with-networkx-and-matplotl

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

```
DiGraph with 66 nodes and 50 edges
```



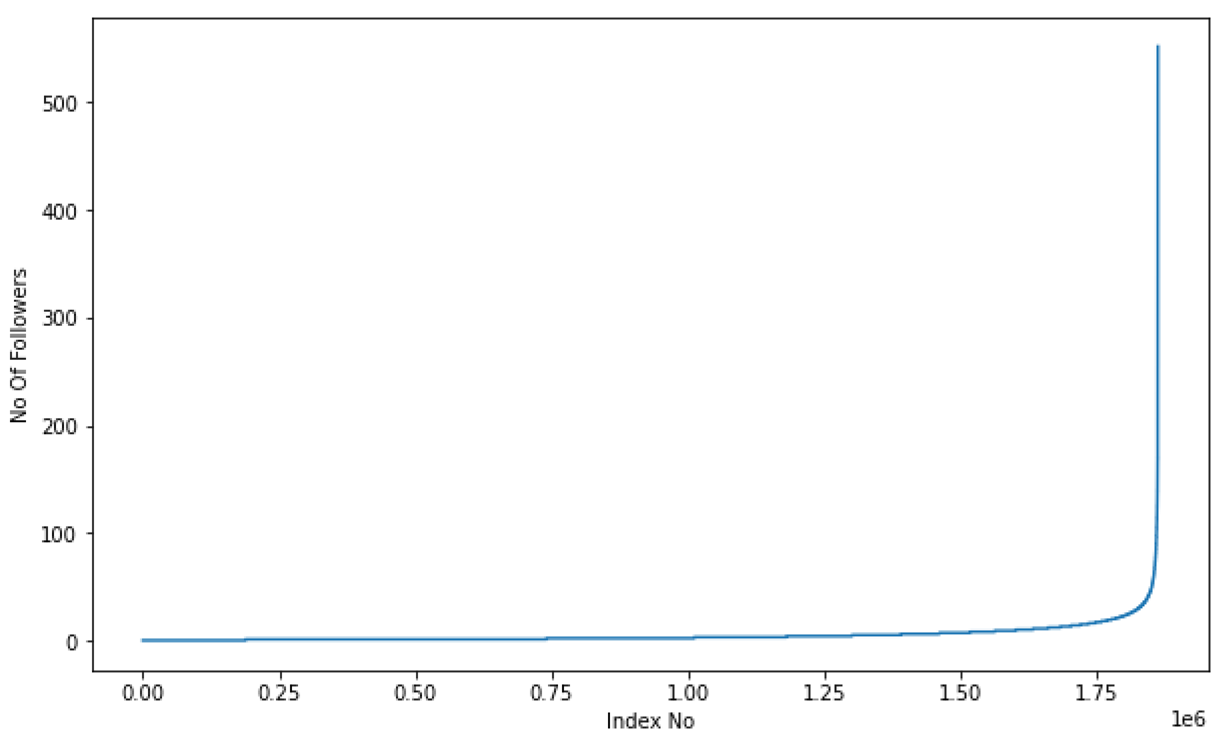
1. Exploratory Data Analysis

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

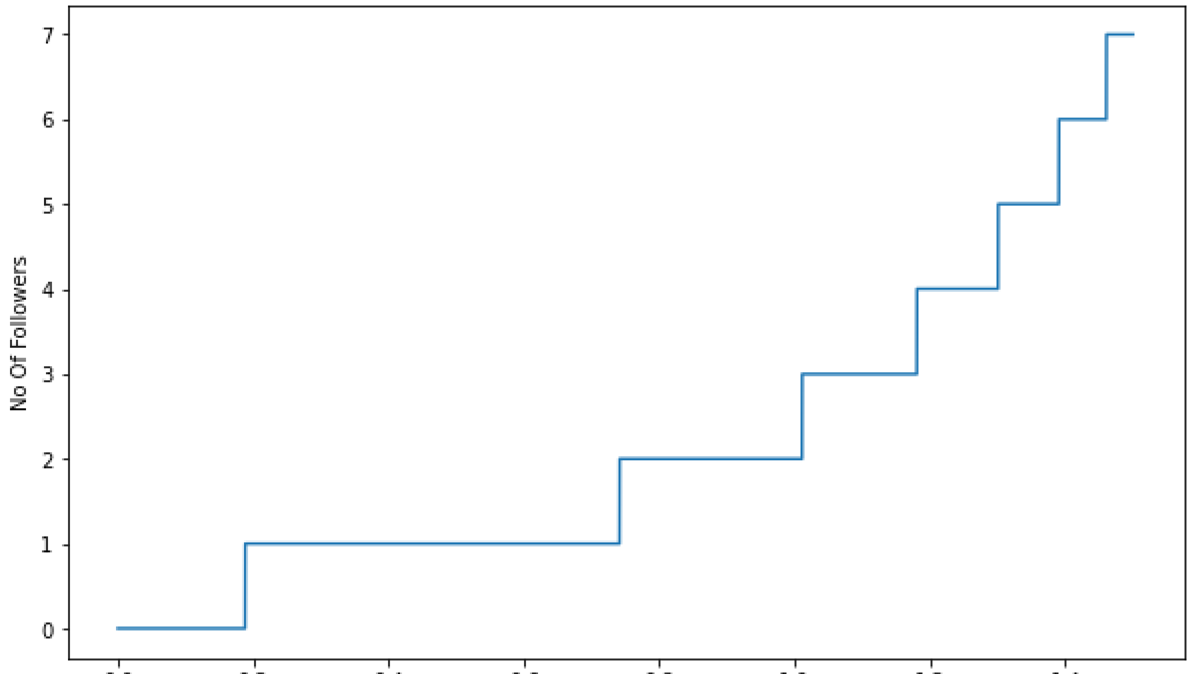
```
The number of unique persons 1862220
```

1.1 No of followers for each person

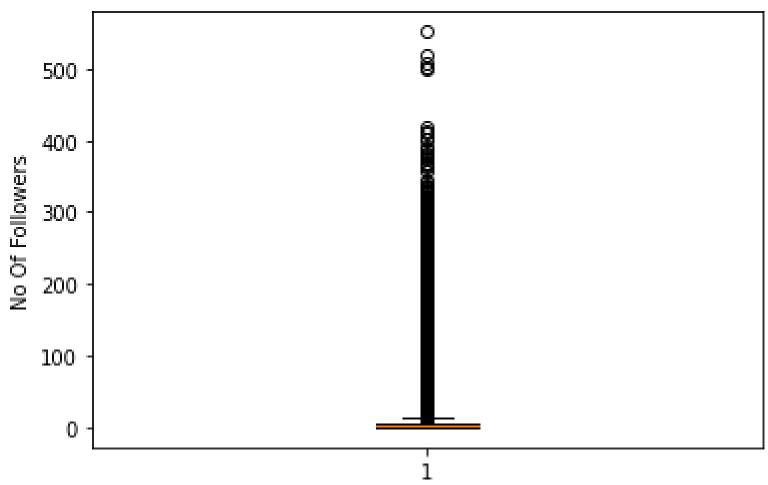
```
indegree_dist = list(dict(g.in_degree()).values())
indegree_dist.sort()
plt.figure(figsize=(10,6))
plt.plot(indegree_dist)
plt.xlabel('Index No')
plt.ylabel('No Of Followers')
plt.show()
```




```
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()
```



```
### 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
```

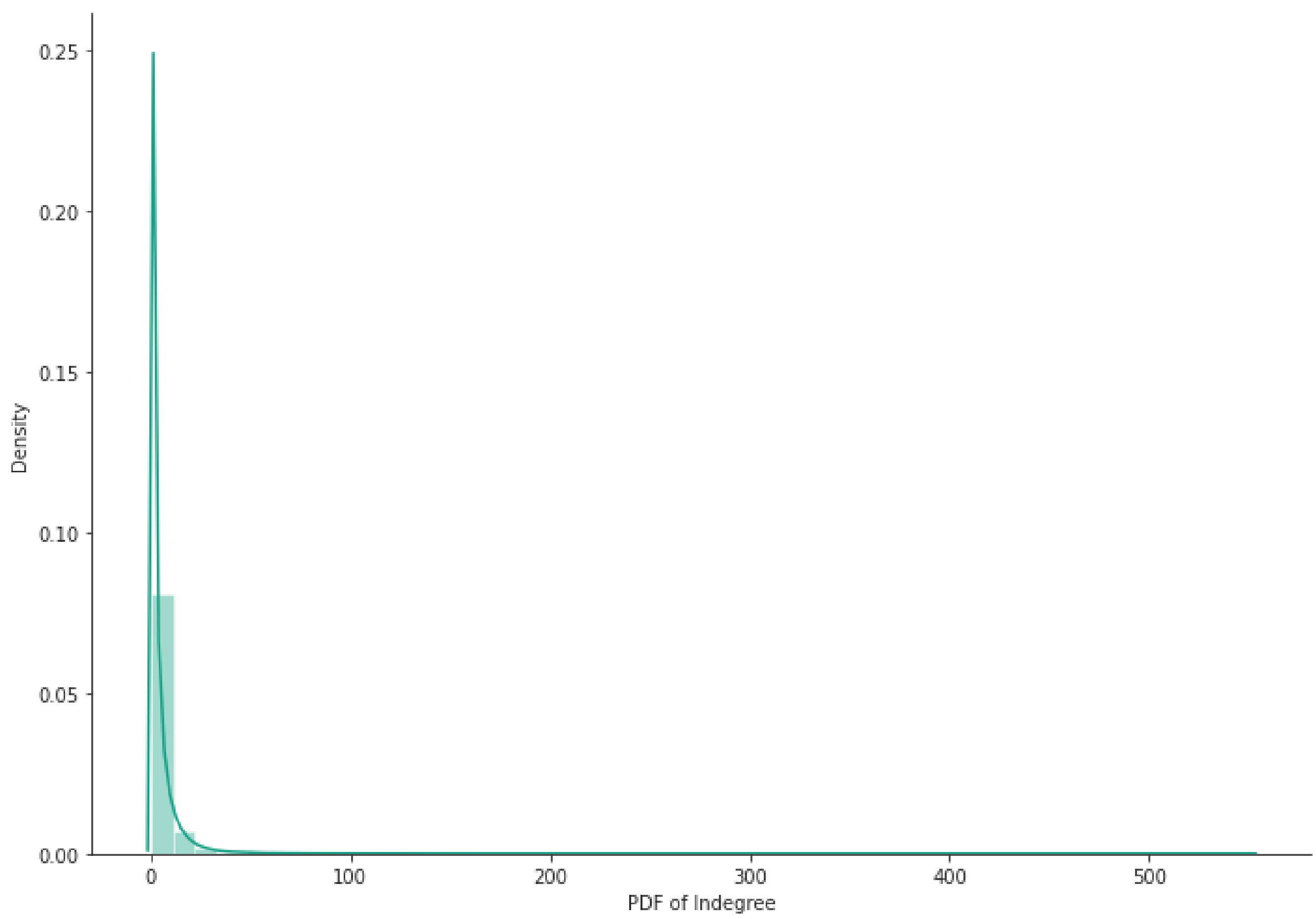
99% of data having followers of 40 only.

```
### 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
```

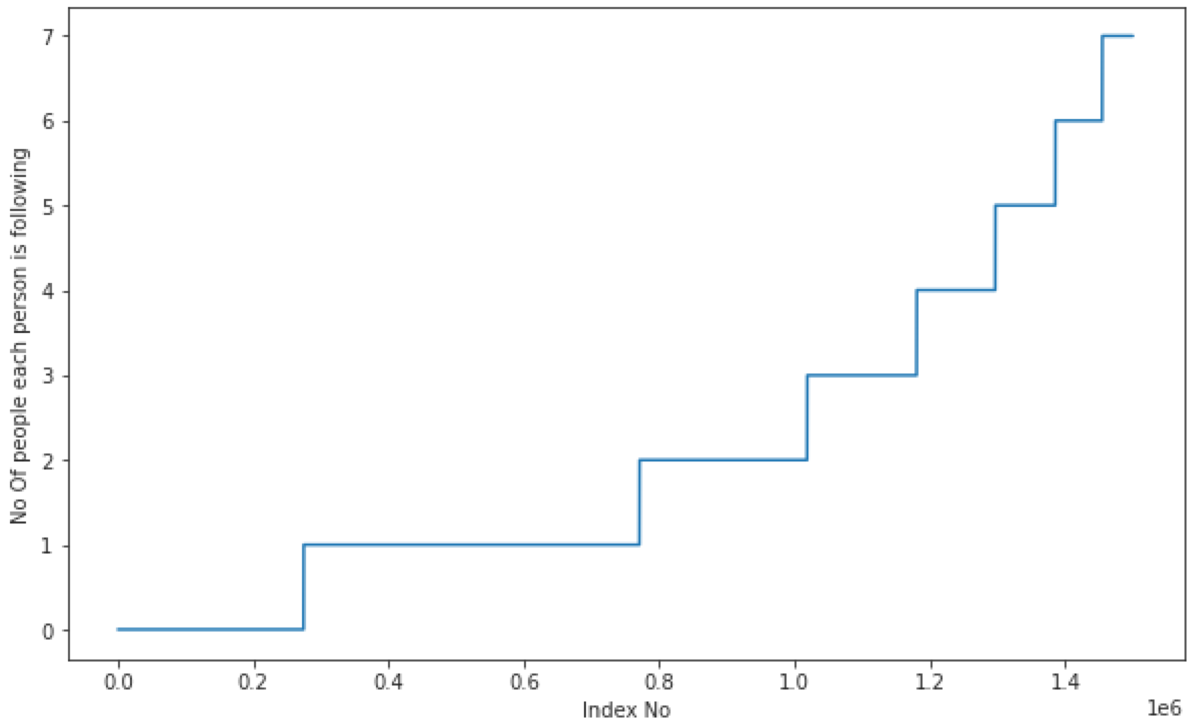
```
%matplotlib inline
sns.set_style('ticks')
fig, ax = plt.subplots()
fig.set_size_inches(11.7, 8.27)
sns.distplot(indegree_dist, color='#16A085')
plt.xlabel('PDF of Indegree')
sns.despine()
#plt.show()
```



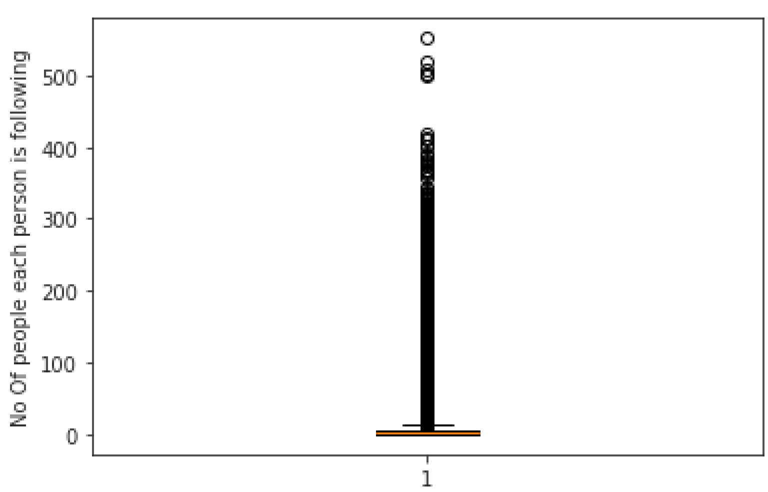
1.2 No of people each person is following

```
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))
```

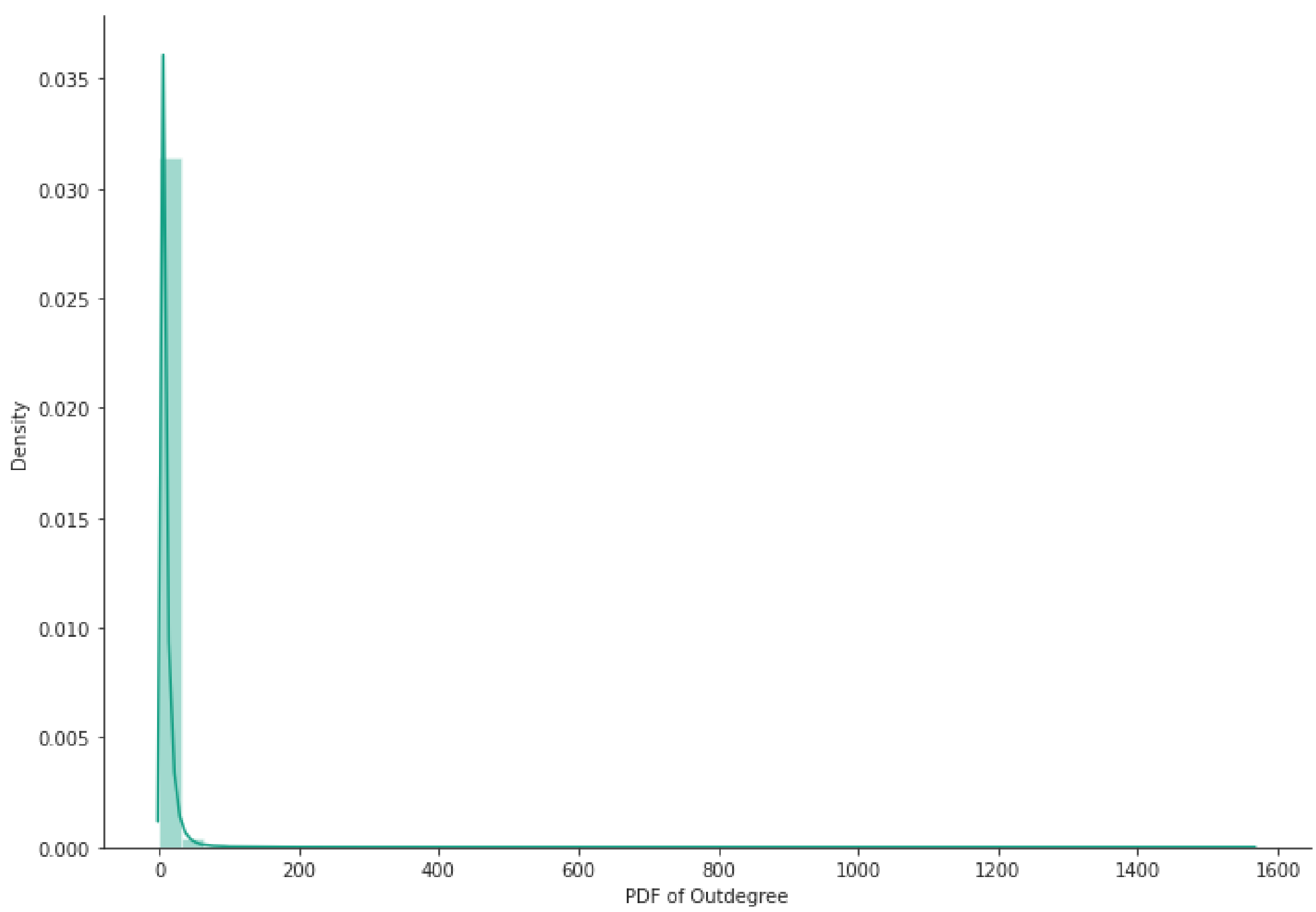
```
90 percentile value is 12.0
91 percentile value is 13.0
92 percentile value is 14.0
93 percentile value is 15.0
94 percentile value is 17.0
95 percentile value is 19.0
96 percentile value is 21.0
```

```
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),'a
                                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
```

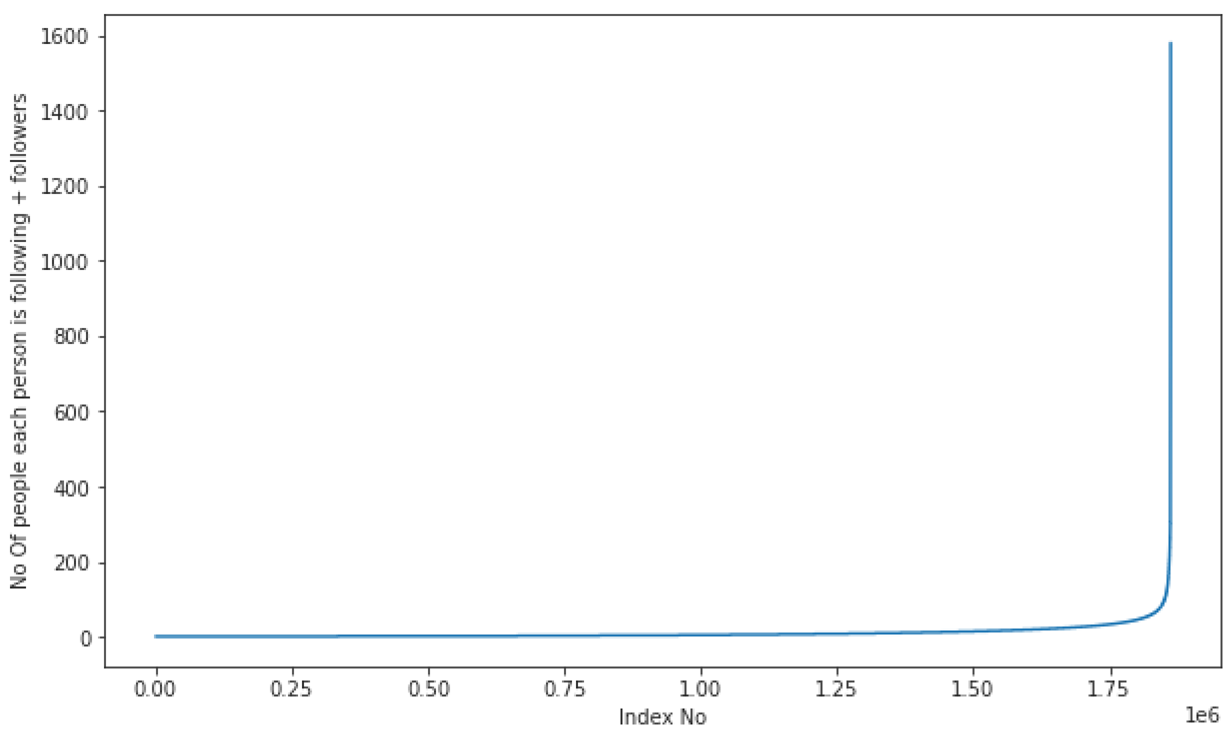
```
for i in g.nodes():
    if len(list(g.predecessors(i)))==0 :
        if len(list(g.successors(i)))==0:
            count+=1
print('No of persons those are not not following anyone and also not having any followers are
```

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

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 + f
```

```
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 + f
```

```
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 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('/data/My Drive/data/after_eda/missing_edges_final.p'):
    #getting all set of edges
    r = csv.reader(open('/data/My Drive/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
            except:
                    missing_edges.add((a,b))
        else:
            continue
    pickle.dump(missing_edges,open('/data/My Drive/data/after_eda/missing_edges_final.p','wb'
else:
    missing_edges = pickle.load(open('/data/My Drive/data/after_eda/missing_edges_final.p','r
```

```
CPU times: user 3.79 s, sys: 3.48 s, total: 7.27 s
Wall time: 8.41 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('/data/My Drive/data/after_eda/train_pos_after_eda.csv')) and (not os.
    #reading total data df
    df_pos = pd.read_csv('/data/My Drive/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
    #and for feature generation
    X_train_pos, X_test_pos, y_train_pos, y_test_pos  = train_test_split(df_pos,np.ones(len(d
    X_train_neg, X_test_neg, y_train_neg, y_test_neg  = train_test_split(df_neg,np.zeros(len(

    print('='*60)
    print("Number of nodes in the train data graph with edges", X_train_pos.shape[0],"=",y_tr
    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],"=",y_test_
    print("Number of nodes in the test data graph without edges", X_test_neg.shape[0],"=",y_t

    #removing header and saving
    X_train_pos.to_csv('/data/My Drive/data/after_eda/train_pos_after_eda.csv',header=False,
    X_test_pos.to_csv('/data/My Drive/data/after_eda/test_pos_after_eda.csv',header=False, in
    X_train_neg.to_csv('/data/My Drive/data/after_eda/train_neg_after_eda.csv',header=False,
    X_test_neg.to_csv('/data/My Drive/data/after_eda/test_neg_after_eda.csv',header=False, in
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('/data/My Drive/data/after_eda/train_pos_after_eda.csv')) and (os.path.isf
    train_graph=nx.read_edgelist('/data/My Drive/data/after_eda/train_pos_after_eda.csv',deli
    test_graph=nx.read_edgelist('/data/My Drive/data/after_eda/test_pos_after_eda.csv',delimi
    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 {} %'.for
```

```
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 7.12007359628
```

> we have a cold start problem here

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

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

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

```
============================================================
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 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 (15100030,)
Shape of traget variable in test (3775008,)
```

```
import networkx as nx
```

```
if os.path.isfile('/data/My Drive/data/after_eda/train_pos_after_eda.csv'):
    train_graph=nx.read_edgelist('/data/My Drive/data/after_eda/train_pos_after_eda.csv',deli
    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))) ==
            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
    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
                                 (len(set(train_graph.predecessors(a)).union(set(train_graph.
        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

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

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

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

```
0.02886751345948129
```

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

```
0
```

3. Ranking Measures

https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.algorithms.link_analysis.pagerank_alg.pagerank.html

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

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

```
else:
    pr = pickle.load(open('/data/My Drive/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 2.709825134193587e-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.r 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
```

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
```

```
                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 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('/data/My Drive/data/fea_sample/katz.p'):
    katz = nx.katz.katz_centrality(train_graph,alpha=0.005,beta=1)
    pickle.dump(katz,open('/data/My Drive/data/fea_sample/katz.p','wb'))
else:
    katz = pickle.load(open('/data/My Drive/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 links. Hubs estimates the node value based on outgoing links.

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

```
if not os.path.isfile('/data/My Drive/data/fea_sample/hits.p'):
    hits = nx.hits(train_graph, max_iter=100, tol=1e-08, nstart=None, normalized=True)
    pickle.dump(hits,open('/data/My Drive/data/fea_sample/hits.p','wb'))
else:
    hits = pickle.load(open('/data/My Drive/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 -3.533801190926718e-19
max 0.004868653379539048
mean 5.615699699308677e-07
```

5. Featurization

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

```
import random
if os.path.isfile('/data/My Drive/data/after_eda/train_after_eda.csv'):
    filename = "/data/My Drive/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('/data/My Drive/data/after_eda/train_after_eda.csv'):
    filename = "/data/My Drive/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('/data/My Drive/data/after_eda/train_after_eda.csv', skiprows=sk
df_final_train['indicator_link'] = pd.read_csv('/data/My Drive/data/train_y.csv', skiprows=sk
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	583350	751732	1

```
df_final_test = pd.read_csv('/data/My Drive/data/after_eda/test_after_eda.csv', skiprows=skip
df_final_test['indicator_link'] = pd.read_csv('/data/My Drive/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	1562045	1824397	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('/data/My Drive/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['des
    df_final_test['jaccard_followers'] = df_final_test.apply(lambda row:
                                            jaccard_for_followers(row['source_node'],row['des

    #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['des
    df_final_test['jaccard_followees'] = df_final_test.apply(lambda row:
                                            jaccard_for_followees(row['source_node'],row['des


        #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['dest
    df_final_test['cosine_followers'] = df_final_test.apply(lambda row:
                                            cosine_for_followers(row['source_node'],row['dest

    #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['dest
    df_final_test['cosine_followees'] = df_final_test.apply(lambda row:
                                            cosine_for_followees(row['source_node'],row['dest


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_follower
```

```
if not os.path.isfile('/data/My Drive/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_st

    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_stag

    hdf = HDFStore('/data/My Drive/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('/data/My Drive/data/fea_sample/storage_sample_stage1.h5', 'tra
    df_final_test = read_hdf('/data/My Drive/data/fea_sample/storage_sample_stage1.h5', 'test
```

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('/data/My Drive/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_
    #mapping adar index on test
```

```
    df_final_test['adar_index'] = df_final_test.apply(lambda row: calc_adar_in(row['source_no

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

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

    #--------------------------------------------------------------------------------------------
    #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['s

    ##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['sou

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

    hdf = HDFStore('/data/My Drive/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('/data/My Drive/data/fea_sample/storage_sample_stage2.h5', 'tra
    df_final_test = read_hdf('/data/My Drive/data/fea_sample/storage_sample_stage2.h5', 'test
```

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

```
#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 [00:32<00:00, 55235.84it/s]
```

```
if not os.path.isfile('/data/My Drive/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.g
    df_final_train['weight_out'] = df_final_train.source_node.apply(lambda x: Weight_out.get(

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


    #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('/data/My Drive/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_p
    df_final_train['page_rank_d'] = df_final_train.destination_node.apply(lambda x:pr.get(x,m

    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,mea
    #================================================================================

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

    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_
    #================================================================================

    #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,

    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(
    df_final_train['authorities_d'] = df_final_train.destination_node.apply(lambda x: hits[1]

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

    hdf = HDFStore('/data/My Drive/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('/data/My Drive/data/fea_sample/storage_sample_stage3.h5', 'tra
    df_final_test = read_hdf('/data/My Drive/data/fea_sample/storage_sample_stage3.h5', 'test
```

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('/data/My Drive/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_
    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_
    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_
    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_
    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('/data/My Drive/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()
```

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 sourse 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
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.successors(b)))== 0
```

```
            return 0
        score = (len(set(train_graph.successors(a))) * len(set(train_graph.successors(b))))

        return score
    except:
        return 0
```

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

    120

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

    120

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

        return score
    except:
        return 0
```

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

    66

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

    66

```
# adding above svd_dot into dataframes
#followers
df_final_train["preferential_followers"] = df_final_train.apply(lambda row: preferential_atta
df_final_test["preferential_followers"] = df_final_train.apply(lambda row: preferential_attac

#followees

df_final_train["preferential_followees"] = df_final_train.apply(lambda row: preferential_atta
df_final_test["preferential_followees"] = df_final_train.apply(lambda row: preferential_attac
```

2 SVD_dot

Add feature called svd_dot. you can calculate svd_dot as Dot product between sourse 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 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()
```

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

```
<1780722x1780722 sparse matrix of type '<class 'numpy.float64'>'
        with 7550015 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.shape[0]):
    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 [00:10<00:00, 9099.86it/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[0]):
    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 [00:05<00:00, 9856.98it/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('/data/My Drive/data/fea_sample/final.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()
```

feature models

```
#reading
from pandas import read_hdf
df_final_train = read_hdf('/data/My Drive/data/fea_sample/final.h5', 'train_df',mode='r')
df_final_test = read_hdf('/data/My Drive/data/fea_sample/final.h5', 'test_df',mode='r')
```

```
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
```

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

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=25)

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_)
```

```
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)
```

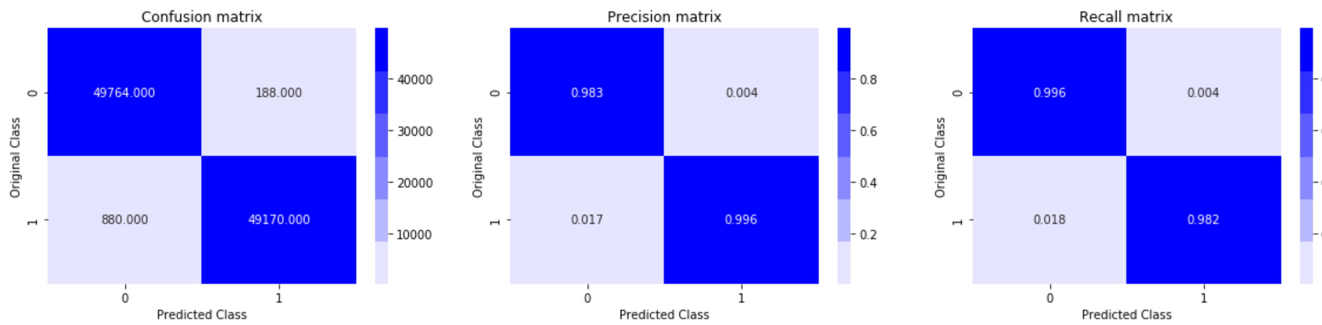
```
clf = XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=11, min_child_weight=1, min_samples_leaf=56,
       min_samples_split=179, missing=None, n_estimators=106, n_jobs=1,
       nthread=None,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


clf.fit(df_final_train,y_train)
y_train_pred = clf.predict(df_final_train)
y_test_pred = clf.predict(df_final_test)


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))
```

```
Train f1 score 0.9892563978754224
Test f1 score 0.9256446597423477
```

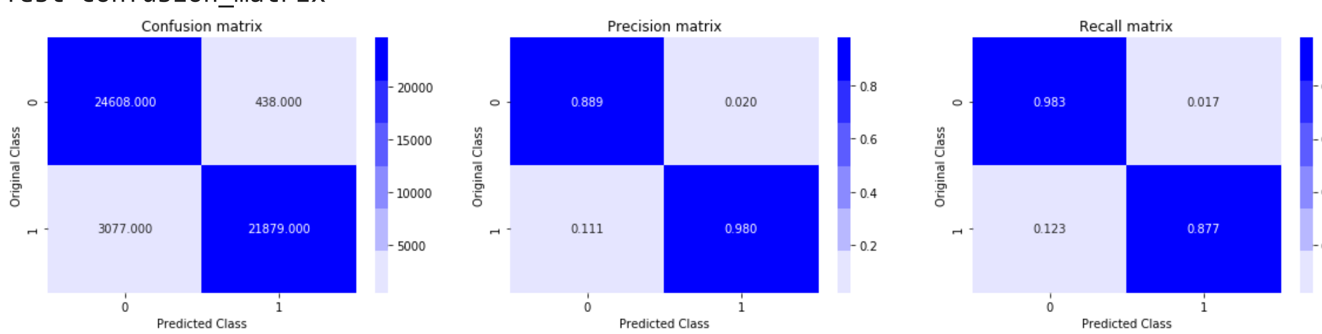
```
from sklearn.metrics import confusion_matrix
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)

    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, 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)
print('Test confusion_matrix')
plot_confusion_matrix(y_test,y_test_pred)
```

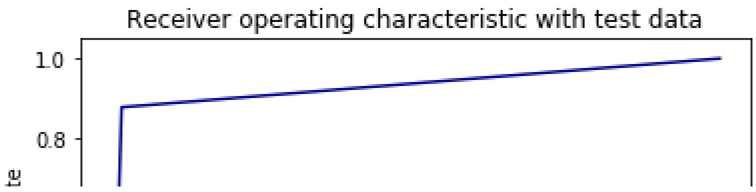
Train confusion_matrix



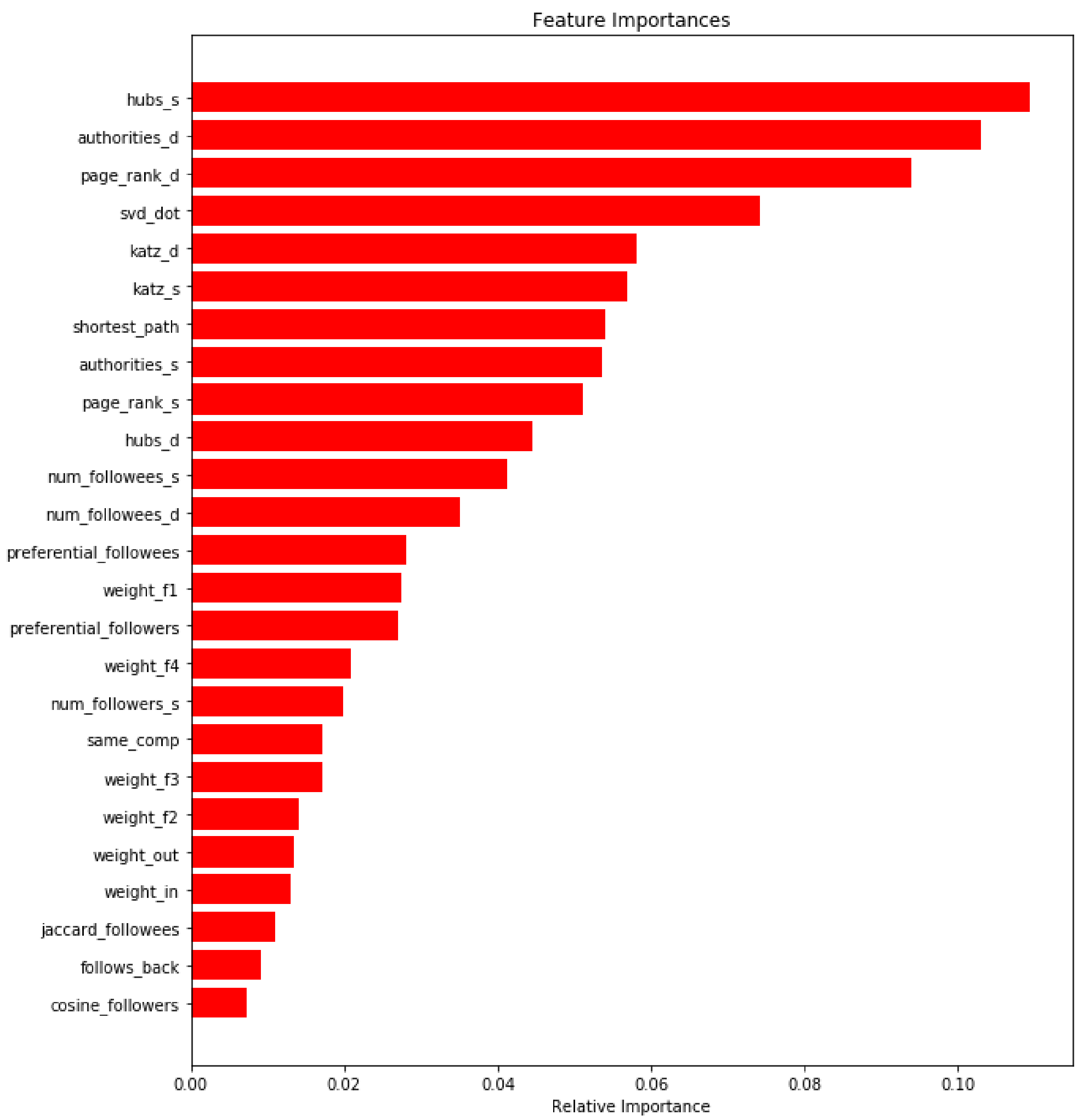
Test confusion_matrix




```
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 liberari load the data with out header
- we observe that the data set contain only two features those are 'scorce_node' and 'destination_node'

- we Mappling this problem into supervised lerning problem
- we use perfomance matrix 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 lessthen 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 lessthen 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 lessthen 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 than10 are 1320326
- No of weakly connected components 45558 weakly connected components wit 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 traget variable in train (15100030,)
- Shape of traget variable in test (3775008,)

Featurization

> Similaity measures

- Jucard Distance

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

- Cosine distance

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

> 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 persion 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*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

- 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 sourse 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

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 matrics

> ######### important features

- hubs_s
- page_rank_d
- svd_dot
- katz-d
- katz-s
- so...on