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 (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://www.cs.cornell.edu/home/kleinber/link-pred.pdf)
  - https://www3.nd.edu/~dial/publications/lichtenwalter2010new.pdf (https://www3.nd.edu/~dial/publications/lichtenwalter2010new.pdf)
  - https://kaggle2.blob.core.windows.net/forum-message-attachments/2594/supervised_link_prediction.pdf (https://kaggle2.blob.core.windows.net/forum-message-attachments/2594/supervised_link_prediction.pdf)
  - https://www.youtube.com/watch?v=2M77Hgy17cg (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

In [1]:

```
#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
```

we have 9437518 no of egde between the different pair of vertices and 2 here is the column number

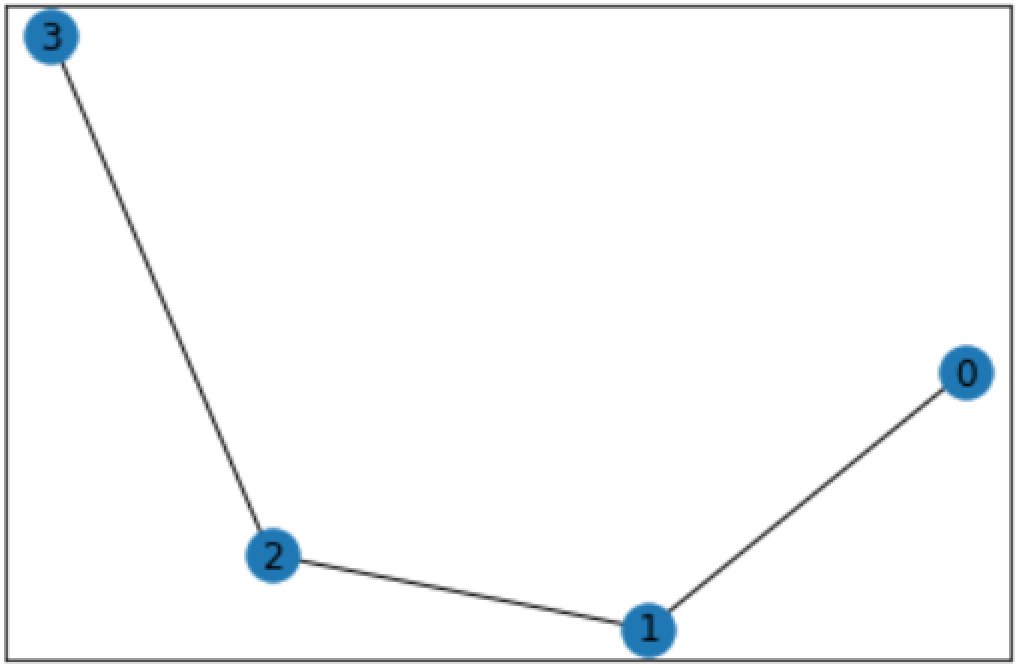
In [2]:

```
#reading graph
if not os.path.isfile('data/after_eda/train_woheader.csv'):
    traincsv = pd.read_csv('data/train.csv')
    print(traincsv[traincsv.isna().any(1)])
    print(traincsv.info())
    print("Number of diplicate entries: ",sum(traincsv.duplicated()))
    traincsv.to_csv('data/after_eda/train_woheader.csv',header=False,index=False)
    print("saved the graph into file")
else:
    # Read a graph from a list of edges.
    g=nx.read_edgelist('data/after_eda/train_woheader.csv',delimiter=',',create_using=n
x.DiGraph(),nodetype=int)
    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

```
nx.draw_networkx(G)
```



```
[4]   pos
```

```
{0: array([0.85867191, 1.        ]),
 1: array([0.30225466, 0.35199499]),
 2: array([-0.30229381, -0.35205172]),
 3: array([-0.85863276, -0.99994327])}
```

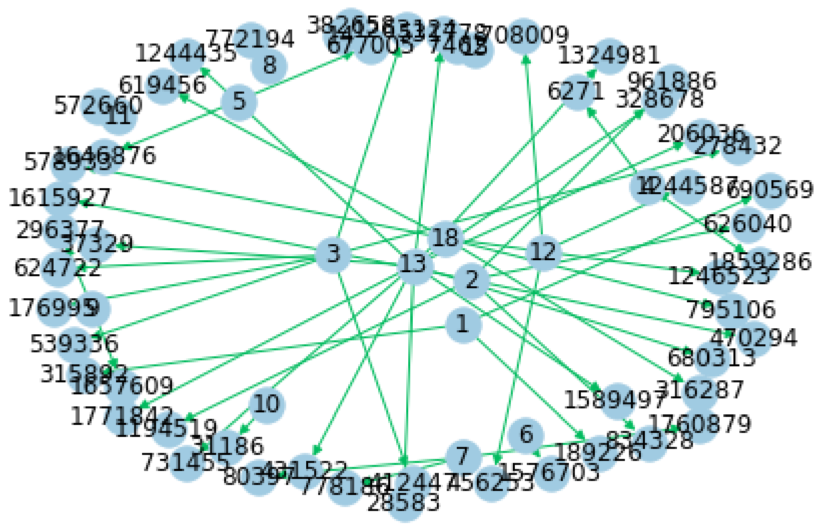
import networkx as nx

G = nx.path_graph(4) pos = nx.spring_layout(G)

In [3]:

```
if not os.path.isfile('train_woheader_sample.csv'):
    pd.read_csv('data/train.csv', nrows=50).to_csv('train_woheader_sample.csv',header=F
alse,index=False)

subgraph=nx.read_edgelist('train_woheader_sample.csv',delimiter=',',create_using=nx.DiG
raph(),nodetype=int)
# https://stackoverflow.com/questions/9402255/drawing-a-huge-graph-with-networkx-and-ma
tplotlib

# spring layout gives the position(cordinates of x and y) of each nodes for ex: node =
 [x-axis value , y-axis value]
pos=nx.spring_layout(subgraph)
nx.draw(subgraph,pos,node_color='#A0CBE2',edge_color='#00bb5e',width=1,edge_cmap=plt.cm
.Blues,with_labels=True)
plt.savefig("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

In [4]:

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

#  we have 9437518 no of egde between the different pair of vertices and 2 here is the
 column number
```

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

1.1 No of followers for each person

In [5]:

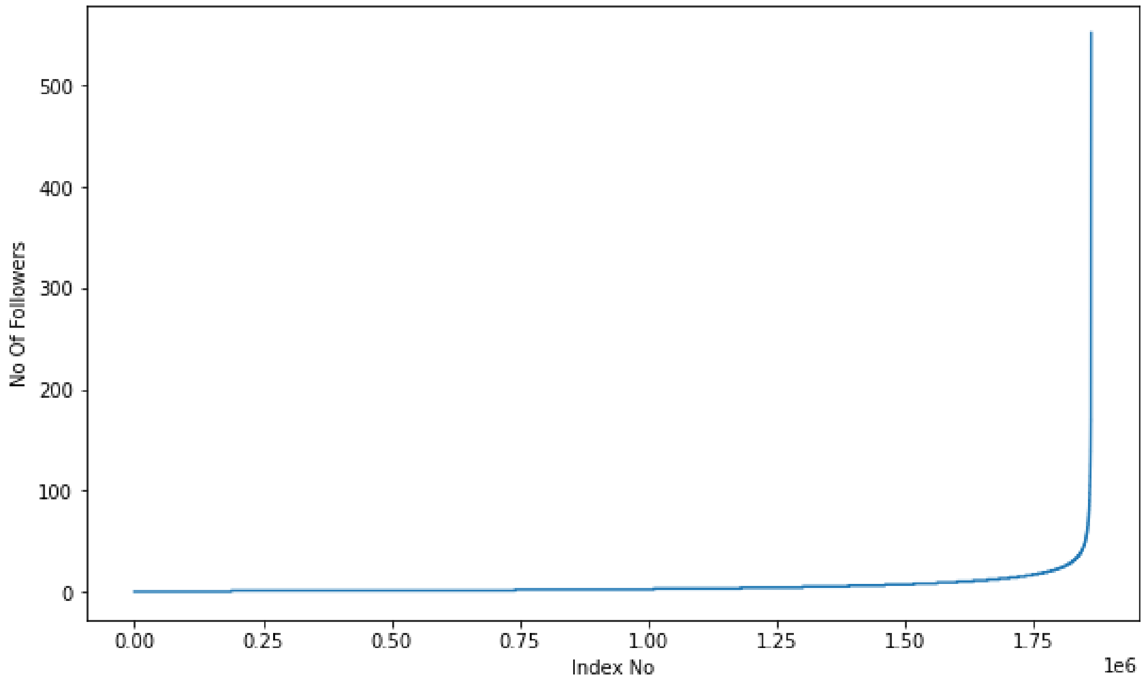
```
# g.in_degree()

# key represent the node/person and value store the number of followers of node/person

# ({1: 3, 690569: 29, 315892: 28, ....})
```

In [6]:

```
#  list(dict(g.in_degree()).values())

# [3,29,28,3,4, .....]
```

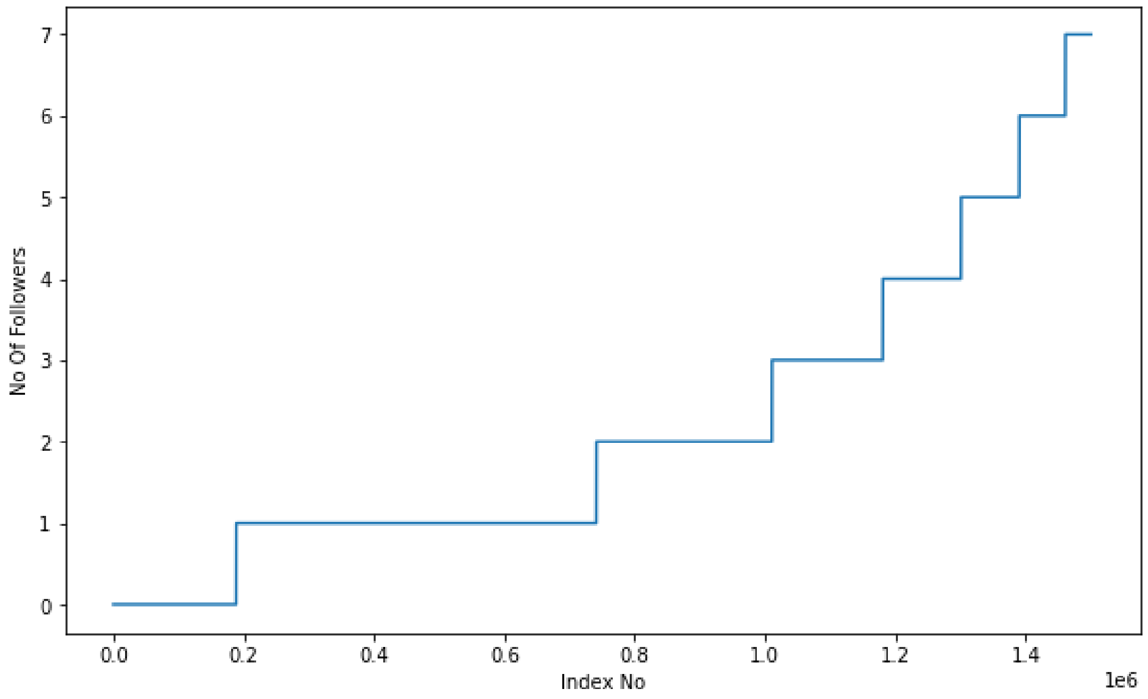
In [7]:

```
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()


# there are 1750000 people havign followers close to 0.
# there is 1 person who is having more than 500 number of followers
```



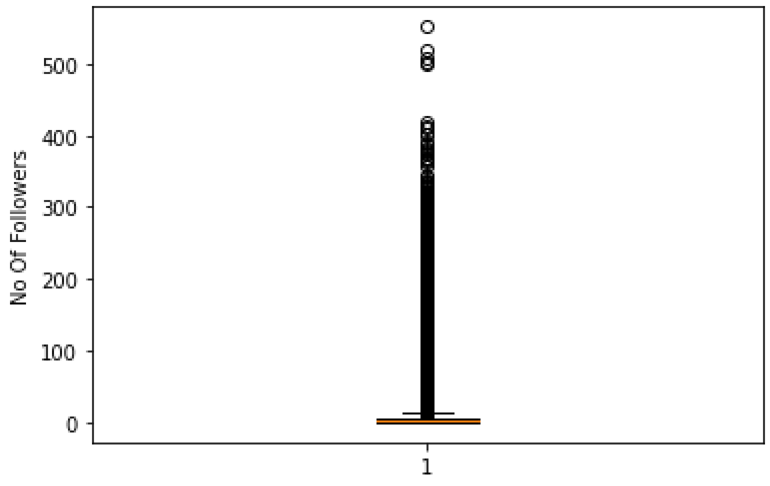
In [8]:

```
# we are intrpreating the small part of user having number of followers (indgreee)

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()


#### observation

# 1) there are 20000 people having 0 followers
# 2) there are 40000 people having 1 followers
# 3) there are 20000 people having 2 followers
# 4) .. so we can intrepret the number of people haivng number of followers
```



In [9]:

```
# observation :- many people are having close to 0 followers

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



In [10]:

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


# 90% of person have 12 followers
# 91% of person have 13 followers
# .. so on
```

```
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.

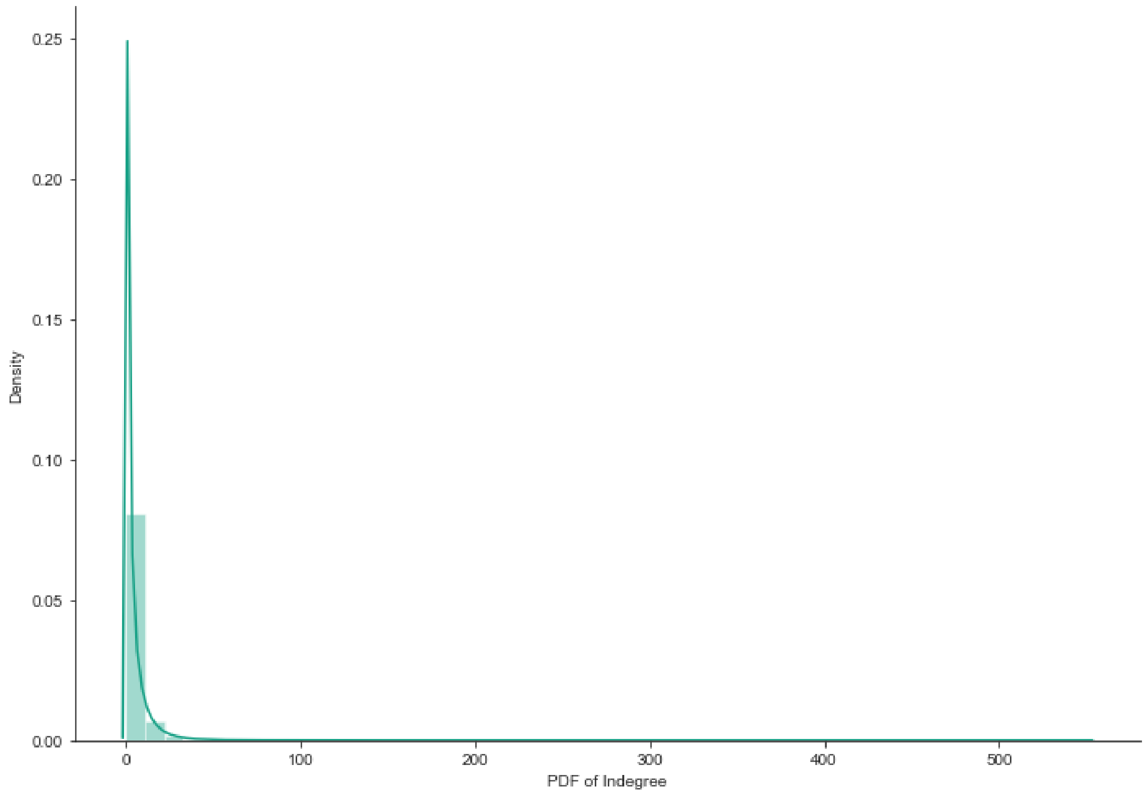
In [11]:

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

# 99.9% of data haivng followers 112 only
```

```
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
```

In [12]:

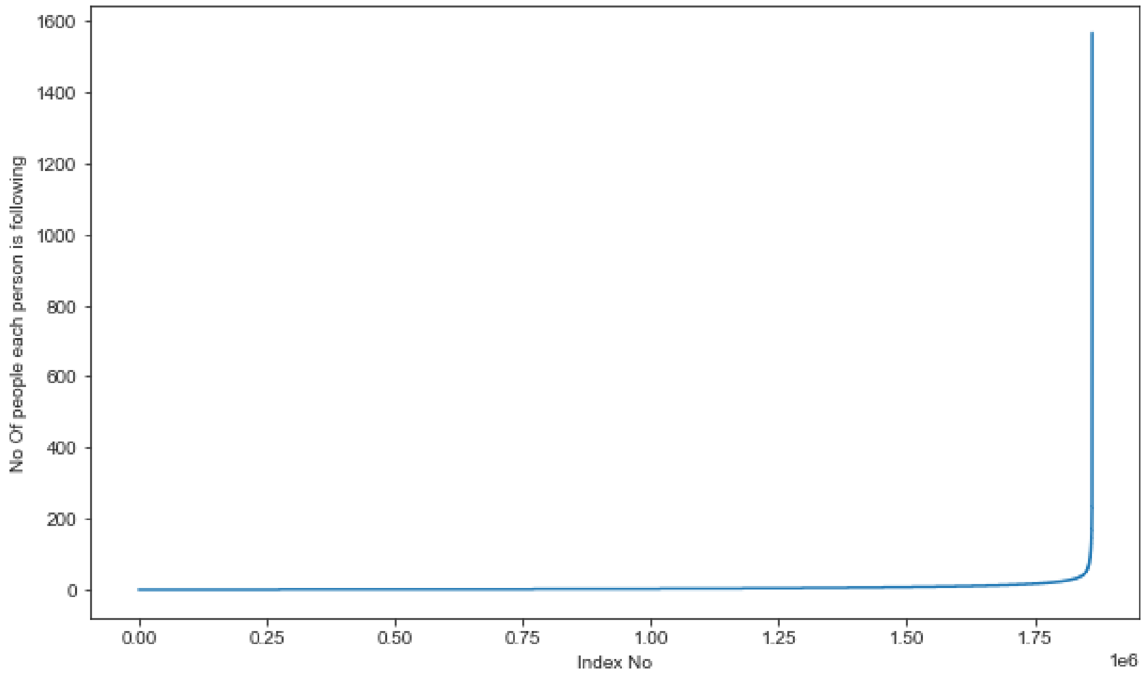
```
%matplotlib inline
sns.set_style('ticks')
fig, ax = plt.subplots()
fig.set_size_inches(11.7, 8.27)
sns.distplot(indegree_dist, color='#16A085')
plt.xlabel('PDF of Indegree')
sns.despine()
#plt.show()


# observation :- same as above
```



1.2 No of people each person is following

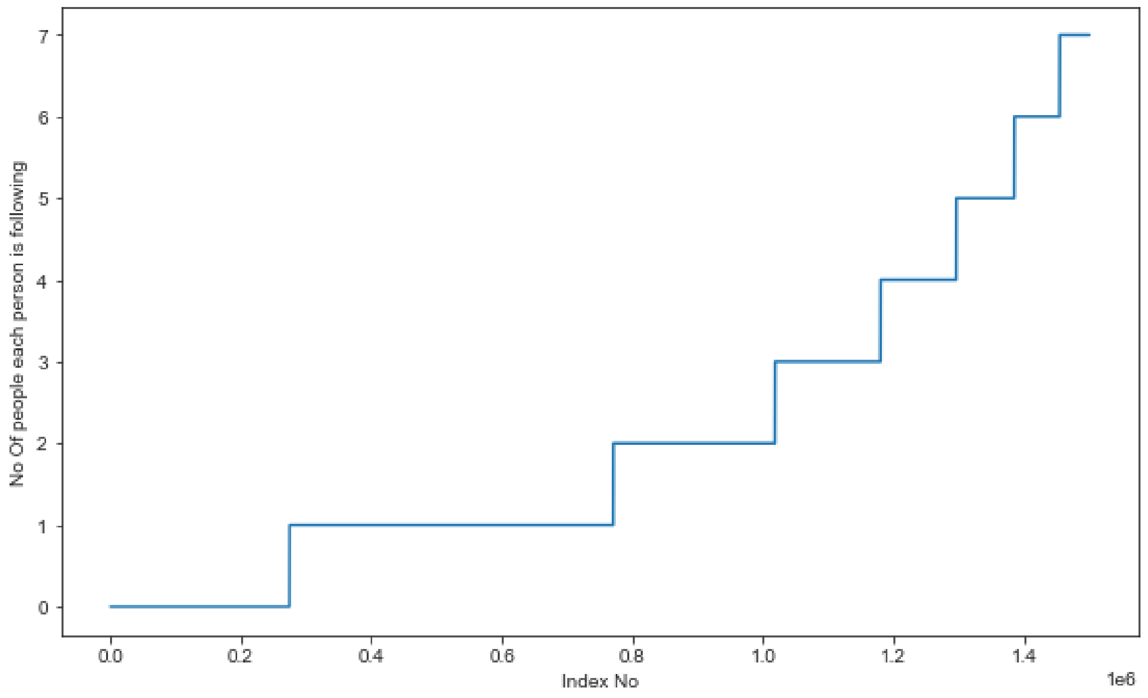
In [13]:

```
# observation :- 1750000 of person have almost close to 0 followers



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()
```



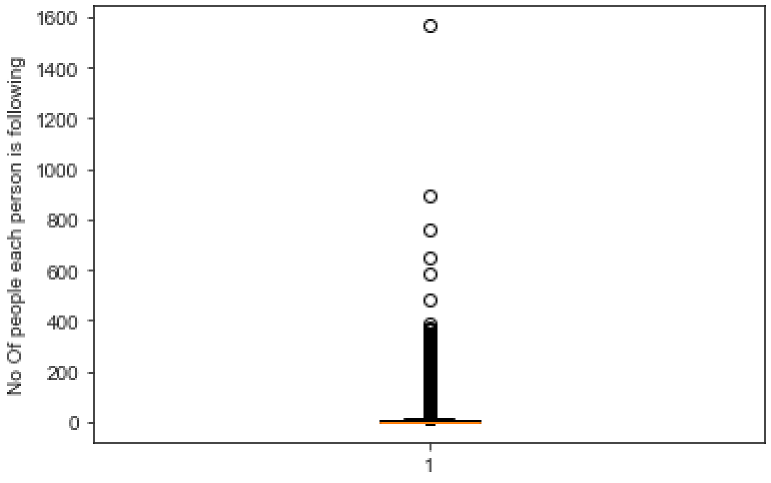
In [14]:

```
# observation :-
# 1) 230000 of people is following 0 only
# 2) 600000 of people is following 1 only
# 3) 200000 of people is following 2 only

outdegree_dist = list(dict(g.out_degree()).values())
outdegree_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()
```



In [15]:

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



In [16]:

```
#new_dict=dict(g.out_degree())  # it gives the number of person following by each perso
n (person number,#of people he is following)
#new_dict.values()  # it gives the value/number of people following by each user
```

In [17]:

```
### 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
```

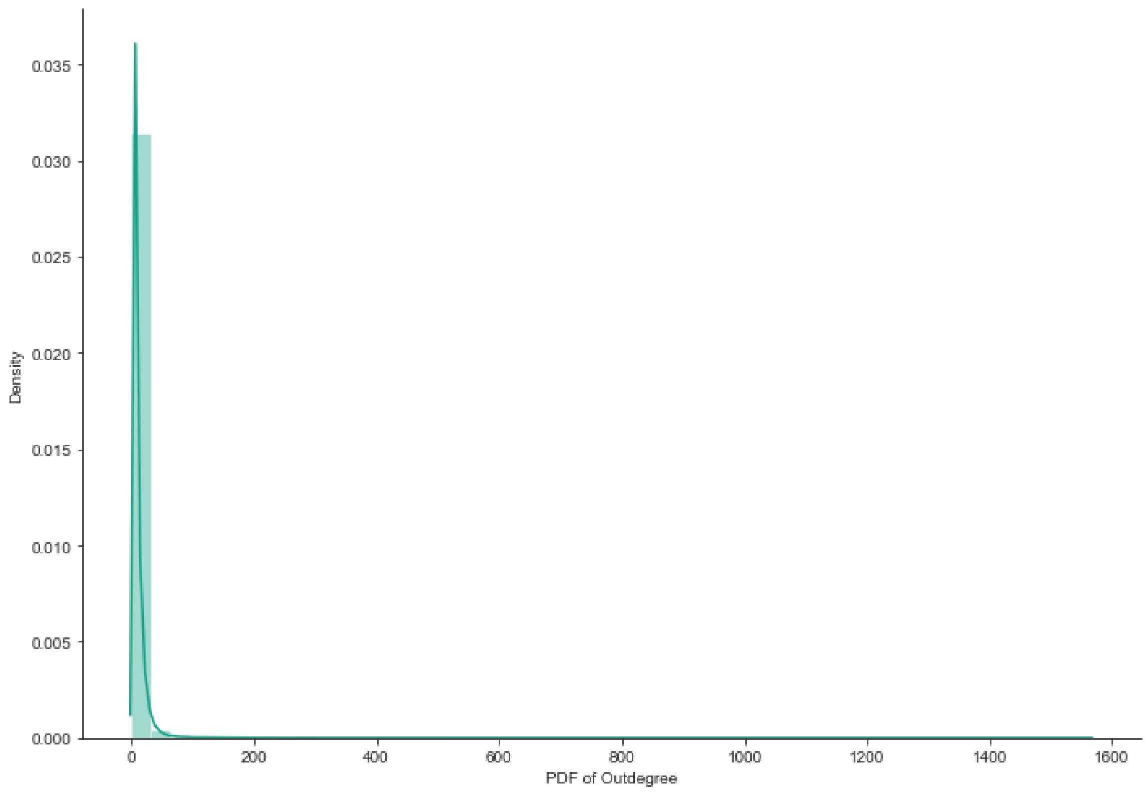
In [18]:

```
### 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
```

In [19]:

```
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()
```



In [20]:

```
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 14.741115
442858524
```

In [21]:

```
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
```

In [22]:

```
# here we are finding the number of user/node having no followers and no followees


###     m -> n

# # A predecessor of n is a node m such that there exists a directed edge from m to n.*
*** predcessor means followers of u
# # A successor of n is a node m such that there exists a directed edge from n to m. **
** successor means followees of u

###  n -> m

count=0
# for each nodes in data/graph
for i in g.nodes():
# Returns an iterator over successor nodes of n.
    if len(list(g.predecessors(i)))==0 :
# Returns an iterator over predecessor nodes of n.
        if len(list(g.successors(i)))==0:
            count+=1
print('No of persons those are not not following anyone and also not having any followe
rs are',count)
```

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

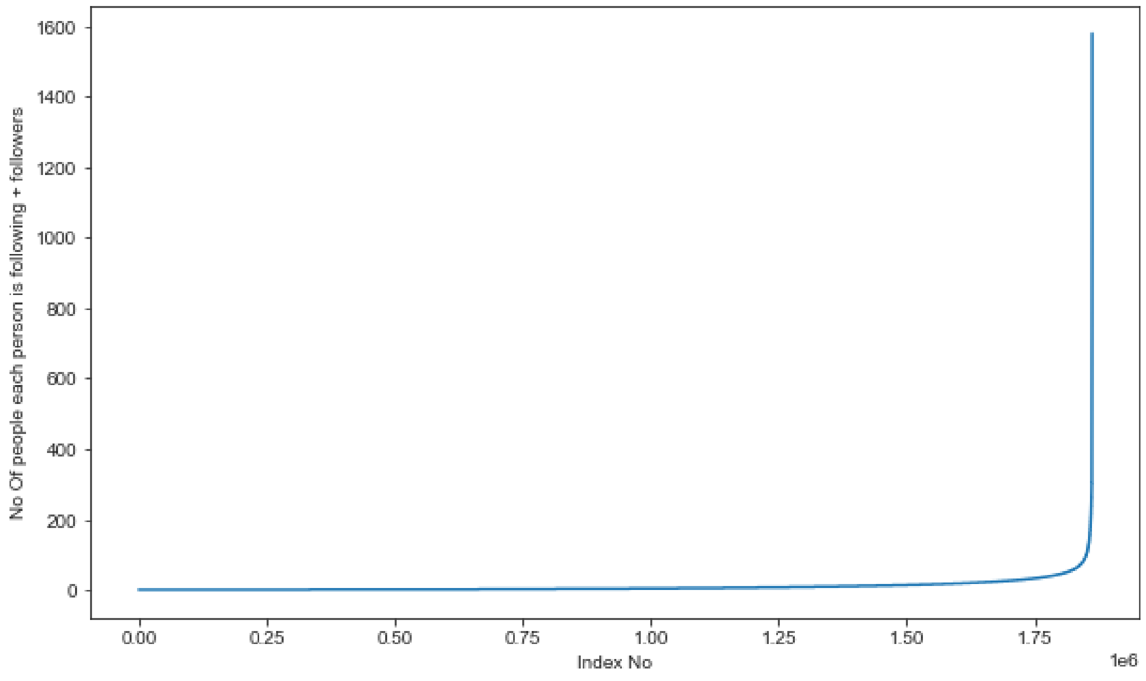
1.3 both followers + following

In [23]:

```
from collections import Counter

# collect the number of followers of each node/user and store them in dict type
dict_in = dict(g.in_degree())

# collect the number of followees of each node/user and store them in dict type
dict_out = dict(g.out_degree())

# len(dict_in)  # so dict_in containsnumber of followers for 1862220 user i.e. unique u
ser as we know there 1862220 unique user

#  sum up the number of followers and followees of each user
d = Counter(dict_in) + Counter(dict_out)

# convert in array
in_out_degree = np.array(list(d.values()))
```

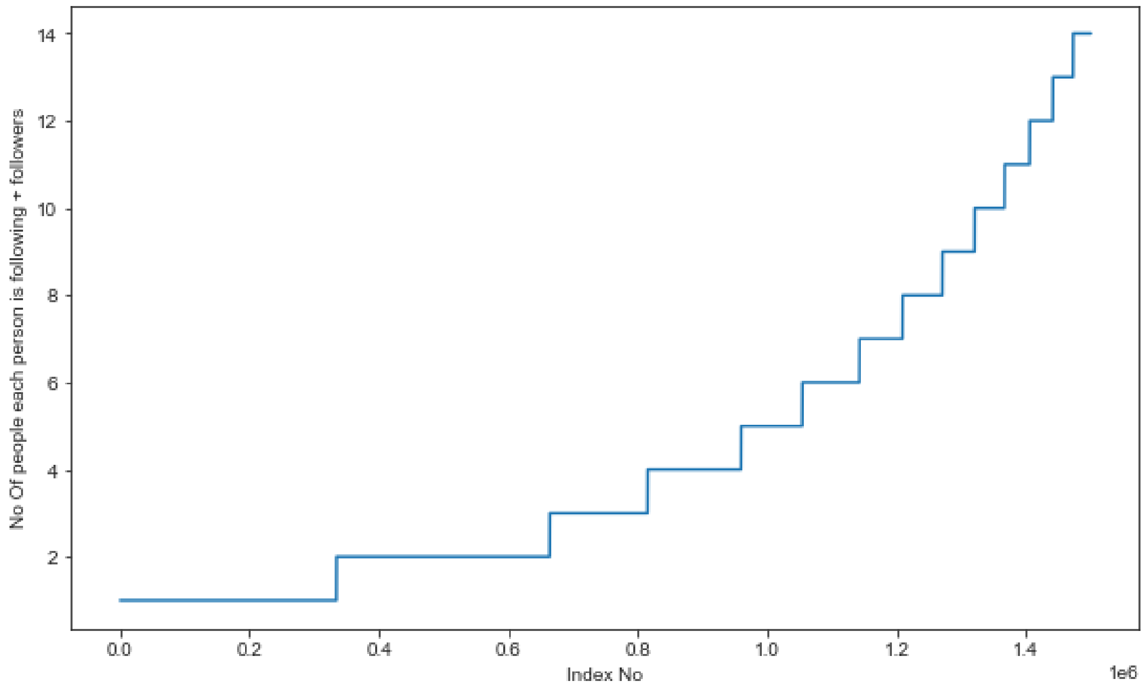
In [24]:

```
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 [25]:

```
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()
```



In [26]:

```
### 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
```

In [27]:

```
### 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
```

In [28]:

```
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 followe
rs + following')
```

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

In [29]:

```
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 followe
rs + following')
```

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

In [30]:

```
print('No of persons having followers + following less than 10 are',np.sum(in_out_degre
e<10))
```

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

In [31]:

```
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.

In [32]:

```
# it contians the pair of nodes having edges between them

r = csv.reader(open('data/after_eda/train_woheader.csv','r'))


for i in r:
    print(i)
    break


# set the connection = 1 between the edges(pair of nodes/user)present in our data.

edges = dict()
for edge in r:
    edges[(edge[0], edge[1])] = 1


# output : ['1', '690569']
# ('1', '315892'): 1,
# ('1', '189226'): 1,
# ('2', '834328'): 1,
# ('2', '1615927'): 1,
# ('2', '1194519'): 1
```

```
['1', '690569']
```

In [33]:

```
%%time
###generating bad edges from given graph ***** ( bad edges means no edge)
import random
if not os.path.isfile('data/after_eda/missing_edges_final.p'):
    #getting all set of edges
    r = csv.reader(open('data/after_eda/train_woheader.csv','r'))
    edges = dict()
    for edge in r:
        edges[(edge[0], edge[1])] = 1
        # It is making the connection set = 1 between the set of edges

    missing_edges = set([])
    while (len(missing_edges)<9437519):

# get the 2 number number for pair of vertices
        a=random.randint(1, 1862220)
        b=random.randint(1, 1862220)

# if this pair of vertices is already present in data get the value else return edge =
 -1
        tmp = edges.get((a,b),-1)

# if there is no edge between the pair of vertices that we are generating and if node a
re not same then check if shortest
# distance between the a and b > 2 or not . if greater than 2 add that pair of vertices
in the missing edges
        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
# store the missing edge in missing edge final
    pickle.dump(missing_edges,open('data/after_eda/missing_edges_final.p','wb'))
#pickle has two main methods. The first one is dump , which dumps an object to a file o
bject and the second one is load ,
# which loads an object from a file object
else:
    missing_edges = pickle.load(open('data/after_eda/missing_edges_final.p','rb'))
```

```
Wall time: 6.37 s
```

In [34]:

```
len(missing_edges)

# there are 9437519 number of pair of vertices where there is no edge
# there are 9437519 number of pair of vertices having an edge

# we have kept equal number of datapoint for each class i.e. no edge(0) and having edge
(1)
# means

#    no of datapoints having an edge = number of datapoints having no edges
```

Out[34]:

```
9437519
```

In [35]:

```
missing_edges   # it contains the pair of vertices that do not have edge between them
```

Out[35]:

```
{(601436, 1322692),
 (891928, 411442),
 (114606, 1845732),
 (669241, 486296),
 (941565, 676512),
 (886515, 666437),
 (1531834, 1045279),
 (532244, 887754),
 (826742, 803876),
 (992702, 1004464),
 (470241, 304412),
 (923940, 865240),
 (299292, 580155),
 (1285487, 812420),
 (772968, 478129),
 (709426, 1590709),
 (171165, 587343),
 (179058, 820372),
 (303384, 1245193),
 (750989, 636333),
 (1544989, 300509),
 (498086, 1698685),
 (1665402, 904893),
 (1118049, 774446),
 (1537308, 1055113),
 (1262718, 479543),
 (1323580, 613557),
 (1616245, 116356),
 (1127800, 137112),
 (1349449, 839499),
 (1703797, 1759845),
 (773282, 317625),
 (1276799, 616525),
 (727094, 1656457),
 (1046304, 1512592),
 (1761611, 1071091),
 (304933, 1594376),
 (980094, 1138822),
 (1602769, 619077),
 (1306181, 1771972),
 (1099334, 828882),
 (186719, 638632),
 (1589047, 801747),
 (1316920, 1652913),
 (802465, 11150),
 (1534803, 767525),
 (1220492, 996114),
 (1266656, 46411),
 (1541817, 1013504),
 (547650, 1175673),
 (1372743, 237610),
 (1492561, 430458),
 (728580, 1584565),
 (585161, 361395),
 (1396993, 1024225),
 (614693, 1634589),
 (1468385, 1680575),
 (944549, 162147),
 (210009, 1392061),
 (1564162, 1246525),
 (909767, 1574005),
 (1803279, 423554),
 (804230, 876582),
 (209120, 1652147),
 (1765746, 1476817),
 (189137, 474943),
 (506016, 30891),
 (836895, 1027192),
 (1047379, 1165059),
 (863861, 1665573),
 (1141538, 1043916),
 (795583, 1347159),
 (639736, 41984),
 (1446198, 424320),
 (93718, 204958),
 (1599877, 1563454),
 (1421827, 976706),
 (338849, 1618093),
 (636965, 1222072),
 (902255, 340385),
 (1697603, 968575),
 (1487005, 259884),
 (741436, 100788),
 (1463736, 376685),
 (789149, 244749),
 (766754, 15641),
 (226748, 189403),
 (254212, 946975),
 (1844493, 1806869),
 (891742, 1085304),
 (1442444, 1595778),
 (1030457, 1710685),
 (894659, 1346361),
 (1206558, 1368704),
 (302682, 1639937),
 (949919, 379507),
 (1443769, 1347128),
 (1019317, 741416),
 (1002156, 278661),
 (1413366, 1668),
 (1833688, 55939),
 (322229, 1408400),
 (1660957, 93497),
 (366686, 293625),
 (1699910, 218128),
 (293560, 1080735),
 (55034, 552409),
 (1179081, 336301),
 (520656, 1181482),
 (1487414, 624109),
 (1103584, 1075566),
 (1372342, 1057713),
 (632844, 950860),
 (1172093, 467509),
 (367666, 1064996),
 (697430, 455236),
 (1487801, 977979),
 (813991, 1613360),
 (242106, 833190),
 (1059328, 1179927),
 (1829312, 612545),
 (468009, 1125037),
 (1860182, 1336212),
 (32712, 634510),
 (1677042, 998903),
 (774243, 934828),
 (145840, 957925),
 (28199, 962298),
 (1216714, 210695),
 (1063206, 1028619),
 (352763, 751097),
 (1281012, 1740395),
 (364662, 684868),
 (1179356, 1518772),
 (787871, 272386),
 (1248590, 1466453),
 (1604793, 1252450),
 (775878, 179500),
 (1128341, 1682321),
 (1467067, 803081),
 (530941, 1050609),
 (684358, 582696),
 (391292, 694589),
 (508065, 935725),
 (885831, 1208219),
 (768193, 475963),
 (675010, 1139627),
 (1599493, 1768423),
 (1113321, 971164),
 (1403554, 579370),
 (1698998, 1247694),
 (1438254, 1194331),
 (1132952, 1098703),
 (1744001, 763663),
 (730187, 1498422),
 (1567359, 1295287),
 (74452, 1499848),
 (1844167, 1730070),
 (1300364, 1597564),
 (361218, 523963),
 (363326, 867160),
 (470586, 1479417),
 (1339168, 1811324),
 (1671496, 893156),
 (485302, 487476),
 (1585157, 1341988),
 (1533006, 1319691),
 (1347685, 209169),
 (1595116, 994362),
 (1030413, 1798255),
 (5826, 1803816),
 (1395888, 1185379),
 (1761624, 886928),
 (982598, 246687),
 (1545070, 1550744),
 (161308, 979380),
 (876128, 1184361),
 (33621, 1513745),
 (530117, 831453),
 (726091, 339173),
 (449741, 819246),
 (587612, 1843664),
 (584208, 1025507),
 (1192000, 839405),
 (1782544, 148930),
 (305130, 180940),
 (1645535, 386476),
 (223429, 388596),
 (436534, 919378),
 (496968, 1660978),
 (600330, 1634689),
 (361152, 1266559),
 (1254206, 1444132),
 (518608, 1019672),
 (1656363, 1413363),
 (96450, 1536593),
 (1662805, 425587),
 (1077806, 1249003),
 (1860735, 206795),
 (857931, 1423877),
 (1619315, 962714),
 (638239, 539333),
 (1517852, 1206115),
 (655381, 1001380),
 (269060, 1491031),
 (1612831, 778935),
 (882547, 1518548),
 (1259899, 1155619),
 (1281446, 856083),
 (307320, 225245),
 (1042513, 862505),
 (52145, 1702186),
 (1612299, 667263),
 (1404372, 302465),
 (1809982, 623627),
 (20104, 1536734),
 (460806, 590649),
 (109622, 117010),
 (1140464, 612730),
 (885109, 1449712),
 (122610, 1019474),
 (862217, 361732),
 (1779321, 1512598),
 (1324517, 1379010),
 (457106, 325300),
 (564704, 1668449),
 (1695620, 1656295),
 (389723, 290644),
 (1535337, 1073473),
 (280683, 1446686),
 (214571, 1604702),
 (732061, 1574259),
 (446459, 1363535),
 (825766, 42078),
 (484234, 305839),
 (1726402, 1789783),
 (1145348, 906970),
 (871631, 1016872),
 (89535, 1151010),
 (846429, 769444),
 (704981, 913714),
 (563207, 357448),
 (944224, 645408),
 (1263405, 1030178),
 (1728699, 759182),
 (1126751, 197554),
 (381540, 440562),
 (587672, 968037),
 (166856, 1230355),
 (108520, 1481736),
 (865214, 1229715),
 (107049, 549784),
 (1790561, 356510),
 (1366620, 1630374),
 (1448466, 1301295),
 (581842, 198712),
 (1069784, 704027),
 (1732207, 1143221),
 (1000061, 1280042),
 (273063, 518054),
 (942315, 1473718),
 (850296, 588826),
 (1503143, 1506046),
 (1296488, 623976),
 (1855503, 1261394),
 (1744863, 1192958),
 (1397771, 1663148),
 (108698, 971667),
 (476533, 1513997),
 (328305, 343620),
 (199212, 1400949),
 (1402784, 1182209),
 (1195167, 97414),
 (849618, 1208234),
 (824667, 484334),
 (764466, 1352815),
 (13982, 1853652),
 (302278, 1272832),
 (1574893, 1495996),
 (1192602, 441189),
 (660807, 172575),
 (1500225, 926601),
 (1366063, 422980),
 (640114, 988703),
 (1630782, 1845177),
 (965495, 443262),
 (956285, 33543),
 (257247, 1166374),
 (1508932, 679986),
 (547576, 741894),
 (1723208, 1530208),
 (959991, 1630408),
 (1658199, 1066984),
 (1389934, 1115288),
 (974161, 1431701),
 (241566, 1646384),
 (201047, 398159),
 (1599076, 204856),
 (844039, 718571),
 (457985, 1066528),
 (793387, 1570354),
 (1854828, 477308),
 (1056370, 325995),
 (116471, 599192),
 (1463286, 265955),
 (1857013, 1237626),
 (572600, 14098),
 (1656556, 422912),
 (1111965, 214174),
 (1359645, 988627),
 (1045953, 1278423),
 (801270, 1728442),
 (1590773, 1295522),
 (530301, 1350325),
 (1633151, 1779982),
 (1667608, 1234096),
 (1350791, 1593133),
 (866961, 746580),
 (967932, 269583),
 (528337, 300547),
 (150272, 441868),
 (1787011, 534170),
 (1322861, 426668),
 (1081028, 1030631),
 (282917, 1194858),
 (1309662, 1843669),
 (904611, 1450816),
 (355925, 248065),
 (1072158, 1221321),
 (1574234, 1061317),
 (73901, 455730),
 (687965, 1464301),
 (1131679, 244354),
 (1794271, 1705753),
 (1482836, 101659),
 (235321, 597660),
 (580472, 420617),
 (1416405, 854265),
 (1688051, 1299167),
 (1545972, 766935),
 (690009, 269606),
 (982865, 335541),
 (514688, 474667),
 (416363, 918145),
 (1783964, 254849),
 (787955, 916611),
 (1210954, 883054),
 (858799, 16196),
 (559553, 1763805),
 (1562958, 1023817),
 (1005797, 153399),
 (1522730, 1017929),
 (1368041, 34477),
 (682731, 502603),
 (1755782, 728733),
 (282420, 1423875),
 (911206, 178246),
 (519583, 79762),
 (44780, 1751643),
 (778536, 904917),
 (1558863, 238015),
 (507871, 452940),
 (64632, 221027),
 (985738, 848974),
 (475044, 2137),
 (1094073, 1240833),
 (1044856, 859447),
 (1315383, 316684),
 (770044, 272602),
 (541698, 1205989),
 (124139, 547632),
 (1785456, 872678),
 (629646, 1001884),
 (536928, 961365),
 (947611, 862702),
 (1361381, 260395),
 (1164827, 1215598),
 (413355, 896866),
 (605635, 1493131),
 (1718588, 657846),
 (152372, 1428650),
 (1177443, 1202601),
 (1089110, 735666),
 (1791974, 657947),
 (623715, 1799281),
 (322072, 697397),
 (708295, 827112),
 (353862, 603439),
 (1618291, 696734),
 (545477, 120990),
 (1590313, 384608),
 (665501, 1470034),
 (962281, 571265),
 (1770013, 138750),
 (907551, 300302),
 (448797, 406495),
 (180317, 1687768),
 (35232, 147230),
 (1481314, 358471),
 (1697561, 1631837),
 (730650, 1362735),
 (1364684, 977076),
 (1388352, 57856),
 (1077396, 198870),
 (975103, 367687),
 (629805, 298051),
 (1543151, 694452),
 (1273233, 338461),
 (738931, 1294202),
 (1270176, 82586),
 (850329, 118963),
 (1544784, 1596797),
 (863826, 1603729),
 (571854, 837766),
 (1421807, 658268),
 (520961, 1735598),
 (800567, 720847),
 (686190, 391125),
 (234599, 172497),
 (494069, 964139),
 (1053934, 969641),
 (223663, 97018),
 (33875, 151294),
 (1084722, 910711),
 (1586260, 702298),
 (1077847, 489182),
 (1547309, 1236087),
 (86391, 1363353),
 (1814297, 998679),
 (1216249, 1204959),
 (662827, 586370),
 (234690, 1847424),
 (1473356, 706431),
 (34398, 284778),
 (240842, 302350),
 (1416840, 486216),
 (1795926, 1458409),
 (376793, 847),
 (698630, 616119),
 (991271, 1132432),
 (1793793, 375704),
 (955339, 47428),
 (1408389, 49509),
 (1249777, 1742044),
 (269074, 1378489),
 (1536961, 1642348),
 (1230997, 1487137),
 (1052036, 1723492),
 (1708385, 99441),
 (986900, 103659),
 (840000, 15537),
 (1436156, 488372),
 (301785, 17435),
 (896615, 948671),
 (1728734, 1141919),
 (333812, 412233),
 (1161537, 914620),
 (1014395, 461538),
 (672264, 1703855),
 (1011055, 1740209),
 (990481, 1255119),
 (670904, 1006686),
 (1839540, 1271821),
 (1005095, 279162),
 (962536, 1403041),
 (135524, 305508),
 (338376, 1830569),
 (796006, 898553),
 (1403600, 1352992),
 (1320386, 152717),
 (592791, 852204),
 (894111, 1726142),
 (883749, 1861276),
 (1314919, 1761832),
 (937175, 198297),
 (1731662, 1128310),
 (1096443, 577271),
 (465100, 1754817),
 (1270847, 106247),
 (489517, 839138),
 (1580695, 927382),
 (1517709, 783876),
 (1482207, 692016),
 (1370272, 1856946),
 (646792, 831039),
 (503070, 1368200),
 (903921, 923010),
 (1675193, 296231),
 (880858, 1343397),
 (711880, 1142120),
 (1798951, 698096),
 (1345191, 142527),
 (781591, 174319),
 (1016738, 1190124),
 (262246, 684517),
 (179139, 612814),
 (1313139, 311880),
 (821983, 1833354),
 (1667445, 347823),
 (438764, 1064385),
 (988206, 1043230),
 (1712620, 194285),
 (1612233, 612949),
 (167344, 851752),
 (212425, 1440905),
 (1026515, 1364507),
 (693919, 468911),
 (396960, 712018),
 (1208482, 214723),
 (764564, 995910),
 (1031790, 275701),
 (1499773, 313326),
 (1313838, 1717385),
 (333141, 92141),
 (510911, 407388),
 (681460, 1784813),
 (491762, 1150723),
 (238871, 66271),
 (318415, 770762),
 (100124, 1258184),
 (859715, 1234152),
 (1689564, 1075439),
 (1438833, 1517964),
 (1475254, 89661),
 (1312715, 1265816),
 (52364, 976395),
 (1020324, 400447),
 (979981, 889599),
 (1372853, 1090994),
 (1757871, 1658255),
 (1391721, 1675775),
 (1854302, 1600108),
 (732355, 1146018),
 (574338, 407836),
 (1368474, 1690369),
 (1211694, 152414),
 (626491, 1298456),
 (668637, 143465),
 (640318, 1548969),
 (805232, 1703876),
 (1028574, 1840407),
 (1799913, 368278),
 (1545509, 1139250),
 (1698525, 473003),
 (1637549, 837647),
 (570514, 1459631),
 (980076, 1002936),
 (73466, 972484),
 (752848, 963471),
 (804198, 882349),
 (956858, 921798),
 (607289, 1610378),
 (368525, 1253014),
 (1213575, 1209877),
 (1739017, 1130190),
 (505214, 258542),
 (450741, 1153174),
 (1503464, 307404),
 (95910, 1857610),
 (283595, 151734),
 (484163, 1192277),
 (759241, 380023),
 (1296774, 478814),
 (153422, 424355),
 (1710634, 359232),
 (180263, 24374),
 (447741, 892869),
 (726282, 459881),
 (297015, 1227993),
 (41040, 126545),
 (1358578, 1738412),
 (488603, 1424142),
 (1296289, 1450967),
 (611688, 1010870),
 (1753562, 31548),
 (732705, 1129103),
 (421573, 1441294),
 (924070, 1534174),
 (1258513, 459235),
 (1836716, 1283600),
 (437740, 239312),
 (1551032, 712033),
 (1105709, 1074612),
 (71916, 350460),
 (1591160, 217258),
 (170642, 1317078),
 (664876, 670484),
 (1656719, 1281101),
 (612001, 1597047),
 (854175, 324100),
 (609983, 750131),
 (1849421, 1644462),
 (219495, 1640589),
 (1686330, 459011),
 (1828415, 1362099),
 (1108224, 945351),
 (545076, 262784),
 (393506, 418866),
 (1165899, 1040602),
 (1601291, 673946),
 (1084566, 1213095),
 (403861, 645541),
 (114015, 1832802),
 (540092, 585966),
 (115558, 1798939),
 (205547, 703247),
 (1432081, 575387),
 (1305182, 1283142),
 (680396, 90536),
 (1613741, 984022),
 (608900, 537245),
 (504927, 1260774),
 (13509, 120650),
 (1323578, 196143),
 (344036, 807539),
 (1540639, 1073864),
 (1278707, 1686384),
 (1526642, 1227473),
 (313032, 451753),
 (1082971, 1016647),
 (200359, 1336490),
 (1287904, 516452),
 (1291675, 1376674),
 (1242638, 999721),
 (1474244, 1723905),
 (1143036, 1215590),
 (1666014, 1046988),
 (341514, 759973),
 (66832, 907493),
 (1601702, 1157213),
 (1306110, 1279348),
 (410917, 1204116),
 (1289418, 1627221),
 (236839, 516894),
 (1470341, 1773903),
 (817449, 964991),
 (993248, 1373641),
 (294811, 391241),
 (634285, 1381685),
 (492186, 841615),
 (636219, 560855),
 (732168, 238862),
 (1821347, 380175),
 (565138, 1270368),
 (792502, 854252),
 (1769008, 124771),
 (1781046, 1768986),
 (589055, 1420912),
 (1404171, 1365955),
 (752202, 94617),
 (1062955, 42851),
 (497829, 65293),
 (1559861, 769538),
 (425465, 339872),
 (200960, 526704),
 (1610500, 223673),
 (107822, 136527),
 (630894, 1153185),
 (517588, 717596),
 (1468295, 1832441),
 (820729, 394696),
 (1313409, 517218),
 (1474986, 1186568),
 (147755, 796170),
 (177038, 133572),
 (1012612, 1858735),
 (1719350, 120133),
 (1186830, 235131),
 (25432, 1607470),
 (791529, 714858),
 (1247745, 1279507),
 (1424369, 668684),
 (959968, 1284676),
 (1259202, 1558767),
 (1509636, 1079605),
 (1452205, 592042),
 (134512, 1437709),
 (278587, 434711),
 (332927, 1857447),
 (255969, 166188),
 (1353051, 921528),
 (607319, 831575),
 (710636, 1763615),
 (609232, 1501787),
 (1500138, 708000),
 (386352, 761678),
 (914201, 1466979),
 (1787705, 1808010),
 (448612, 564074),
 (1099054, 1543722),
 (1021073, 1601001),
 (1285226, 334961),
 (785379, 1497581),
 (414882, 1003378),
 (1245971, 921319),
 (880174, 1559321),
 (1119825, 1259966),
 (716791, 312834),
 (16185, 99367),
 (602500, 1033517),
 (645519, 1806721),
 (720757, 912074),
 (686445, 1124381),
 (232000, 1607370),
 (1205038, 958935),
 (1166815, 592138),
 (706184, 331119),
 (676522, 544296),
 (51203, 245196),
 (1550112, 1117039),
 (57405, 682718),
 (957623, 1271548),
 (840577, 539611),
 (945428, 466100),
 (1401595, 261561),
 (1583044, 1237819),
 (1390085, 162419),
 (1767512, 1117551),
 (994557, 1450815),
 (297048, 450362),
 (69839, 209984),
 (219415, 1245504),
 (521374, 739199),
 (380218, 686409),
 (1705786, 935466),
 (1730573, 595935),
 (786335, 1268235),
 (1101642, 1408293),
 (1506663, 1467893),
 (1004291, 705379),
 (268040, 779778),
 (1023427, 110157),
 (821901, 1160558),
 (685726, 1637789),
 (1587130, 35945),
 (1484939, 1002038),
 (212743, 1122547),
 (344354, 1297421),
 (235942, 697016),
 (1334677, 1645136),
 (636998, 226669),
 (293008, 553081),
 (742686, 1853811),
 (554529, 535623),
 (569703, 1602647),
 (25357, 1749377),
 (39646, 1409786),
 (1067589, 1304266),
 (689609, 267681),
 (1242150, 1606717),
 (944638, 1774261),
 (1779247, 1249726),
 (1084262, 359914),
 (637504, 624595),
 (553136, 302274),
 (630358, 551568),
 (1446222, 1329753),
 (943941, 529230),
 (1205108, 1167550),
 (22176, 812849),
 (1663016, 376957),
 (1394380, 368363),
 (868044, 1349355),
 (621355, 392623),
 (1114892, 1460188),
 (1452006, 89199),
 (958549, 1720877),
 (1782769, 565553),
 (311805, 991164),
 (537714, 527683),
 (251200, 867251),
 (1782294, 957263),
 (1022720, 1234770),
 (1788092, 405992),
 (1304076, 1495045),
 (877092, 1704694),
 (749293, 1542835),
 (1200325, 642875),
 (1446005, 285393),
 (1860451, 1756026),
 (1781806, 52642),
 (614473, 625158),
 (260532, 697156),
 (1669640, 802468),
 (1842996, 32812),
 (1835116, 1042932),
 (1077718, 489538),
 (376380, 105552),
 (1685618, 586136),
 (1177040, 1331933),
 (1581397, 291656),
 (1347587, 987582),
 (687620, 486541),
 (921819, 105479),
 (1238744, 866691),
 (693868, 1773348),
 (1088236, 633874),
 (8218, 255202),
 (961138, 806313),
 (207113, 1265861),
 (919250, 1162869),
 (1733087, 65909),
 (32932, 75595),
 (768475, 1002301),
 (568257, 1376546),
 (525302, 997844),
 (1663991, 1774841),
 (174616, 736180),
 (1593524, 1097849),
 (761598, 1107349),
 (877452, 1786234),
 (698996, 1213492),
 (924909, 868922),
 (932620, 704197),
 (1379274, 1522904),
 (1711891, 506370),
 (1574594, 363542),
 (911053, 938802),
 (1317470, 1767603),
 (999042, 1413016),
 (756740, 1306209),
 (1281448, 1568613),
 (1179018, 101788),
 (608326, 144263),
 (1803751, 1538490),
 (994682, 1085274),
 (942945, 1313562),
 (660111, 315548),
 (1793805, 1058929),
 (1771492, 1850999),
 (1324815, 8935),
 (1267798, 501373),
 (560564, 1403269),
 (1235743, 915476),
 (226044, 986906),
 (946, 489835),
 (943750, 1732112),
 (7523, 1143993),
 (156756, 481174),
 (1469897, 880449),
 (1513996, 498464),
 (1049778, 1182264),
 (750554, 478205),
 (1824676, 806084),
 (380129, 775089),
 (236694, 1385765),
 (1399861, 1766147),
 (31349, 1023555),
 (783961, 1231482),
 (880864, 1717996),
 (326096, 1027259),
 (1179487, 1406104),
 (812393, 513124),
 (1342311, 344529),
 (422379, 1397465),
 (1576162, 1391940),
 (685988, 639902),
 (339324, 803507),
 (327298, 1768036),
 (1218292, 951095),
 (1310990, 1707295),
 (1622025, 1206805),
 (743543, 362198),
 (1145953, 66769),
 (1242381, 977788),
 (499170, 687436),
 (249273, 1647668),
 (1068895, 1459732),
 (1749179, 1068584),
 (687268, 1296159),
 (842273, 1763083),
 (961970, 87681),
 (777378, 310589),
 (1305790, 1170401),
 (203652, 1859717),
 (1048293, 406509),
 (1779747, 289717),
 (1476371, 682776),
 (520816, 1512060),
 (137721, 1851712),
 (1154823, 1837317),
 (423454, 1253093),
 (1806749, 1344184),
 (1512880, 1029972),
 (1246014, 1048519),
 (274797, 630837),
 (855148, 1086037),
 (884918, 1540632),
 (111885, 1304921),
 (113041, 1220992),
 (43945, 250755),
 (835865, 1089836),
 (1197068, 1162515),
 (359941, 71994),
 (623978, 330996),
 (1103984, 78575),
 (1544744, 716210),
 (1242303, 1489524),
 (803683, 1650288),
 (1221378, 993061),
 (1606059, 1152559),
 (154965, 1651015),
 (1411764, 1397192),
 (1236903, 652116),
 (1797272, 1550622),
 (1818400, 1647829),
 (1280225, 676566),
 (1065237, 1511872),
 (1845643, 1098820),
 (809325, 808173),
 (822192, 1028383),
 (228198, 277441),
 (1167708, 642013),
 (543320, 1009893),
 (1842475, 199299),
 (206874, 934300),
 (1456950, 1181297),
 (359335, 1582841),
 (1563322, 1300715),
 (952696, 1347947),
 (359162, 181286),
 (1037563, 1114217),
 (1802422, 90720),
 (225870, 1389853),
 (1047530, 1524233),
 (1339028, 1231152),
 (1816031, 203499),
 (501890, 113095),
 (184413, 24167),
 (1626910, 985728),
 (739637, 700278),
 (1672533, 162454),
 (1377311, 1513349),
 (909159, 1593572),
 (64682, 339349),
 (1589058, 804579),
 (1544829, 635791),
 (1526493, 221445),
 (209035, 408589),
 (397312, 1616929),
 (354977, 660354),
 (1120590, 1283402),
 (1181786, 78587),
 (942074, 816340),
 (1162372, 599017),
 (1292977, 1536935),
 (1673261, 1753294),
 (1435537, 23901),
 (831340, 1746074),
 (1512419, 1666433),
 (1485107, 288433),
 (409785, 1521145),
 (447854, 1210808),
 (699754, 1825484),
 (501392, 1335234),
 (1651790, 555731),
 (1499541, 921860),
 (331484, 923009),
 (1479698, 23251),
 (1618298, 1584394),
 (1819608, 1154721),
 (424461, 777065),
 (887971, 1753156),
 (547724, 1666280),
 (1599476, 76346),
 (741711, 1734765),
 (1771650, 1623507),
 (1838626, 1194630),
 (652578, 1186642),
 (8332, 1282489),
 (45478, 151808),
 (719954, 1131463),
 (1745639, 17267),
 (1662861, 1672689),
 (831911, 895237),
 (1163949, 1776253),
 (228608, 1334084),
 (146152, 1090897),
 (28110, 508468),
 (913599, 244355),
 (703220, 1256517),
 (564785, 1566833),
 (601893, 1329736),
 (1469631, 149258),
 (1067107, 1233878),
 (1378851, 296877),
 (200646, 1728198),
 (205023, 364590),
 (501878, 963834),
 (1329949, 333375),
 (1256643, 760820),
 (706678, 1405489),
 (1265265, 1615999),
 (1612639, 1181133),
 (444948, 1053929),
 (1180548, 1017494),
 (201347, 623229),
 (719661, 1294857),
 (1150217, 1652344),
 (1071889, 537232),
 ...}
```

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

Now we will split the D1 into train , test similarly split the D2 into train and test now now we will merge the D1 train and D2 train to make complete D_train.Same for D2_test.

In [35]:

```
from sklearn.model_selection import train_test_split
if (not os.path.isfile('data/after_eda/train_pos_after_eda.csv')) and (not os.path.isfi
le('data/after_eda/test_pos_after_eda.csv')):
    #reading total data df

    # train.csv contains the pair of vertices having an edge
    df_pos = pd.read_csv('data/train.csv')

    # converting the missing edge data into dataframe and denote it as df_neg
    df_neg = pd.DataFrame(list(missing_edges), columns=['source_node', 'destination_nod
e'])


    # we have to 2 dataframe df_pos will contains the data having an egdes
    # df_neg will contains data having no edges

    # as we know to apply a classification algorithm/model on data first needs to have
 binary(0/1) class label datapoint
    # we have both datapoints of each class in equal number i.e. balanced dataset

    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 creating graph
    #and for feature generation
    # np.ones(len(df_pos)) : will generate the array of size =9437519 contianing 0  i.
e. y class label for datapoint in df_pos
    X_train_pos, X_test_pos, y_train_pos, y_test_pos  = train_test_split(df_pos,np.ones
(len(df_pos)),test_size=0.2, random_state=9)

    # np.zeroes : will generate the array of size(df_neg)= 9437519 containing 1 i.e y c
lass label for datapoint in df_neg
    X_train_neg, X_test_neg, y_train_neg, y_test_neg  = train_test_split(df_neg,np.zero
s(len(df_neg)),test_size=0.2, random_state=9)

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

    #removing header and saving
    X_train_pos.to_csv('data/after_eda/train_pos_after_eda.csv',header=False, index=Fal
se)
    X_test_pos.to_csv('data/after_eda/test_pos_after_eda.csv',header=False, index=False
)
    X_train_neg.to_csv('data/after_eda/train_neg_after_eda.csv',header=False, index=Fal
se)
    X_test_neg.to_csv('data/after_eda/test_neg_after_eda.csv',header=False, index=False
)
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
```

In [36]:

```
if (os.path.isfile('data/after_eda/train_pos_after_eda.csv')) and (os.path.isfile('dat
a/after_eda/test_pos_after_eda.csv')):
    train_graph=nx.read_edgelist('data/after_eda/train_pos_after_eda.csv',delimiter=','
,create_using=nx.DiGraph(),nodetype=int)
    test_graph=nx.read_edgelist('data/after_eda/test_pos_after_eda.csv',delimiter=',',c
reate_using=nx.DiGraph(),nodetype=int)
    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_trN/len(test_nodes_pos)*100))
```

```
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
7.1200735962845405 %
```

we will be facing the cold start problem becuase 7% of datapoint/person/vertices are test but not in train we have a cold start problem here

In [38]:

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


if (os.path.isfile('data/after_eda/train_pos_after_eda.csv')) and \
(os.path.isfile('data/after_eda/test_pos_after_eda.csv')) and \
(os.path.isfile('data/after_eda/train_neg_after_eda.csv')) and \
(os.path.isfile('data/after_eda/test_neg_after_eda.csv')):

    # x train pos contains the training data that have class=1/edge       datapoint/ver
tices/people
    X_train_pos = pd.read_csv('data/after_eda/train_pos_after_eda.csv', names=['source_
node', 'destination_node'])
    X_test_pos = pd.read_csv('data/after_eda/test_pos_after_eda.csv', names=['source_no
de', 'destination_node'])
    # x train neg contians the training data that have class=-1/0 / no edge     datapoi
nt/vertices/people
    X_train_neg = pd.read_csv('data/after_eda/train_neg_after_eda.csv', names=['source_
node', 'destination_node'])
    # test datset containg datapoint having no edge/class =0
    X_test_neg = pd.read_csv('data/after_eda/test_neg_after_eda.csv', names=['source_no
de', 'destination_node'])

    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])

    # combine the training datapoint having class =0 and 1 and store it in X train
    X_train = X_train_pos.append(X_train_neg,ignore_index=True)
    # class label of X train
    y_train = np.concatenate((y_train_pos,y_train_neg))
    # combine the test datapoints having class = 0  and 1 and store it in X_test
    X_test = X_test_pos.append(X_test_neg,ignore_index=True)
    # class label of X_test
    y_test = np.concatenate((y_test_pos,y_test_neg))


    # we have final training and test data i.e. X_train and X_test

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

```
============================================================
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
```

In [39]:

```
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,)
```

In [40]:

```
# computed and store the data for featurization
# please check out FB_featurization.ipynb
```

In [41]:

```
#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
```

In [42]:

```
import warnings
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
```

2.1 Jaccard Distance:

http://www.statisticshowto.com/jaccard-index/ (http://www.statisticshowto.com/jaccard-index/)

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

In [43]:

```
#for followees
# jaccard similarity is also known as jaccard index not jacard distance. To calculate j
acard distance = 1- jaccard index
# let u1 -> u2
# predcessor(u1) it gives the vertex/node of followers of u1
# successor(u2) it gives the vertex/node of followees by u2


# predcessor(u) means followers of u (indegree)
# successor(u) means followees of u  (outdegree)


###                                        sim store the similarity value

def jaccard_for_followees(a,b):
    try:
      # if there is no followers of either a or b then return 0 ( means there is 100 %
 chance that they will not follow each other )
        if len(set(train_graph.successors(a))) == 0  | len(set(train_graph.successors(b
))) == 0:
            return 0
        # calculate the jacc_similarity
        # find the number of common fololowers of a and b / total number of unique foll
owers of a and b
        sim = (len(set(train_graph.successors(a)).intersection(set(train_graph.successo
rs(b)))))/\
                                    (len(set(train_graph.successors(a)).union(set(train
_graph.successors(b)))))
    except:
        return 0
    return sim

# |X U Y| :--- it means total number of unique followers/followees of both a and b
# |X interesection Y| means number of common followers/followees of a and b
```

In [44]:

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

```
0.0
```

In [45]:

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

```
0.0
```

In [46]:

```
#for followers
def jaccard_for_followers(a,b):
    try:
       # if there is no followers of either a or b then return 0 ( means there is 100 %
chance that they will not follow each other )
        if len(set(train_graph.predecessors(a))) == 0  | len(set(g.predecessors(b))) ==
0:
            return 0
        # find jaccard similarity
        # fidn the number of common followers by a and b / total number of unique follo
wers by a and b
        sim = (len(set(train_graph.predecessors(a)).intersection(set(train_graph.predec
essors(b)))))/\
                                 (len(set(train_graph.predecessors(a)).union(set(train_
graph.predecessors(b)))))
        return sim
    except:
        return 0
```

In [47]:

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

```
0.0
```

In [48]:

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

```
0
```

2.2 Cosine distance

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

In [49]:

```
#for followees



def cosine_for_followees(a,b):
    try:
      # if there is no followees of either a or b then return 0 ( means there is 100 %
 chance that they will not follow each other
        if len(set(train_graph.successors(a))) == 0  | len(set(train_graph.successors(b
))) == 0:
            return 0
      # calculate the cosine distance
      # find the number of common followees of a and b / number of followees of a * num
ber of followees of b
        sim = (len(set(train_graph.successors(a)).intersection(set(train_graph.successo
rs(b)))))/\
                                    (math.sqrt(len(set(train_graph.successors(a)))*len
((set(train_graph.successors(b))))))
        return sim
    except:
        return 0
```

In [50]:

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

```
0.0
```

In [51]:

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

```
0
```

In [52]:

```
def cosine_for_followers(a,b):
    try:
        # if there is no followers of either a or b then return 0 ( means there is 100
 % chance that they will not follow each other )
        if len(set(train_graph.predecessors(a))) == 0  | len(set(train_graph.predecesso
rs(b))) == 0:
            return 0
        # calculate the cosine similarity
        # find the number of common followers of a and b / number of followers of a * n
umber of followers of b
        sim = (len(set(train_graph.predecessors(a)).intersection(set(train_graph.predec
essors(b)))))/\
                                     (math.sqrt(len(set(train_graph.predecessors(a))))*
(len(set(train_graph.predecessors(b)))))
        return sim
    except:
        return 0
```

In [53]:

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

```
0.02886751345948129
```

In [54]:

```
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 (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 (https://en.wikipedia.org/wiki/PageRank)

QUES) why use only positive datapoint to for paegrank, daccard simmilairty , cosine similarity ?

because pagerank depends on present links/edges right. if their is no edge between two nodes then it doesnot add any value to final result. so while computing page rank we use only positive edges. but while adding this page rank feature we add this to all the points both positive and negative for negative edges we use mean of page rank of positive edges

In [55]:

```
# we are finding the page rank score for the pair of nodes having edge
# train_graph contains the data having class = 1 and class= 0 also (class = 0 means no
 edge , class = 1 means having edge )


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

In [56]:

```
# gives largest , min , average value of page rank score provided to nodes

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
```

In [57]:

```
#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 that edge and calculating path.

In [58]:

```
#if has direct edge then deleting that edge and calculating shortest path
def compute_shortest_path_length(a,b):
    p=-1
    try:
    # if there is direct edge remove the edge and then calculate the shortesrt path and
again add the original value btw a,b
        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)
    # if no direct edge calculate the shortest path
        else:
            p= nx.shortest_path_length(train_graph,source=a,target=b)
        return p
    # if there is no edge between a,b then shortest path = -1
    except:
        return -1
```

In [59]:

```
#testing
compute_shortest_path_length(77697, 826021)
```

Out[59]:

```
10
```

In [60]:

```
compute_shortest_path_length(669354,1635354)
```

Out[60]:

```
-1
```

4.2 Checking for same community

In [61]:

```
#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
```

In [62]:

```
belongs_to_same_wcc(861, 1659750)
```

Out[62]:

```
0
```

In [63]:

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

Out[63]:

```
0
```

4.3 Adamic/Adar Index:

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

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

In [64]:

```
#adar index

def calc_adar_in(a,b):
    sum=0
    try:
        # find the list of common nodes of a and b.
        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
```

In [65]:

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

Out[65]:

```
0
```

In [66]:

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

Out[66]:

```
0
```

4.4 Is persion was following back:

In [67]:

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

In [68]:

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

Out[68]:

```
1
```

In [69]:

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

Out[69]:

```
0
```

4.5 Katz Centrality:

https://en.wikipedia.org/wiki/Katz_centrality (https://en.wikipedia.org/wiki/Katz_centrality)

https://www.geeksforgeeks.org/katz-centrality-centrality-measure/ (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

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

In [70]:

```
if not os.path.isfile('data/fea_sample/katz.p'):
    katz = nx.katz.katz_centrality(train_graph,alpha=0.025,beta=1)
    pickle.dump(katz,open('data/fea_sample/katz.p','wb'))
else:
    katz = pickle.load(open('data/fea_sample/katz.p','rb'))
```

In [71]:

```
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
```

In [72]:

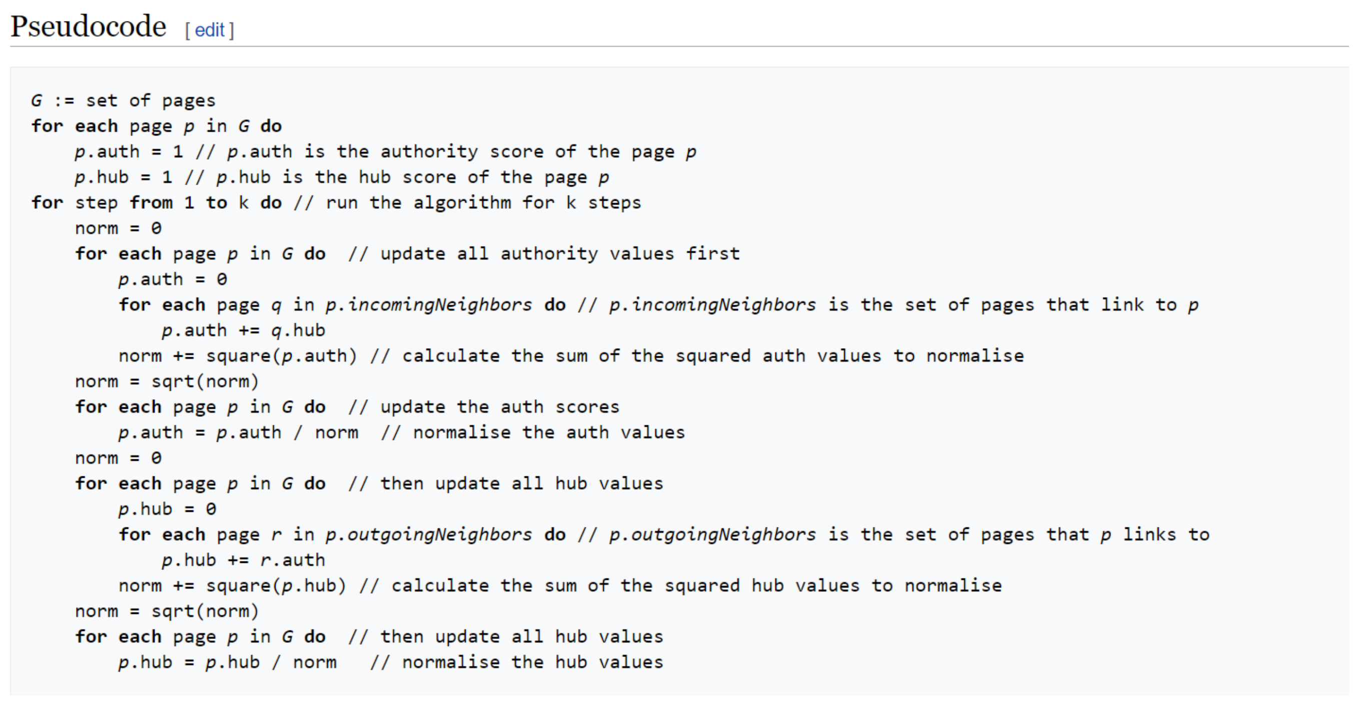
```
mean_katz = float(sum(katz.values())) / len(katz)
print(mean_katz)
```

```
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 (https://en.wikipedia.org/wiki/HITS_algorithm)



Pseudocode [edit]

```
G := set of pages
for each page p in G do
    p.auth = 1 // p.auth is the authority score of the page p
    p.hub = 1 // p.hub is the hub score of the page p
for step from 1 to k do // run the algorithm for k steps
    norm = 0
    for each page p in G do  // update all authority values first
        p.auth = 0
        for each page q in p.incomingNeighbors do // p.incomingNeighbors is the set of pages that link to p
            p.auth += q.hub
        norm += square(p.auth) // calculate the sum of the squared auth values to normalise
    norm = sqrt(norm)
    for each page p in G do  // update the auth scores
        p.auth = p.auth / norm  // normalise the auth values
    norm = 0
    for each page p in G do  // then update all hub values
        p.hub = 0
        for each page r in p.outgoingNeighbors do // p.outgoingNeighbors is the set of pages that p links to
            p.hub += r.auth
        norm += square(p.hub) // calculate the sum of the squared hub values to normalise
    norm = sqrt(norm)
    for each page p in G do  // then update all hub values
        p.hub = p.hub / norm   // normalise the hub values
```

http://pi.math.cornell.edu/~mec/Winter2009/RalucaRemus/Lecture4/lecture4.html (http://pi.math.cornell.edu/~mec/Winter2009/RalucaRemus/Lecture4/lecture4.html)

In [73]:

```
if not os.path.isfile('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/fea_sample/hits.p','wb'))
else:
    hits = pickle.load(open('data/fea_sample/hits.p','rb'))
```

In [74]:

```
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

In [75]:

```
import random

if os.path.isfile('data/after_eda/train_after_eda.csv'):
    filename = "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
```

In [76]:

```
if os.path.isfile('data/after_eda/train_after_eda.csv'):
    filename = "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 h
eader)
    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
```

In [77]:

```
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
```

In [78]:

```
# add the indicator_link as the columns which acts a yi(class label) for xi's


df_final_train= pd.read_csv('data/after_eda/train_after_eda.csv',skiprows=skip_train,na
mes=['source_node','destination_node'])
df_final_train['indicator_link']=pd.read_csv('data/train_y.csv',skiprows=skip_train,nam
es=['indicator_link'])
print('our final train data',df_final_train.shape)
df_final_train.head()
```

```
our final train data (100002, 3)
```

Out[78]:

	source_node	destination_node	indicator_link
0	273084	1505602	1
1	333578	879520	1
2	1711901	1539921	1
3	606966	1224294	1
4	1511622	1352161	1

In [79]:

```
# add the indicator_link in the columns whihc acts a class label(yi) for all xi's

df_final_test = pd.read_csv('data/after_eda/test_after_eda.csv',skiprows=skip_test,name
s=['source_node','destination_node'])
df_final_test['indicator_link']=pd.read_csv('data/test_y.csv',skiprows=skip_test,names=
['indicator_link'])
print('our final test data',df_final_test.shape)
df_final_test.head()
```

```
our final test data (50002, 3)
```

Out[79]:

	source_node	destination_node	indicator_link
0	848424	784690	1
1	1562045	1824397	1
2	131103	187682	1
3	971595	646855	1
4	1593616	727663	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

In [80]:

```
if not os.path.isfile('data/fea_sample/storage_sample_stage1.h5'):
    #mapping jaccrd followers to train and test data

    # We can use the apply() function to apply the lambda function to both rows and col
umns of a dataframe.
    # If the axis argument in the apply() function is 0, then the lambda function gets
 applied to each column, and
    # if 1, then the function gets applied to each row.


    df_final_train['jaccard_followers'] = df_final_train.apply(lambda row:
                                            jaccard_for_followers(row['source_node'],ro
w['destination_node']),axis=1)
    df_final_test['jaccard_followers'] = df_final_test.apply(lambda row:
                                            jaccard_for_followers(row['source_node'],ro
w['destination_node']),axis=1)

    #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'],ro
w['destination_node']),axis=1)
    df_final_test['jaccard_followees'] = df_final_test.apply(lambda row:
                                            jaccard_for_followees(row['source_node'],ro
w['destination_node']),axis=1)


        #mapping cosine 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_node']),axis=1)
    df_final_test['cosine_followers'] = df_final_test.apply(lambda row:
                                            cosine_for_followers(row['source_node'],row
['destination_node']),axis=1)

    #mapping cosine 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_node']),axis=1)
    df_final_test['cosine_followees'] = df_final_test.apply(lambda row:
                                            cosine_for_followees(row['source_node'],row
['destination_node']),axis=1)
```

In [81]:

```
def compute_features_stage1(df_final):
   # compute the number of followers, followees of source node and destination nodes
    # compute the number of intersection/common number of followers , followees for sou
rce node and destination node

    num_followers_s=[]
    num_followees_s=[]
    num_followers_d=[]
    num_followees_d=[]
    inter_followers=[]
    inter_followees=[]

    for i,row in df_final.iterrows():
        try:
            # compute the list of incoming link for each source node
            s1=set(train_graph.predecessors(row['source_node']))
            # compute the list of outgoing link for each source node
            s2=set(train_graph.successors(row['source_node']))
        except:
            s1 =set()
            s2 = set()
        try:
            # compute the lsit of incoming link for each destination node
            d1 = set(train_graph.predecessors(row['destination_node']))
            # compute the list of outgoing link for each destination node/user
            d2= set(train_graph.successors(row['destination_node']))
        except:
            d1 = set()
            d2 = set()

        # add the number of followers i num_followers and num of followes of source nod
e in num_followees
        num_followers_s.append(len(s1))
        num_followees_s.append(len(s2))

        # add the number of followers of destination node in num_followers and num of_f
ollowees of destintion node in num_followees
        num_followers_d.append(d1)
        num_followees_d.append(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_fo
llowers, inter_followees
```

In [82]:

```
if not os.path.isfile('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_featu
res_stage1(df_final_train)

    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_feature
s_stage1(df_final_test)

    hdf = HDFStore('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/fea_sample/storage_sample_stage1.h5', 'train_df',mo
de='r')
    df_final_test = read_hdf('data/fea_sample/storage_sample_stage1.h5', 'test_df',mode
='r')
```

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

In [83]:

```
if not os.path.isfile('data/fea_sample/storage_sample_stage2.h5'):

    # map the adar index to train data
    df_final_train['adar_index'] = df_final_train.apply(lambda row:
                                            calc_adar_in(row['source_node'],row['destin
ation_node']),axis=1)
    # map the adar index to test data
    df_final_test['adar_index']=df_final_test.apply(lambda row:
                                            calc_adar_in(row['source_node'],row['destin
ation_node']),axis=1)

    #--------------------------------------------------------------------------------
----------------------

    # map the is follwoing back to train data
    df_final_train['is_following_back'] = df_final_train.apply(lambda row:
                                            follows_back(row['source_node'],row['destin
ation_node']),axis=1)
    # map the is following back to test data
    df_final_test['is_following_back']=df_final_test.apply(lambda row:
                                            follows_back(row['source_node'],row['destin
ation_node']),axis=1)

    #--------------------------------------------------------------------------------
----------------------


    # 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_node'],row[
'destination_node']),axis=1)

    # 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_node'],row[
'destination_node']),axis=1)

    #--------------------------------------------------------------------------------
----------------------


    # mapping shortest path between soruce and destination node
    df_final_train['shortest_path '] = df_final_train.apply(lambda row:
                                            compute_shortest_path_length(row['source_no
de'],row['destination_node']),axis=1)
    # map the adar index to test data
    df_final_test['shortest_path']=df_final_test.apply(lambda row:
                                            compute_shortest_path_length(row['source_no
de'],row['destination_node']),axis=1)


    hdf = HDFStore('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/fea_sample/storage_sample_stage2.h5', 'train_df',mo
de='r')
```

```
    df_final_test = read_hdf('data/fea_sample/storage_sample_stage2.h5', 'test_df',mode
='r')
```

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

In [84]:

```
#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 [00:54<00:00, 32803.89it/s]
```

In [85]:

```
if not os.path.isfile('data/fea_sample/storage_sample_stage3.h5'):
    #mapping to pandas train
    df_final_train['weight_in'] = df_final_train.destination_node.apply(lambda x: Weigh
t_in.get(x,mean_weight_in))
    df_final_train['weight_out'] = df_final_train.source_node.apply(lambda x: Weight_ou
t.get(x,mean_weight_out))

    #mapping to pandas test
    df_final_test['weight_in'] = df_final_test.destination_node.apply(lambda x: Weight_
in.get(x,mean_weight_in))
    df_final_test['weight_out'] = df_final_test.source_node.apply(lambda x: Weight_out.
get(x,mean_weight_out))


    #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_ou
t)
    df_final_test['weight_f4'] = (1*df_final_test.weight_in + 2*df_final_test.weight_ou
t)
```

In [86]:

```
if not os.path.isfile('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.g
et(x,mean_pr))

    df_final_test['page_rank_s'] = df_final_test.source_node.apply(lambda x:pr.get(x,me
an_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,me
an_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].ge
t(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: h
its[1].get(x,0))

    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: hit
s[1].get(x,0))
    #================================================================================

    hdf = HDFStore('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/fea_sample/storage_sample_stage3.h5', 'train_df',mo
de='r')
    df_final_test = read_hdf('data/fea_sample/storage_sample_stage3.h5', 'test_df',mode
='r')
```

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

In [87]:

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

In [88]:

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

In [89]:

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

In [90]:

```
Adj.shape
```

Out[90]:

```
(1780722, 1780722)
```

In [91]:

```
print(Adj[:10])   # it gives the similarity matrix i.e.
```

```
  (0, 180907)	1.0
  (0, 301965)	1.0
  (1, 598394)	1.0
  (1, 797643)	1.0
  (1, 919667)	1.0
  (1, 1142212)	1.0
  (1, 1545223)	1.0
  (2, 169215)	1.0
  (2, 266193)	1.0
  (2, 515484)	1.0
  (2, 1207861)	1.0
  (3, 5985)	1.0
  (3, 1777903)	1.0
  (5, 1507720)	1.0
  (6, 76841)	1.0
  (6, 743869)	1.0
  (6, 1683883)	1.0
  (8, 1585129)	1.0
  (9, 547335)	1.0
```

In [92]:

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

```
ajdency matrix shape (1780722, 1780722)
U shape (1780722, 6)
s shape (6,)
v shape (6, 1780722)
```

In [93]:

```
if not os.path.isfile('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', 'sv
d_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','sv
d_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', 'sv
d_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','sv
d_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('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()
else:
    df_final_train = read_hdf('data/fea_sample/storage_sample_stage4.h5', 'train_df',mo
de='r')
    df_final_test = read_hdf('data/fea_sample/storage_sample_stage4.h5', 'test_df',mode
='r')
```

In [94]:

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

Out[94]:

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

2 rows × 54 columns

In [95]:

```
# df_Final_train_copy will be used later for xgboost hypereparameter tuning

df_final_train_copy = df_final_train
df_final_test_copy = df_final_test
```

In [96]:

```
df_final_train.to_csv('data/after_eda/final_train.csv', encoding='utf-8', index=False)
df_final_test.to_csv('data/after_eda/final_test.csv',encoding='utf-8',index=False)
```

In [97]:

```
df_final_train= pd.read_csv('data/after_eda/final_train.csv')
df_final_test= pd.read_csv('data/after_eda/final_test.csv')
```

In [98]:

```
df_final_train.shape
```

Out[98]:

```
(100002, 54)
```

In [99]:

```
df_final_train_copy = df_final_train
df_final_test_copy= df_final_test
```

In [100]:

```
df_final_train_copy.shape
```

Out[100]:

```
(100002, 54)
```

Models

In [101]:

```
df_final_train.columns
```

Out[101]:

```
Index(['source_node', 'destination_node', 'indicator_link',
       'jaccard_followers', 'jaccard_followees', 'cosine_followers',
       'cosine_followees', 'num_followers_s', 'num_followees_s',
       'num_followees_d', 'inter_followers', 'inter_followees', 'adar_inde
x',
       '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'],
      dtype='object')
```

In [102]:

```
print(df_final_train.shape)
```

```
(100002, 54)
```

In [103]:

```
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)
```

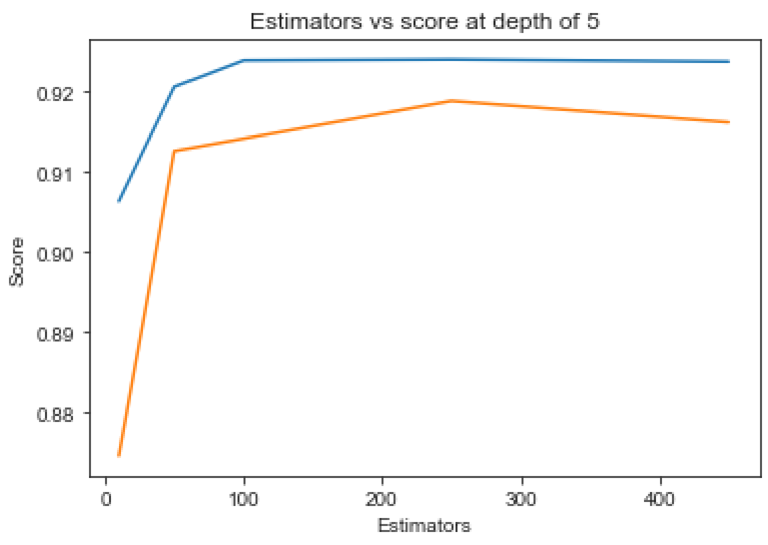
In [104]:

```
# buidl th random forest model with n_estimator and depth as hyperparameter

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score

estimator = [10,50,100,250,450]

train_scores=[]
test_scores=[]

for i in estimator:
    clf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=5, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=52, min_samples_split=120,
            min_weight_fraction_leaf=0.0, n_estimators=i, n_jobs=-1,random_state=25,ver
bose=0,warm_start=False)
    clf.fit(df_final_train,y_train)
    train_sc = f1_score(y_train,clf.predict(df_final_train))
    test_sc= f1_score(y_test,clf.predict(df_final_test))

    train_scores.append(train_sc)
    test_scores.append(test_sc)

    print('Estimators = ',i,'Train Score',train_sc,'test Score',test_sc)
plt.plot(estimator,train_scores,label='Train Score')
plt.plot(estimator,test_scores,label='Test Score')
plt.xlabel('Estimators')
plt.ylabel('Score')
plt.title('Estimators vs score at depth of 5')
```

```
Estimators =  10 Train Score 0.9063252121775113 test Score 0.8745605278006
858
Estimators =  50 Train Score 0.9205725512208812 test Score 0.9125653355634
538
Estimators =  100 Train Score 0.9238690848446947 test Score 0.914119971415
3599
Estimators =  250 Train Score 0.9239789348046863 test Score 0.918800723266
4732
Estimators =  450 Train Score 0.9237190618658074 test Score 0.916150768582
8595
```

Out[104]:

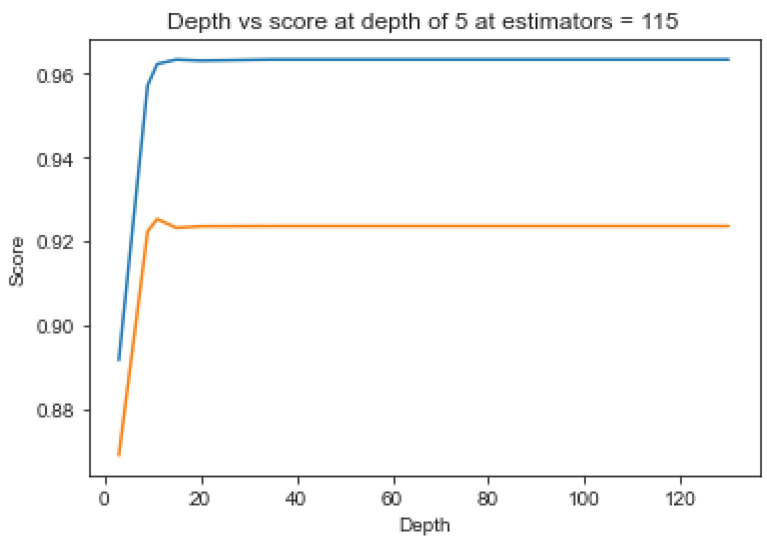
```
Text(0.5, 1.0, 'Estimators vs score at depth of 5')
```



In [105]:

```
depths = [3,9,11,15,20,35,50,70,130]
train_scores = []
test_scores = []
for i in depths:
    clf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=i, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=52, min_samples_split=120,
            min_weight_fraction_leaf=0.0, n_estimators=115, n_jobs=-1,random_state=25,v
erbose=0,warm_start=False)
    clf.fit(df_final_train,y_train)
    train_sc = f1_score(y_train,clf.predict(df_final_train))
    test_sc = f1_score(y_test,clf.predict(df_final_test))
    test_scores.append(test_sc)
    train_scores.append(train_sc)
    print('depth = ',i,'Train Score',train_sc,'test Score',test_sc)
plt.plot(depths,train_scores,label='Train Score')
plt.plot(depths,test_scores,label='Test Score')
plt.xlabel('Depth')
plt.ylabel('Score')
plt.title('Depth vs score at depth of 5 at estimators = 115')
plt.show()
```

```
depth =  3 Train Score 0.8916120853581238 test Score 0.8687934859875491
depth =  9 Train Score 0.9572226298198419 test Score 0.9222953031452904
depth =  11 Train Score 0.9623451340902863 test Score 0.9252318758281279
depth =  15 Train Score 0.9634267621927706 test Score 0.9231288356496615
depth =  20 Train Score 0.9631629153051491 test Score 0.9235051024711141
depth =  35 Train Score 0.9634333127085721 test Score 0.9235601652753184
depth =  50 Train Score 0.9634333127085721 test Score 0.9235601652753184
depth =  70 Train Score 0.9634333127085721 test Score 0.9235601652753184
depth =  130 Train Score 0.9634333127085721 test Score 0.9235601652753184
```



In [106]:

```
from sklearn.metrics import f1_score
from sklearn.ensemble import RandomForestClassifier
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 = RandomForestClassifier(random_state=25,n_jobs=-1)

rf_random = RandomizedSearchCV(clf, param_distributions=param_dist,
                                   n_iter=5,cv=10,scoring='f1',random_state=25,return_t
rain_score=True)

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'])
```

```
mean test scores [0.96225042 0.96215492 0.9605708  0.96194014 0.96330005]
mean train scores [0.96294922 0.96266735 0.96115674 0.96263457 0.96430539]
```

In [107]:

```
print(rf_random.best_estimator_)
```

```
RandomForestClassifier(max_depth=14, min_samples_leaf=28, min_samples_spli
t=111,
                       n_estimators=121, n_jobs=-1, random_state=25)
```

In [108]:

```
clf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=14, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=28, min_samples_split=111,
            min_weight_fraction_leaf=0.0, n_estimators=121, n_jobs=-1,
            oob_score=False, random_state=25, verbose=0, warm_start=False)
```

In [109]:

```
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 0.9652533106548414
Test f1 score 0.9241678239279553
```

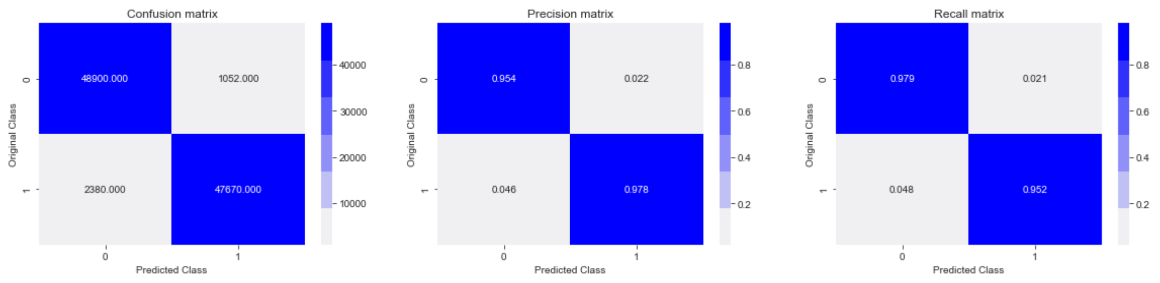
In [110]:

```
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=la
bels)
    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=la
bels)
    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=la
bels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()
```

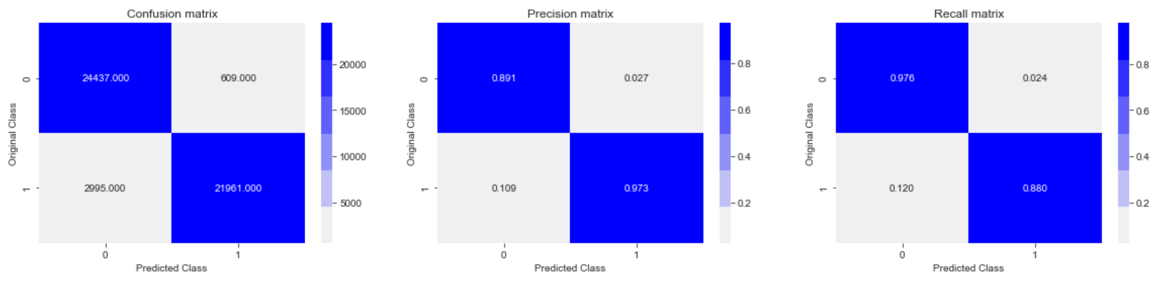
In [111]:

```
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
```

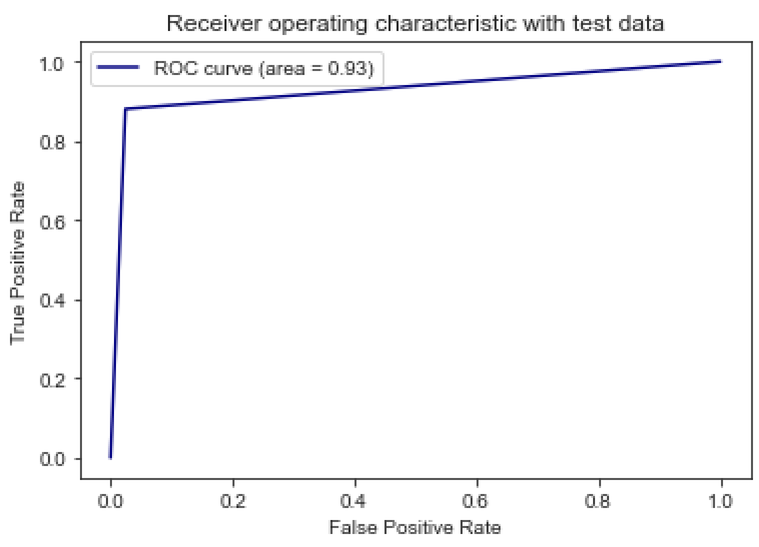


```
Test confusion_matrix
```



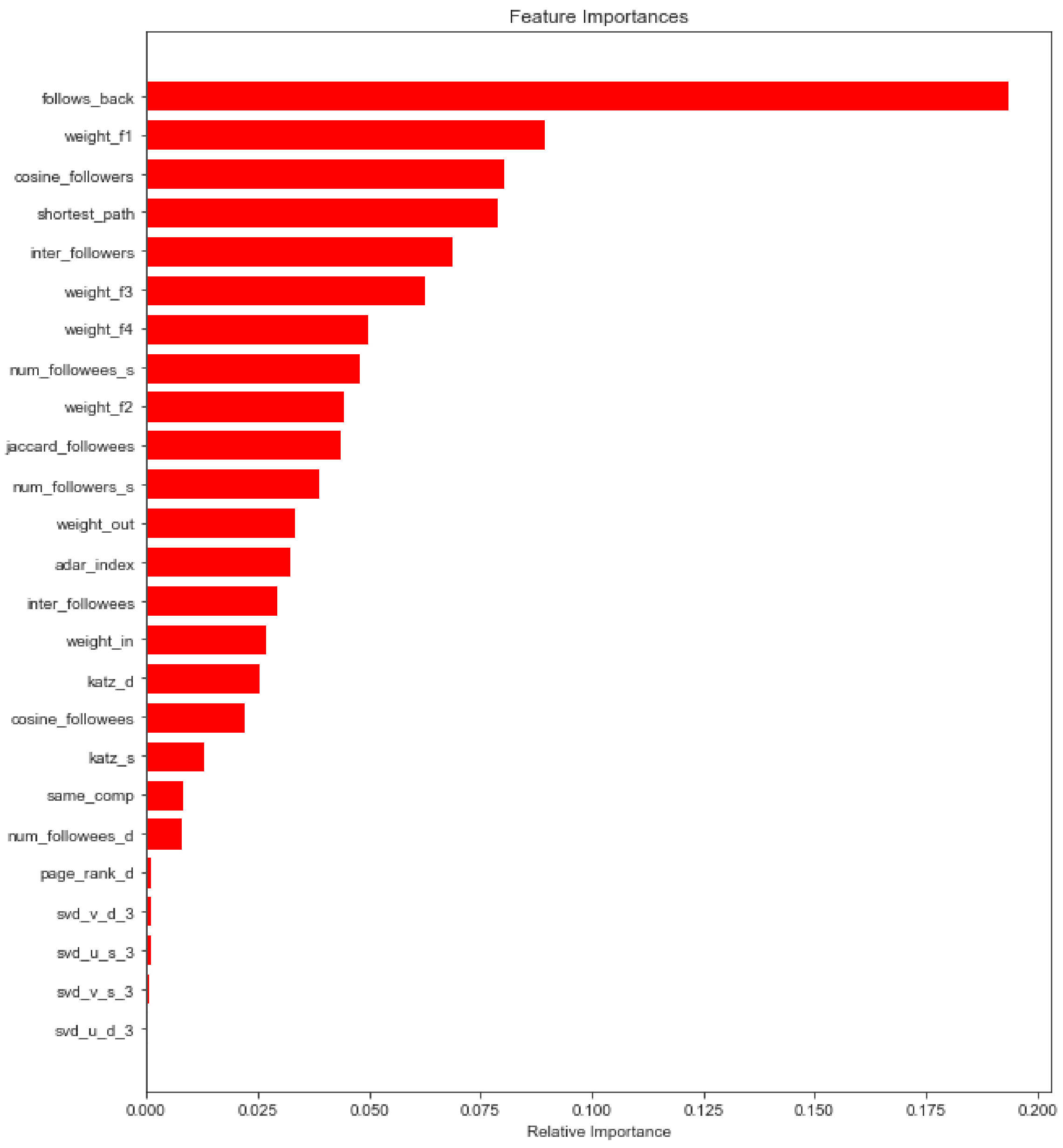
In [112]:

```
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()
```



In [113]:

```
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()
```



In [121]:

```
df_final_train_copy = pd.read_csv('data/after_eda/final_train.csv')
df_final_test_copy = pd.read_csv('data/after_eda/final_test.csv')
```

adding svd_dot

it is calculated as dot product between the source node svd and destinaion node svd feature

In [122]:

```
df_final_train_copy['svd_u_s1*d1']=df_final_train_copy['svd_u_s_1']*df_final_train_copy['svd_u_d_1']
df_final_train_copy['svd_u_s2*d2']=df_final_train_copy['svd_u_s_2']*df_final_train_copy['svd_u_d_2']
df_final_train_copy['svd_u_s3*d3']=df_final_train_copy['svd_u_s_3']*df_final_train_copy['svd_u_d_3']
df_final_train_copy['svd_u_s4*d4']=df_final_train_copy['svd_u_s_4']*df_final_train_copy['svd_u_d_4']
df_final_train_copy['svd_u_s5*d5']=df_final_train_copy['svd_u_s_5']*df_final_train_copy['svd_u_d_5']
df_final_train_copy['svd_u_s6*d6']=df_final_train_copy['svd_u_s_6']*df_final_train_copy['svd_u_d_6']

df_final_train_copy['svd_v_s1*d1']=df_final_train_copy['svd_v_s_1']*df_final_train_copy['svd_v_d_1']
df_final_train_copy['svd_v_s2*d2']=df_final_train_copy['svd_v_s_2']*df_final_train_copy['svd_v_d_2']
df_final_train_copy['svd_v_s3*d3']=df_final_train_copy['svd_v_s_3']*df_final_train_copy['svd_v_d_3']
df_final_train_copy['svd_v_s4*d4']=df_final_train_copy['svd_v_s_4']*df_final_train_copy['svd_v_d_4']
df_final_train_copy['svd_v_s5*d5']=df_final_train_copy['svd_v_s_5']*df_final_train_copy['svd_v_d_5']
df_final_train_copy['svd_v_s6*d6']=df_final_train_copy['svd_v_s_6']*df_final_train_copy['svd_v_d_6']




# ===================================================================================

df_final_test_copy['svd_u_s1*d1']=df_final_test_copy['svd_u_s_1']*df_final_test_copy['svd_u_d_1']
df_final_test_copy['svd_u_s2*d2']=df_final_test_copy['svd_u_s_2']*df_final_test_copy['svd_u_d_2']
df_final_test_copy['svd_u_s3*d3']=df_final_test_copy['svd_u_s_3']*df_final_test_copy['svd_u_d_3']
df_final_test_copy['svd_u_s4*d4']=df_final_test_copy['svd_u_s_4']*df_final_test_copy['svd_u_d_4']
df_final_test_copy['svd_u_s5*d5']=df_final_test_copy['svd_u_s_5']*df_final_test_copy['svd_u_d_5']
df_final_test_copy['svd_u_s6*d6']=df_final_test_copy['svd_u_s_6']*df_final_test_copy['svd_u_d_6']


df_final_test_copy['svd_v_s1*d1']=df_final_test_copy['svd_v_s_1']*df_final_test_copy['svd_v_d_1']
df_final_test_copy['svd_v_s2*d2']=df_final_test_copy['svd_v_s_2']*df_final_test_copy['svd_v_d_2']
df_final_test_copy['svd_v_s3*d3']=df_final_test_copy['svd_v_s_3']*df_final_test_copy['svd_v_d_3']
df_final_test_copy['svd_v_s4*d4']=df_final_test_copy['svd_v_s_4']*df_final_test_copy['svd_v_d_4']
df_final_test_copy['svd_v_s5*d5']=df_final_test_copy['svd_v_s_5']*df_final_test_copy['svd_v_d_5']
df_final_test_copy['svd_v_s6*d6']=df_final_test_copy['svd_v_s_6']*df_final_test_copy['svd_v_d_6']
```

In [123]:

```
df_final_train_copy.head(2)
```

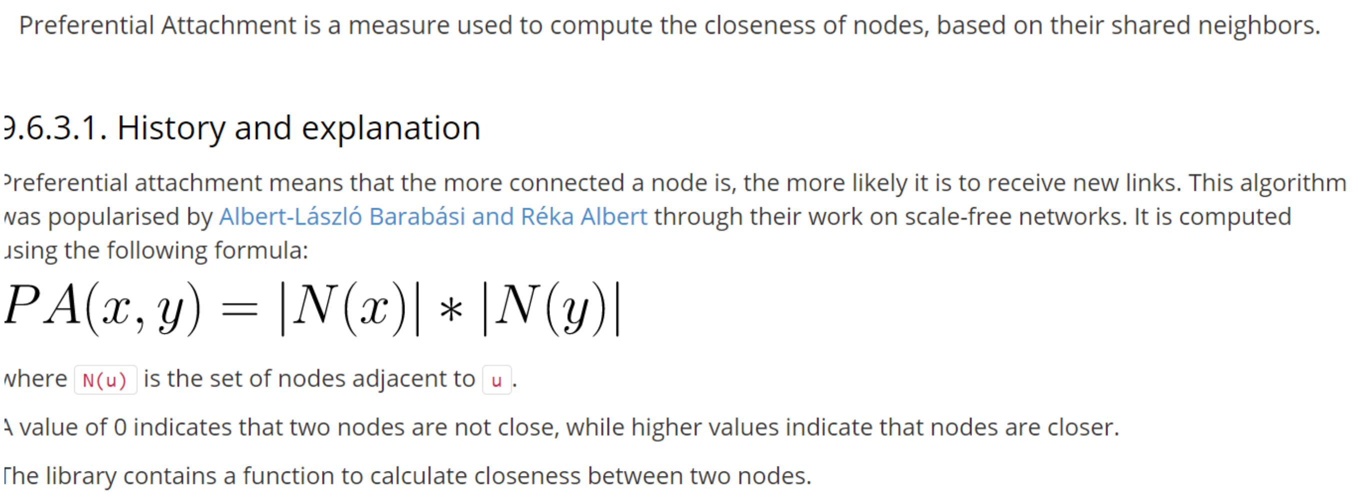
Out[123]:

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

2 rows × 66 columns

Adding Preferential Attachment feature

https://neo4j.com/docs/graph-algorithms/current/labs-algorithms/preferential-attachment/ (https://neo4j.com/docs/graph-algorithms/current/labs-algorithms/preferential-attachment/)



Preferential Attachment is a measure used to compute the closeness of nodes, based on their shared neighbors.

9.6.3.1. History and explanation

Preferential attachment means that the more connected a node is, the more likely it is to receive new links. This algorithm was popularised by Albert-László Barabási and Réka Albert through their work on scale-free networks. It is computed using the following formula:

$$PA(x, y) = |N(x)| * |N(y)|$$

where `N(u)` is the set of nodes adjacent to `u`.

A value of 0 indicates that two nodes are not close, while higher values indicate that nodes are closer.

The library contains a function to calculate closeness between two nodes.

In [124]:

```
def compute_perf_attach(df_final):

    perf_attach_followers=[]
    perf_attach_followees=[]

    for i,row in df_final.iterrows():
        try:
        # compute product of followees of source node and destination node
            fe1 =set(train_graph.successors(row['source_node']))
            fe2 =set(train_graph.successors(row['destination_node']))
            product1  = len(fe1)*len(fe2)
        except:
            fe1=0
            fe2=0
            product1 = 0

    # compute product of followers of source node and destination node
        try:
            fo1 = set(train_graph.predecessors(row['source_node']))
            fo2 =set(train_graph.predecessors(row['destination_node']))
            product2 = len(fo1)*len(fo2)
        except:
            fo1=0
            fo2=0
            product2=0

        perf_attach_followees.append(product1)
        perf_attach_followers.append(product2)

    return perf_attach_followers,perf_attach_followees
```

In [125]:

```

# map the perferential attachment feature to training data and testing data

df_final_train_copy['perf_attachment_followers'],df_final_train_copy['perf_attachment_f
ollowees'] = compute_perf_attach(df_final_train_copy)

df_final_test_copy['perf_attachment_followers'],df_final_test_copy['perf_attachment_fol
lowees'] = compute_perf_attach(df_final_test_copy)
```

In [126]:

```
df_final_train_copy.head(2)
```

Out[126]:

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

2 rows × 68 columns

Xgboost with hyperparameter tuning

In [127]:

```
df_final_train_copy.shape
```

Out[127]:

```
(100002, 68)
```

In [128]:

```
df_final_test_copy.shape
```

Out[128]:

```
(50002, 68)
```

In [129]:

```
import xgboost as xgb
from xgboost import cv
```

In [130]:

```
y_train = df_final_train_copy['indicator_link']
y_test  = df_final_test_copy['indicator_link']

df_final_train_copy.drop(['source_node', 'destination_node','indicator_link'],axis=1,in
place=True)
df_final_test_copy.drop(['source_node', 'destination_node','indicator_link'],axis=1,inp
lace=True)
```

In [133]:

```
xgb_clf = xgb.XGBClassifier()
parameter={ 'learning_rate' :[0.1,0.2,0.3,0.4],
 'n_estimators':[10,50,100,150,200,],
 'alpha':[1e-5, 1e-2, 0.1, 1, 100],
 'lambda':[1e-5, 1e-2, 0.1, 1, 100],
 'max_depth':[5,6,7,8,9],
 'min_child_weight':[1,3,5,7],
 'gamma':[0.1,0.3,0.5,0.7],
 'subsample':[0.5,0.6,0.7,0.8],
 'colsample_bytree':[0.5,0.6,0.7,0.8],
 'objective':['binary:logitraw'],
 'nthread':[4],
 'scale_pos_weight':[1],
 'seed':[27]
}


clf_2 = RandomizedSearchCV(xgb_clf,param_distributions=parameter,cv = 3,scoring='f1',n_
jobs = 5,return_train_score=True)

clf_2.fit(df_final_train_copy,y_train)
```

Out[133]:

```
RandomizedSearchCV(cv=3,
                   estimator=XGBClassifier(base_score=None, booster=None,
                                           colsample_bylevel=None,
                                           colsample_bynode=None,
                                           colsample_bytree=None, gamma=No
ne,
                                           gpu_id=None, importance_type='g
ain',
                                           interaction_constraints=None,
                                           learning_rate=None,
                                           max_delta_step=None, max_depth=
None,
                                           min_child_weight=None, missing=
nan,
                                           monotone_constraints=None,
                                           n_estimators=100,...
                                        'colsample_bytree': [0.5, 0.6, 0.
7,
                                                             0.8],
                                        'gamma': [0.1, 0.3, 0.5, 0.7],
                                        'lambda': [1e-05, 0.01, 0.1, 1, 10
0],
                                        'learning_rate': [0.1, 0.2, 0.3,
0.4],
                                        'max_depth': [5, 6, 7, 8, 9],
                                        'min_child_weight': [1, 3, 5, 7],
                                        'n_estimators': [10, 50, 100, 150,
200],
                                        'nthread': [4],
                                        'objective': ['binary:logitraw'],
                                        'scale_pos_weight': [1], 'seed':
[27],
                                        'subsample': [0.5, 0.6, 0.7, 0.
8]},
                   return_train_score=True, scoring='f1')
```

In [134]:

```
clf_2.best_score_
```

Out[134]:

```
0.9794968908087286
```

In [135]:

```
clf_2.best_params_
```

Out[135]:

```
{'subsample': 0.5,
 'seed': 27,
 'scale_pos_weight': 1,
 'objective': 'binary:logitraw',
 'nthread': 4,
 'n_estimators': 200,
 'min_child_weight': 5,
 'max_depth': 9,
 'learning_rate': 0.1,
 'lambda': 1,
 'gamma': 0.3,
 'colsample_bytree': 0.7,
 'alpha': 0.01}
```

In [136]:

```
xgb_clf =xgb.XGBClassifier(subsample=  0.5, seed= 27,
 scale_pos_weight=1,
 objective = 'binary:logitraw',
 nthread = 4,
 n_estimators = 200,
 min_child_weight =5,
 max_depth= 9,
 learning_rate = 0.1,
 gamma = 0.3,
 colsample_bytree = 0.7,
 reg_lambda=1,
 alpha = 0.01)

xgb_clf.fit(df_final_train_copy,y_train)
y_train_pred = xgb_clf.predict(df_final_train_copy)
y_test_pred = xgb_clf.predict(df_final_test_copy)


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 0.9930341257476001
Test f1 score 0.9278837962174136
```

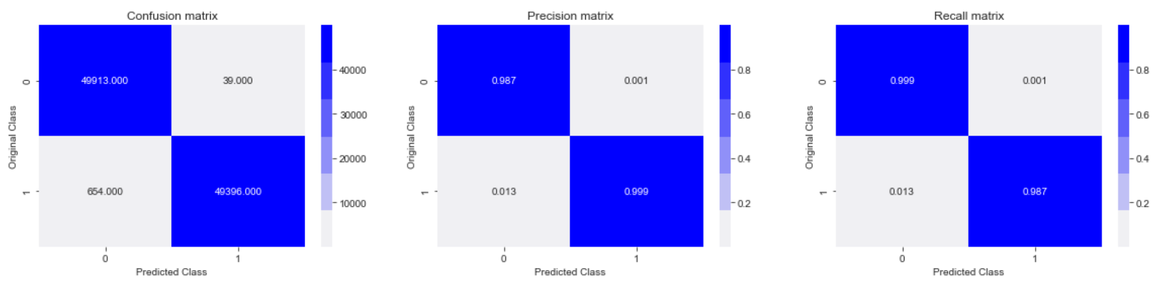
In [137]:

```
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=la
bels)
    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=la
bels)
    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=la
bels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()
```

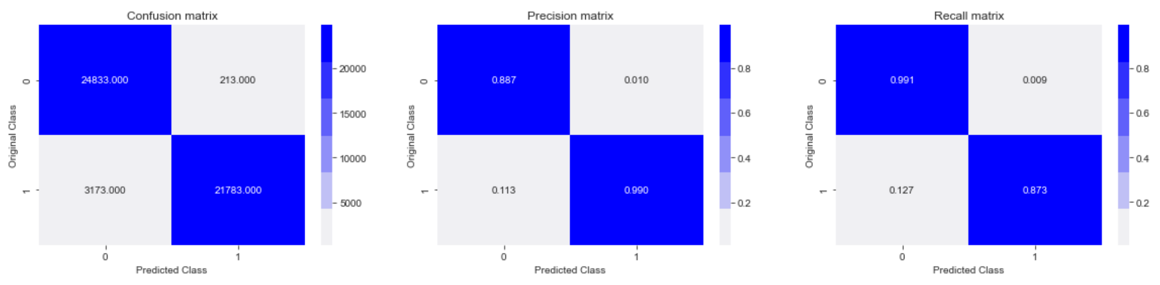
In [138]:

```
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
```

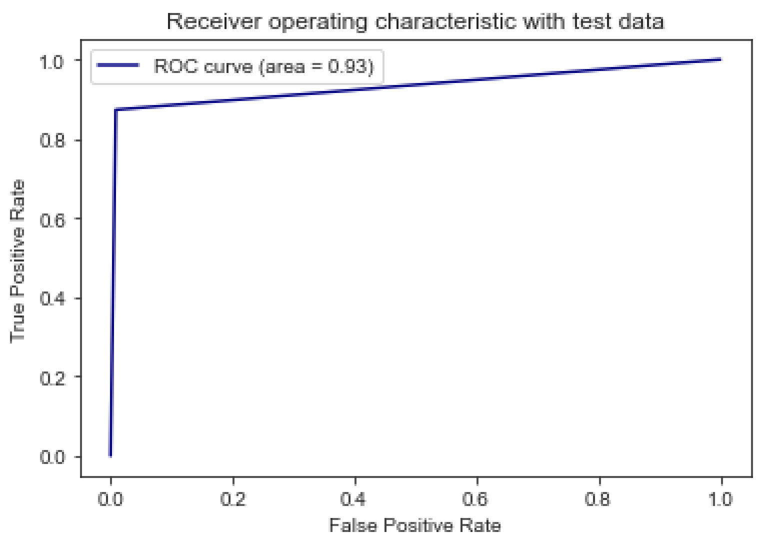


```
Test confusion_matrix
```



In [139]:

```
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()
```



In [147]:

```
features=df_final_train_copy.columns
features
```

Out[147]:

```
Index(['jaccard_followers', 'jaccard_followees', 'cosine_followers',
       'cosine_followees', 'num_followers_s', 'num_followees_s',
       'num_followees_d', 'inter_followers', 'inter_followees', 'adar_inde
x',
       '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_u_s1*d1', 'svd_u_s2*d2', 'svd_u_s3*d3', 'svd_u_s4*d4',
       'svd_u_s5*d5', 'svd_u_s6*d6', 'svd_v_s1*d1', 'svd_v_s2*d2',
       'svd_v_s3*d3', 'svd_v_s4*d4', 'svd_v_s5*d5', 'svd_v_s6*d6',
       'perf_attachment_followers', 'perf_attachment_followees'],
      dtype='object')
```

In [148]:

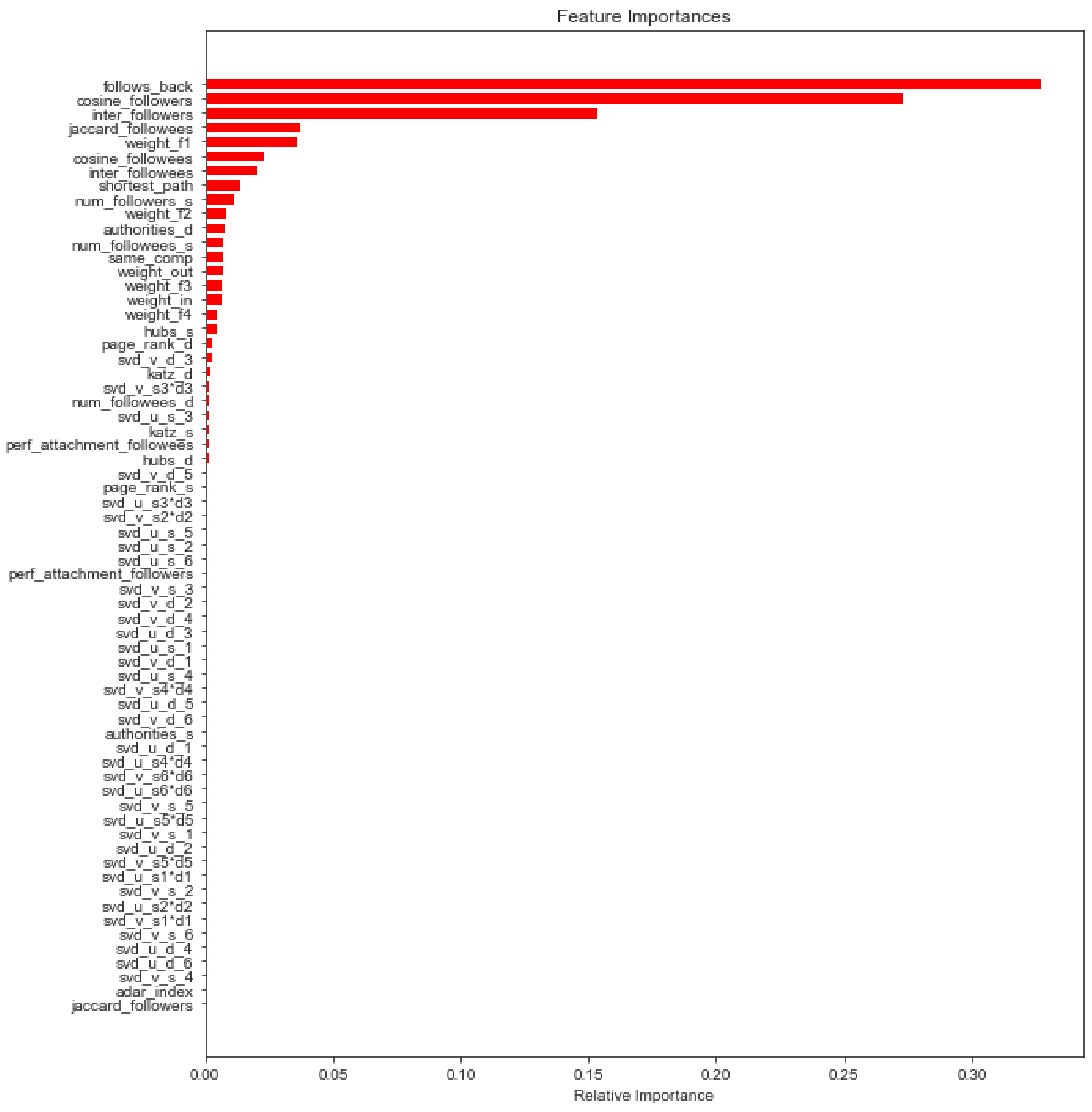
```
df_final_train_copy.shape
```

Out[148]:

```
(100002, 65)
```

In [152]:

```
features = df_final_train_copy.columns
importances = xgb_clf.feature_importances_
indices = (np.argsort(importances))[-65:]
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()
```



In [144]:

```
importances = xgb_clf.feature_importances_
importances
```

Out[144]:

```
array([0.        , 0.03730383, 0.27268806, 0.02316953, 0.01156834,
       0.00715777, 0.00135749, 0.1538193 , 0.02054361, 0.        ,
       0.3271599 , 0.00714733, 0.01368762, 0.00626062, 0.00689581,
       0.03591841, 0.00825653, 0.00651198, 0.00459308, 0.00114378,
       0.00295033, 0.00132711, 0.00200237, 0.00448103, 0.00119776,
       0.00075897, 0.00741148, 0.00086684, 0.00099969, 0.00133104,
       0.00083349, 0.00104382, 0.00095872, 0.00075058, 0.00070594,
       0.00088318, 0.00063263, 0.00076977, 0.00062956, 0.00071494,
       0.00069066, 0.00090857, 0.00060483, 0.00073462, 0.00066351,
       0.00083507, 0.00090416, 0.002632  , 0.00090124, 0.00118151,
       0.00076902, 0.00070387, 0.00067993, 0.00113421, 0.00074836,
       0.00073092, 0.00073937, 0.00067974, 0.00106439, 0.00177611,
       0.00081242, 0.00070589, 0.00074545, 0.00091623, 0.00130568],
      dtype=float32)
```

Conclusion

1) Given problem : we are given a social directed graph dataset and our task is to predict the missing link to recommend user (link prediction)

2) our data contains the 2 columns i.e. source node and destination node . original data cotains only nodes that have an edge between them i.e. edge = 1

3) Then performed EDA to get some insights about data such as 1) number of followers of each person 2) number of people each person following 3) No of persons those are not following anyone are 4) No of persons having zero followers are 5) Max number of followers + followees 6) Min number of followers + followees 7) Number of poeple having min number of followers+following 8) Number of people havign max number of followers+following

4) after performing EDA, then we generated some missing edge which were not present in our data because In Our given data we are given only 9437519 number of pair of node having an edge i.e. edge = 1 But if we want to convert our problem into supervised problem we first has to balanced our data i.e. there should be equal no of datapoints of each class i.e. edge = 0 and edge =1

```
so we created the 9437519 number of pair of node having no edge i.e. edge = 0
```

5) performed featurations i..e created some new feature such as for both train and test data

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
11. adar index
12. is following back
13. belongs to same weakly connect components
14. shortest path between source and destination
15. Weight Features
16. weight of incoming edges
17. weight of outgoing edges
18. weight of incoming edges + weight of outgoing edges
19. weight of incoming edges * weight of outgoing edges
20. 2*weight of incoming edges + weight of outgoing edges
21. weight of incoming edges + 2*weight of outgoing edges
22. Page Ranking of source
23. Page Ranking of dest
24. katz of source
25. katz of dest
26. hubs of source
27. hubs of dest
28. authorities_s of source

29. authorities_s of dest
30. SVD features for both source and destination

6) using all the above created feature build the random forest model by finding the best value of parameter(n_estiamtor , max depth etc.) by performing hyperparamter tuning using randomsearchCv

```
We achieve the F1 score for test data = 0.9241678239279553
F1 score for train data  =  0.9652533106548414
```

7) In order to see increase in model performance we created and added new feature in train data and test data and used xgboost algorithm

```
  New features are :  1) product of source node svd and destination node svd fea
ture
                      2) perfertial attachment feature


 Build the xgboost model using all the previous + new features and perfomed hype
rparamter tuning for xgboost

 using the best value of parameter ( such as n_estimator , max depth , min_child
_weight, lambda, gamma,   colsample_by_Tree) build the final xgboost model which
gave very slight increase in model performance

 i.e. F1 score on test data = 0.9278837962174136
 F1 score on train data = 0.9930341257476001
```

8) So perferntial attachment and product of source node svd and destination svd features does not help in increasing the model performance

9) the most important features are: - follow_back, cosine_followers, inter_followers, jaccard_followees, cosine_followees, weight_f1, inter_followees, shortest_path,

In [ ]:

```

```