In [2]:

```
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
```

In [3]:

```
auth.authenticate_user()
gauth= GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

In [4]:

```
downloaded = drive.CreateFile({'id': '15QwqCxPg6UO-RBNasgMwdRm41LzJQ2q2'})
downloaded.GetContentFile('train.csv')
```

In [5]:

```
import pandas as pd
data = pd.read_csv('train.csv')
data.head()
```

Out[5]:

	source_node	destination_node
0	1	690569
1	1	315892
2	1	189226
3	2	834328
4	2	1615927

In [6]:

```
print('The Data consists of {} data points.' .format(data.shape[0]))
```

```
The Data consists of 9437519 data points.
```

In [7]:

```
import warnings
warnings.filterwarnings("ignore")

import csv
import pandas as pd
import datetime
import time
import numpy as np
import matplotlib
import matplotlib.pylab as plt
import seaborn as sns
from matplotlib import rcParams
from sklearn.cluster import MiniBatchKMeans, KMeans
import math
import pickle
import os

import xgboost as xgb

import warnings
import networkx as nx
import pdb
import pickle
```

In [8]:

```
data.isnull().sum()
```

Out[8]:

```
source_node         0
destination_node    0
dtype: int64
```

In [9]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9437519 entries, 0 to 9437518
Data columns (total 2 columns):
 #   Column            Dtype
---  ------            -----
 0   source_node       int64
 1   destination_node  int64
dtypes: int64(2)
memory usage: 144.0 MB
```

In [10]:

```
print('Number of duplicate Entries : ', data.duplicated().sum())
```

```
Number of duplicate Entries :  0
```

In [11]:

```
data.to_csv('train_woheader.csv',header=False,index=False)
```

In [12]:

```
g = nx.read_edgelist('train_woheader.csv', delimiter=',',create_using=nx.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
```

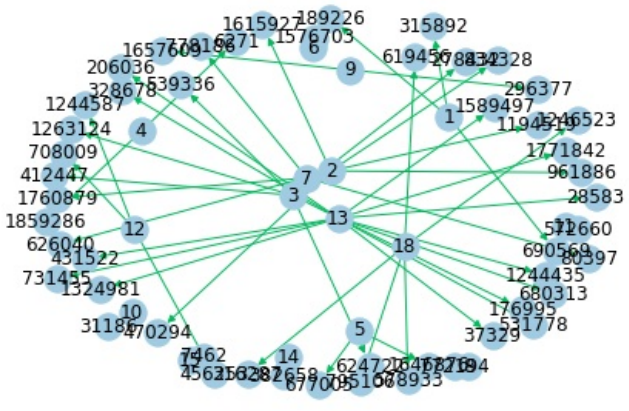
In [13]:

```
pd.read_csv('train.csv', nrows=50).to_csv('train_woheader_sample.csv', header=False, index=False)
```

In [14]:

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

pos=nx.spring_layout(subgraph)
nx.draw(subgraph,pos,node_color='#A0CBE2',edge_color='#00bb5e',width=1,edge_cmap=plt.cm.Blues,with_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
```



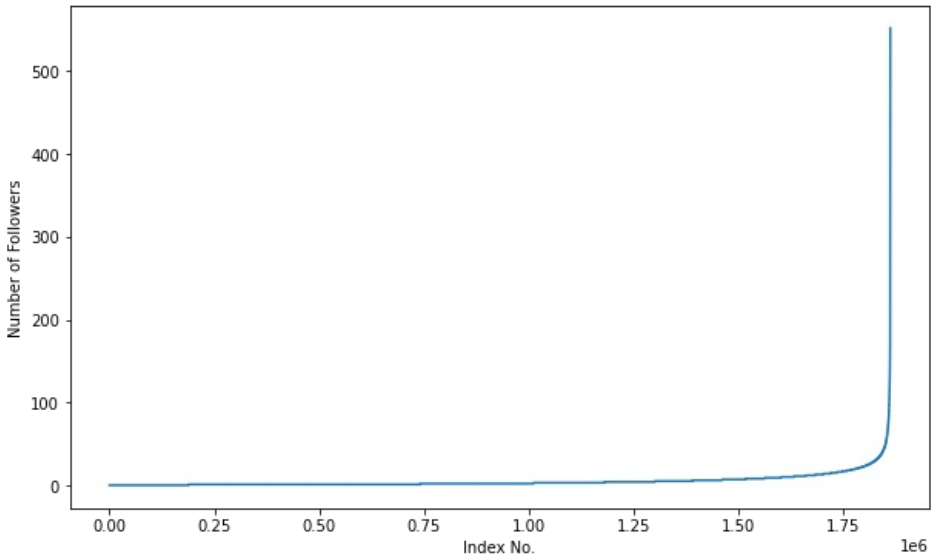
In [15]:

```
print('Number of unique persons :' , len(g.nodes()))
```

```
Number of unique persons : 1862220
```

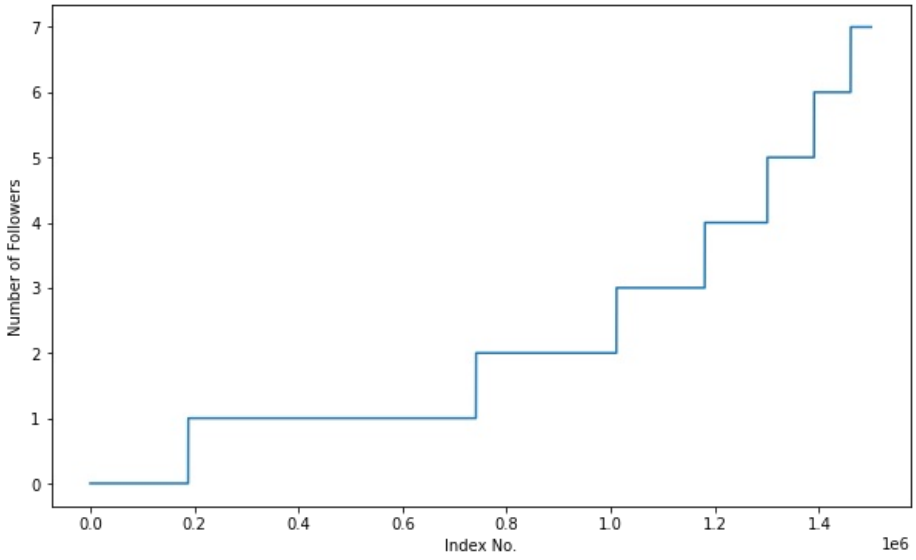
In [16]:

```
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('Number of Followers')
plt.show()
```



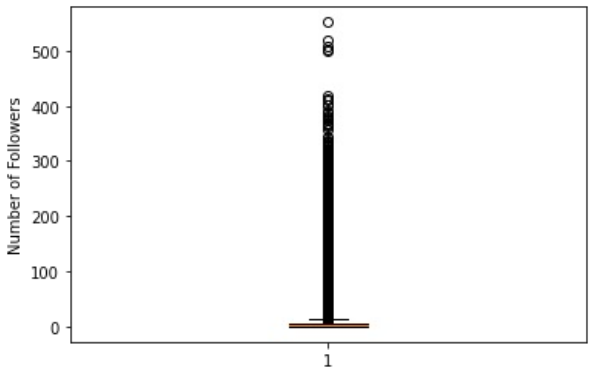
In [17]:

```
plt.figure(figsize=(10,6))
plt.plot(indegree_dist[:1500000])
plt.xlabel('Index No.')
plt.ylabel('Number of Followers')
plt.show()
```



In [18]:

```
plt.boxplot(indegree_dist)
plt.ylabel('Number of Followers')
plt.show()
```



In [19]:

```
for i in range(1,11):
  print(90+i, 'percentile value is ', np.percentile(indegree_dist, 90+i))
```

```
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
```

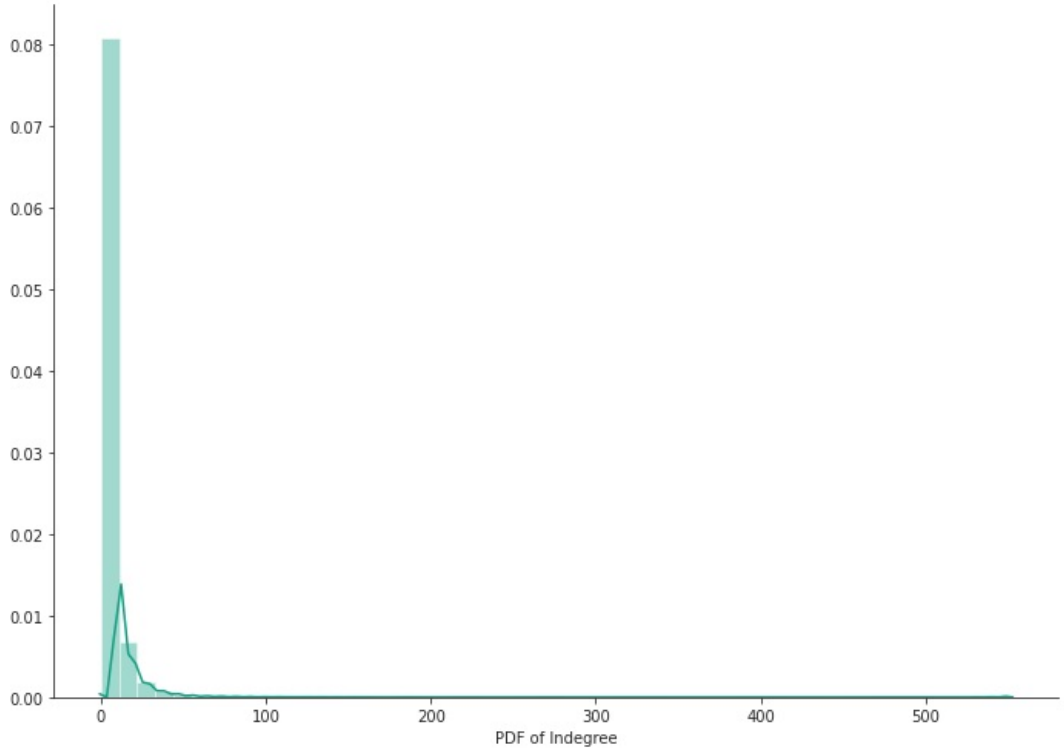
In [20]:

```
for i in range(1,11):
  print(99+(i/10), 'percentile value is ', np.percentile(indegree_dist, (99+(i/10))))
```

```
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
```

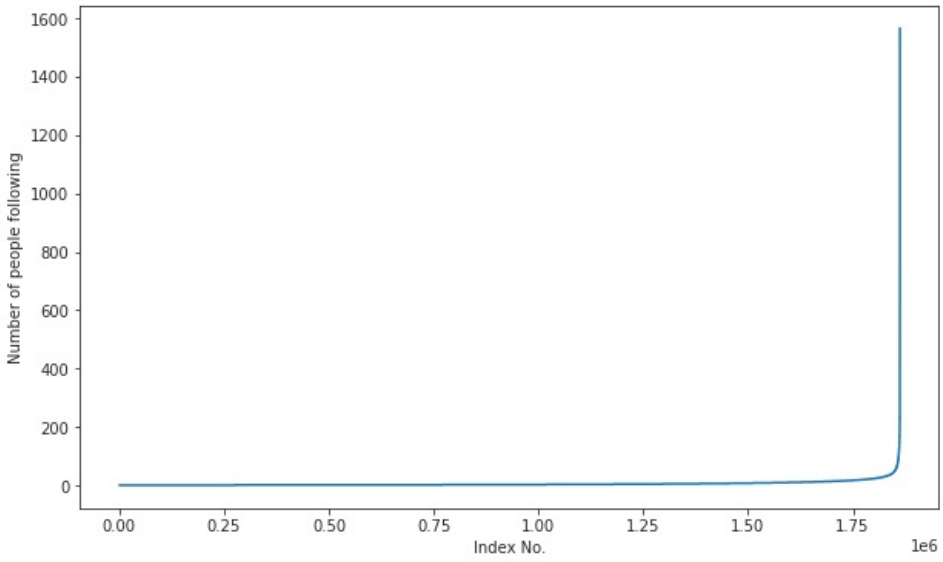
In [21]:

```
%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()
```



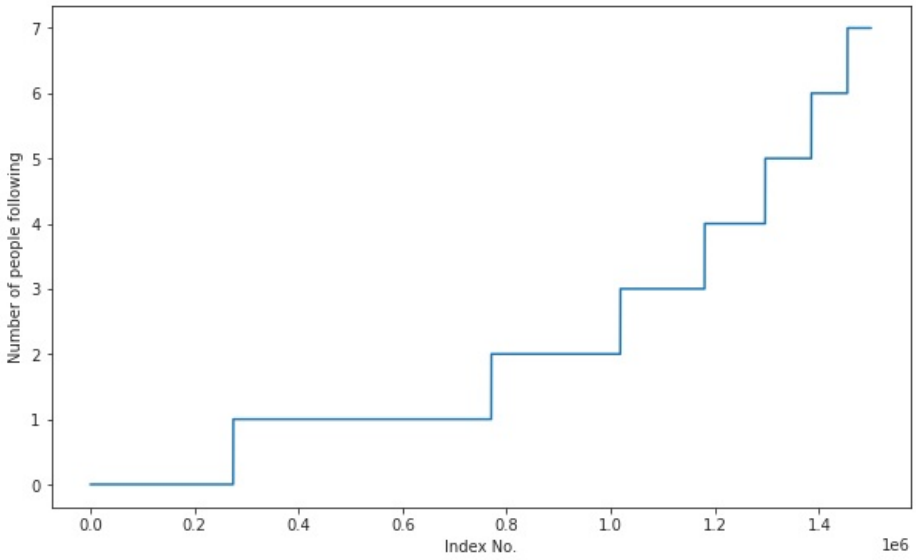
In [22]:

```
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('Number of people following')
plt.show()
```



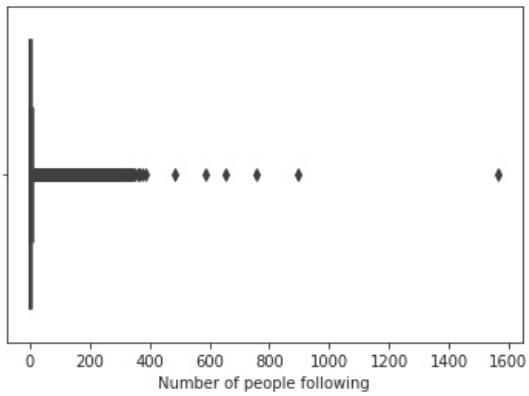
In [23]:

```
plt.figure(figsize=(10,6))
plt.plot(outdegree_dist[:1500000])
plt.xlabel('Index No.')
plt.ylabel('Number of people following')
plt.show()
```



In [24]:

```
sns.boxplot(outdegree_dist)
plt.xlabel('Number of people following')
plt.show()
```



In [25]:

```
for i in range(1,11):
  print(90+i, 'percentile value is ', np.percentile(outdegree_dist, 90+i))
```

```
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
```

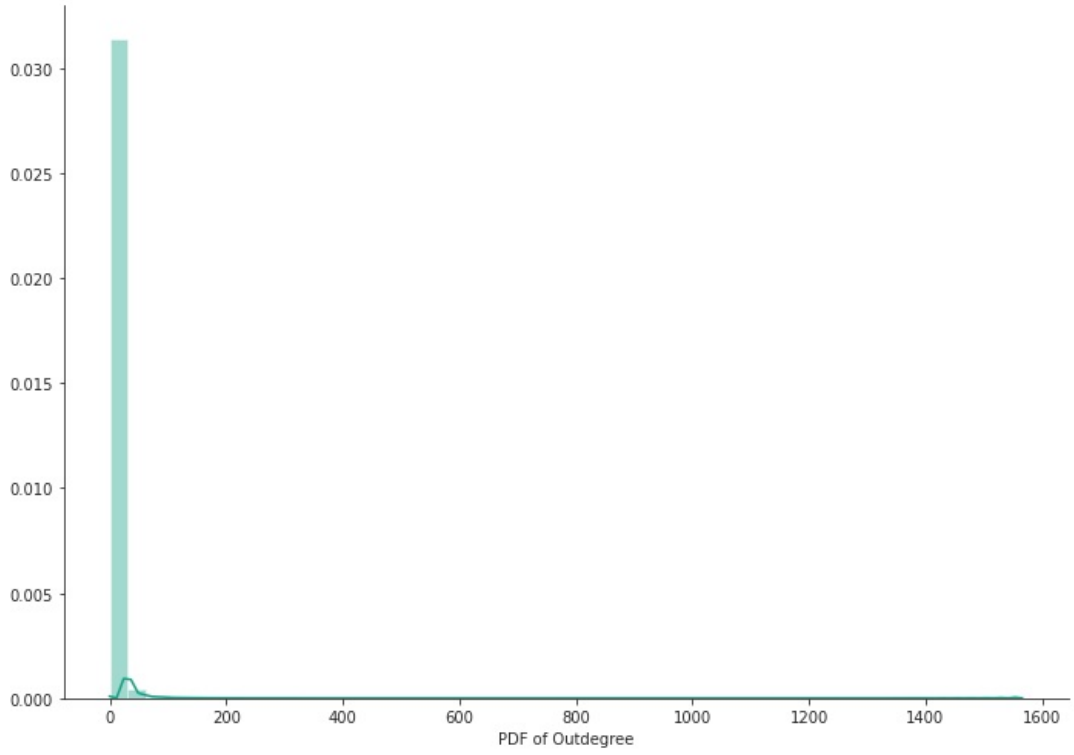
In [26]:

```
for i in range(1,11):
  print(99+(i/10), 'percentile value is ', np.percentile(outdegree_dist, (99+(i/10))))
```

```
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 [27]:

```
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 [28]:

```
print('Number of people who are not following anyone are ', sum(np.array(outdegree_dist) == 0))
print('Therefore almost {:.2f}% of the people donot follow anyone' .format(((sum(np.array(outdegree_dist) == 0))/
len(outdegree_dist))*100))
```

```
Number of people who are not following anyone are  274512
Therefore almost 14.74% of the people donot follow anyone
```

In [29]:

```
print('Number of people with no followers are ', (sum(np.array(indegree_dist) == 0)))
print('Therefore, almost {:.2f}% of the people have no followers ' .format(((sum(np.array(indegree_dist) == 0))/l
en(indegree_dist))*100))
```

```
Number of people with no followers are  188043
Therefore, almost 10.10% of the people have no followers
```

In [30]:

```
count = 0
for i in g.nodes():
  if len(list(g.predecessors(i)))==0:
    if len(list(g.successors(i)))==0:
      count = count +1
print('Number of people who are either not having any follower nor following others are ', count)
```

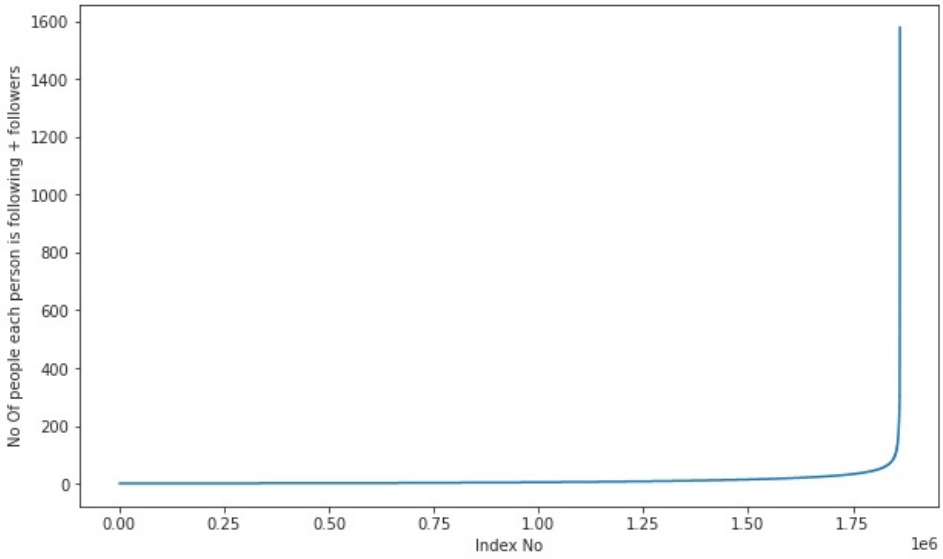
```
Number of people who are either not having any follower nor following others are  0
```

In [31]:

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

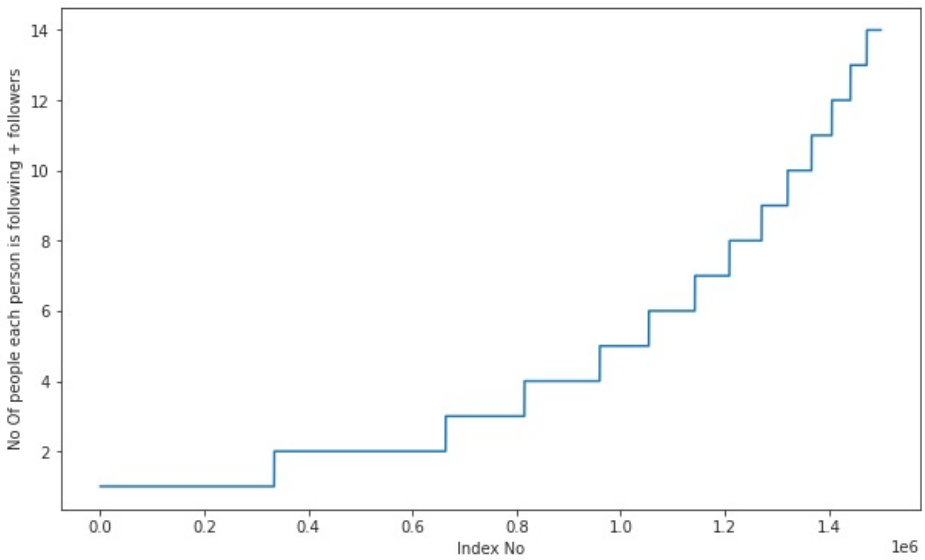
In [32]:

```
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 [33]:

```
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 [34]:

```
### 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 [35]:

```
### 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 [36]:

```
print('Minimum number of follower + following :', in_out_degree.min())
```

```
Minimum number of follower + following : 1
```

In [37]:

```
print(np.sum((in_out_degree == in_out_degree.min())), 'people have minimum number of following + follower.')
```

```
334291 people have minimum number of following + follower.
```

In [38]:

```
print('Maximum number of follower+ following :', in_out_degree.max())
print(np.sum(in_out_degree == in_out_degree.max()), 'people have maximum number of follower+ following.')
```

```
Maximum number of follower+ following : 1579
1 people have maximum number of follower+ following.
```

In [39]:

```
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
```

In [40]:

```
import random
```

In [41]:

```
downloaded = drive.CreateFile({'id': '12EDnzTprsDj7Ask-S13xu-nUQKWRhl3V'})
downloaded.GetContentFile('missing_edges_final.p')
```

In [42]:

```
%%time
###generating bad edges from given graph
import random
if not os.path.isfile('missing_edges_final.p'):
    #getting all set of edges
    r = csv.reader(open('train_woheader.csv','r'))
    edges = dict()
    for edge in r:
        edges[(edge[0], edge[1])] = 1


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

                    missing_edges.add((a,b))
                else:
                    continue
            except:
                    missing_edges.add((a,b))
        else:
            continue
    pickle.dump(missing_edges,open('missing_edges_final.p','wb'))
else:
    missing_edges = pickle.load(open('missing_edges_final.p','rb'))
```

```
CPU times: user 1.99 s, sys: 912 ms, total: 2.91 s
Wall time: 2.9 s
```

In [43]:

```
len(missing_edges)
```

Out[43]:

```
9437519
```

In [44]:

```
from sklearn.model_selection import train_test_split
df_pos = pd.read_csv('train.csv')
df_neg = pd.DataFrame(list(missing_edges), columns=['source_node', 'destination_node'])

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

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=53)
X_train_neg, X_test_neg, y_train_neg, y_test_neg = train_test_split(df_neg, np.zeros(len(df_neg)), test_size=0.2,
random_state=53)

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('train_pos_after_eda.csv',header=False, index=False)
X_test_pos.to_csv('test_pos_after_eda.csv',header=False, index=False)
X_train_neg.to_csv('train_neg_after_eda.csv',header=False, index=False)
X_test_neg.to_csv('test_neg_after_eda.csv',header=False, index=False)
```

```
Number of nodes in 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 [45]:

```
if (os.path.isfile('data/after_eda/train_pos_after_eda.csv')) and (os.path.isfile('data/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=',',create_using=nx.DiGraph(),n
odetype=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))
```

In [46]:

```
# #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')):

X_train_pos = pd.read_csv('train_pos_after_eda.csv', names=['source_node', 'destination_node'])
X_test_pos = pd.read_csv('test_pos_after_eda.csv', names=['source_node', 'destination_node'])
X_train_neg = pd.read_csv('train_neg_after_eda.csv', names=['source_node', 'destination_node'])
X_test_neg = pd.read_csv('test_neg_after_eda.csv', names=['source_node', '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])
X_train = X_train_pos.append(X_train_neg,ignore_index=True)
y_train = np.concatenate((y_train_pos,y_train_neg))
X_test = X_test_pos.append(X_test_neg,ignore_index=True)
y_test = np.concatenate((y_test_pos,y_test_neg))

X_train.to_csv('train_after_eda.csv',header=False,index=False)
X_test.to_csv('test_after_eda.csv',header=False,index=False)
pd.DataFrame(y_train.astype(int)).to_csv('train_y.csv',header=False,index=False)
pd.DataFrame(y_test.astype(int)).to_csv('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 [47]:

```
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 [48]:

```
from pandas import HDFStore,DataFrame
from pandas import read_hdf
from scipy.sparse.linalg import svds, eigs
import gc
from tqdm import tqdm
```

In [49]:

```
train_graph = nx.read_edgelist('train_pos_after_eda.csv', delimiter=',', create_using=nx.DiGraph(), nodetype=int)
print(nx.info(train_graph))
```

```
Name:
Type: DiGraph
Number of nodes: 1781029
Number of edges: 7550015
Average in degree:   4.2391
Average out degree:   4.2391
```

In [50]:

```
def Jaccard_distance_for_followee(a, b):
  try:
    if len(set(train_graph.successors(a))) == 0 | len(set(train_graph.successors(b))) == 0:
      return 0
    sim = (len(set(train_graph.successors(a)).intersection(set(train_graph.successors(b)))))/(len(set(train_graph
.successors(a)).union(set(train_graph.successors(b)))))

  except:
    return 0
  return sim
```

In [51]:

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

```
0.0
```

In [52]:

```
# For Followers
def Jaccard_distance_for_followers(a,b):
  try:
    if len(set(train_graph.predecessors(a))) == 0| len(set(train_graph.predecessors(b))) == 0:
      return 0
    sim = (len(set(train_graph.predecessors(a)).intersection(set(train_graph.predecessors(b)))))/(len(set(train_g
raph.predecessors(a)).union(set(train_graph.predecessors(b)))))

  except:
    return 0
  return sim
```

In [53]:

```
print(Jaccard_distance_for_followers(273084, 1505602))
```

```
0.0
```

In [54]:

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

In [55]:

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

```
0.0
```

In [56]:

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

```
0.0
```

In [57]:

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

        if len(set(train_graph.predecessors(a))) == 0  | len(set(train_graph.predecessors(b))) == 0:
            return 0
        sim = (len(set(train_graph.predecessors(a)).intersection(set(train_graph.predecessors(b)))))/\
                                    (math.sqrt(len(set(train_graph.predecessors(a))))*(len(set(train_graph.prede
cessors(b)))))
        return sim
    except:
        return 0
```

In [58]:

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

```
0.02886751345948129
```

In [59]:

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

```
0
```

In [60]:

```
pr = nx.pagerank(train_graph, alpha=0.85)
pickle.dump(pr, open('page_rank.p', 'wb'))
```

In [61]:

```
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 1.6560093065944057e-07
max 2.48544585932791e-05
mean 5.614731708432429e-07
```

In [62]:

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

```
5.614731708432429e-07
```

In [63]:

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

In [64]:

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

Out[64]:

```
10
```

In [65]:

```
#testing
compute_shortest_path_length(669354,1635354)
```

Out[65]:

```
-1
```

In [66]:

```
#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 [67]:

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

Out[67]:

```
0
```

In [68]:

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

Out[68]:

```
0
```

In [69]:

```
#adar index
def calc_adar_in(a,b):
    sum=0
    try:
        n=list(set(train_graph.successors(a)).union(set(train_graph.predecessors(a))))
        n_1=list(set(train_graph.successors(b)).union(set(train_graph.predecessors(b))))
        n = list(set(n).intersection(set(n_1)))
        if len(n)!=0:
            for i in n:
                sum=sum+(1/np.log10(len(list(set(train_graph.predecessors(i)).union(set(train_graph.successors(i)
))))))
            return sum
        else:
            return 0
    except:
        return 0
```

In [70]:

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

Out[70]:

```
0.6986344247631282
```

In [71]:

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

Out[71]:

```
0
```

In [72]:

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

In [73]:

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

Out[73]:

```
1
```

In [74]:

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

Out[74]:

```
0
```

In [75]:

```
katz = nx.katz.katz_centrality(train_graph, alpha=0.005, beta=1)
pickle.dump(katz, open('katz.p', 'wb'))
```

In [76]:

```
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 0.0007312961508672877
max 0.0034008156667216364
mean 0.0007483176125921135
```

In [77]:

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

```
0.0007483176125921135
```

In [78]:

```
downloaded = drive.CreateFile({'id': '19ejZEGSG54Ja9VJVO5dDmD1DfNgIlvkJ'})
downloaded.GetContentFile('hits.p')
```

In [79]:

```
if not os.path.isfile('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('hits.p','rb'))
```

In [80]:

```
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
```

In [81]:

```
import random
if os.path.isfile('train_after_eda.csv'):
    filename = "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 [82]:

```
if os.path.isfile('train_after_eda.csv'):
    filename = "test_after_eda.csv"
    # you uncomment this line, if you dont know the lentgh of the file name
    # here we have hardcoded the number of lines as 3775008
    # n_test = sum(1 for line in open(filename)) #number of records in file (excludes header)
    n_test = 3775006
    s = 50000 #desired sample size
    skip_test = sorted(random.sample(range(1,n_test+1),n_test-s))
    #https://stackoverflow.com/a/22259008/4084039
```

In [83]:

```
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 [84]:

```
df_final_train = pd.read_csv('train_after_eda.csv', skiprows=skip_train, names=['source_node', 'destination_node'
])
df_final_train['indicator_link'] = pd.read_csv('train_y.csv', skiprows=skip_train, names=['indicator_link'])
print("Our train matrix size ",df_final_train.shape)
df_final_train.head(2)
```

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

Out[84]:

	source_node	destination_node	indicator_link
0	80618	1349607	1
1	1163388	1282921	1

In [85]:

```
df_final_test = pd.read_csv('test_after_eda.csv', skiprows=skip_test, names=['source_node', 'destination_node'])
df_final_test['indicator_link'] = pd.read_csv('test_y.csv', skiprows=skip_test, names=['indicator_link'])
print("Our test matrix size ",df_final_test.shape)
df_final_test.head(2)
```

```
Our test matrix size  (50002, 3)
```

Out[85]:

	source_node	destination_node	indicator_link
0	389533	1809791	1
1	1771515	1533476	1

In [86]:

```
df_final_train
```

Out[86]:

	source_node	destination_node	indicator_link
0	80618	1349607	1
1	1163388	1282921	1
2	1760665	457363	1
3	1861321	1021630	1
4	46083	1846334	1
...	...	...	...
99997	1430051	373512	0
99998	37708	1664541	0
99999	1291064	32063	0
100000	838074	231133	0
100001	501826	771628	0

100002 rows × 3 columns

In [87]:

```
df_final_train['jaccard_followers'] = df_final_train.apply(lambda row:Jaccard_distance_for_followers(row['source_
node'],row['destination_node']),axis=1)
```

In [88]:

```
# if not os.path.isfile('data/fea_sample/storage_sample_stage1.h5'):
#mapping jaccrd followers to train and test data
df_final_test['jaccard_followers'] = df_final_test.apply(lambda row:Jaccard_distance_for_followers(row['source_no
de'],row['destination_node']),axis=1)
#mapping jaccrd followees to train and test data
df_final_train['jaccard_followees'] = df_final_train.apply(lambda row:Jaccard_distance_for_followee(row['source_n
ode'],row['destination_node']),axis=1)
df_final_test['jaccard_followees'] = df_final_test.apply(lambda row:Jaccard_distance_for_followee(row['source_nod
e'],row['destination_node']),axis=1)

#mapping jaccrd followers to train and test data
df_final_train['cosine_followers'] = df_final_train.apply(lambda row:cosine_for_followers(row['source_node'],row[
'destination_node']),axis=1)
df_final_test['cosine_followers'] = df_final_test.apply(lambda row:cosine_for_followers(row['source_node'],row['d
estination_node']),axis=1)
#mapping jaccrd followees to train and test data
df_final_train['cosine_followees'] = df_final_train.apply(lambda row:cosine_for_followees(row['source_node'],row[
'destination_node']),axis=1)
df_final_test['cosine_followees'] = df_final_test.apply(lambda row:cosine_for_followees(row['source_node'],row['d
estination_node']),axis=1)
```

In [89]:

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

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

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

    return num_followers_s, num_followers_d, num_followees_s, num_followees_d, inter_followers, inter_followees
```

In [90]:

```
df_final_train['num_followers_s'], df_final_train['num_followers_d'], \
df_final_train['num_followees_s'], df_final_train['num_followees_d'], \
df_final_train['inter_followers'], df_final_train['inter_followees']= compute_features_stage1(df_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_features_stage1(df_final_test)

hdf = HDFStore('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()
```

In [91]:

```
#mapping adar index on train
df_final_train['adar_index'] = df_final_train.apply(lambda row: calc_adar_in(row['source_node'],row['destination_
node']),axis=1)
#mapping adar index on test
df_final_test['adar_index'] = df_final_test.apply(lambda row: calc_adar_in(row['source_node'],row['destination_no
de']),axis=1)
#-----------------------------------------------------------------------------------------------------------
#mapping followback or not on train
df_final_train['follows_back'] = df_final_train.apply(lambda row: follows_back(row['source_node'],row['destinatio
n_node']),axis=1)
#mapping followback or not on test
df_final_test['follows_back'] = df_final_test.apply(lambda row: follows_back(row['source_node'],row['destination_
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['destin
ation_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['destinat
ion_node']),axis=1)

#-----------------------------------------------------------------------------------------------------------
#mapping shortest path on train
df_final_train['shortest_path'] = df_final_train.apply(lambda row: compute_shortest_path_length(row['source_node'
],row['destination_node']),axis=1)
#mapping shortest path on test
df_final_test['shortest_path'] = df_final_test.apply(lambda row: compute_shortest_path_length(row['source_node'],
row['destination_node']),axis=1)
hdf = HDFStore('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()
```

In [92]:

```
#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%|██████████| 1781029/1781029 [00:16<00:00, 110962.81it/s]
```

In [93]:

```
#mapping to pandas train
df_final_train['weight_in'] = df_final_train.destination_node.apply(lambda x: Weight_in.get(x,mean_weight_in))
df_final_train['weight_out'] = df_final_train.source_node.apply(lambda x: Weight_out.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_out)
df_final_test['weight_f4'] = (1*df_final_test.weight_in + 2*df_final_test.weight_out)
```

In [94]:

```
df_final_train
```

Out[94]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	num_follo
0	80618	1349607	1	0.000000	0.000000	0.000000	0.000000	
1	1163388	1282921	1	0.058824	0.045455	0.031250	0.099015	
2	1760665	457363	1	0.200000	0.240000	0.129219	0.426401	
3	1861321	1021630	1	0.037736	0.035714	0.014493	0.078750	
4	46083	1846334	1	0.000000	0.111111	0.000000	0.230940	
...	...	...	...	...	...	...	...	
99997	1430051	373512	0	0.000000	0.000000	0.000000	0.000000	
99998	37708	1664541	0	0.000000	0.000000	0.000000	0.000000	
99999	1291064	32063	0	0.000000	0.000000	0.000000	0.000000	
100000	838074	231133	0	0.000000	0.000000	0.000000	0.000000	
100001	501826	771628	0	0.000000	0.000000	0.000000	0.000000	

100002 rows × 23 columns

In [95]:

```
df_final_train['PA_followers'] = df_final_train['num_followers_s'] * df_final_train['num_followers_d']
df_final_train['PA_followees'] = df_final_train['num_followees_s'] * df_final_train['num_followees_d']

# For Test Data
df_final_test['PA_followers'] = df_final_test['num_followers_s'] * df_final_test['num_followers_d']
df_final_test['PA_followees'] = df_final_test['num_followees_s'] * df_final_test['num_followees_d']
```

In [96]:

```
downloaded = drive.CreateFile({'id':'1pP2bGjzzqVdCIF9AMzVOSTErpUpDP_La'})
downloaded.GetContentFile('storage_sample_stage3.h5')
```

In [97]:

```
if not os.path.isfile('storage_sample_stage3.h5'):

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

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

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

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

    #Hits algorithm score for source and destination in Train and test
    #if anything not there in train graph then.destination_node.apply(lambda x: hits[0].get(x,0))

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

    #Hits algorithm score for source and destination in Train and Test
    #if anything not there in train graph then adding 0
    df_final_train['authorities_s'] = df_final_train.source_node.apply(lambda x: hits[1].get(x,0))
    df_final_train['authorities_d'] = df_final_train.destination_node.apply(lambda x: hits[1].get(x,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: hits[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('storage_sample_stage3.h5', 'train_df',mode='r')
    df_final_test = read_hdf('storage_sample_stage3.h5', 'test_df',mode='r')
```

In [98]:

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

In [99]:

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

In [100]:

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

In [101]:

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

```
Adjacency matrix Shape (1781029, 1781029)
U Shape (1781029, 6)
V Shape (6, 1781029)
s Shape (6,)
```

In [102]:

```
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', 'svd_u_s_6']] = \
    df_final_train.source_node.apply(lambda x: svd(x, U)).apply(pd.Series)

    df_final_train[['svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3', 'svd_u_d_4', 'svd_u_d_5','svd_u_d_6']] = \
    df_final_train.destination_node.apply(lambda x: svd(x, U)).apply(pd.Series)
    #================================================================================================

    df_final_train[['svd_v_s_1','svd_v_s_2', 'svd_v_s_3', 'svd_v_s_4', 'svd_v_s_5', 'svd_v_s_6',]] = \
    df_final_train.source_node.apply(lambda x: svd(x, V.T)).apply(pd.Series)

    df_final_train[['svd_v_d_1', 'svd_v_d_2', 'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5','svd_v_d_6']] = \
    df_final_train.destination_node.apply(lambda x: svd(x, V.T)).apply(pd.Series)
    #================================================================================================

    df_final_test[['svd_u_s_1', 'svd_u_s_2','svd_u_s_3', 'svd_u_s_4', 'svd_u_s_5', 'svd_u_s_6']] = \
    df_final_test.source_node.apply(lambda x: svd(x, U)).apply(pd.Series)

    df_final_test[['svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3', 'svd_u_d_4', 'svd_u_d_5','svd_u_d_6']] = \
    df_final_test.destination_node.apply(lambda x: svd(x, U)).apply(pd.Series)

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

    df_final_test[['svd_v_s_1','svd_v_s_2', 'svd_v_s_3', 'svd_v_s_4', 'svd_v_s_5', 'svd_v_s_6',]] = \
    df_final_test.source_node.apply(lambda x: svd(x, V.T)).apply(pd.Series)

    df_final_test[['svd_v_d_1', 'svd_v_d_2', 'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5','svd_v_d_6']] = \
    df_final_test.destination_node.apply(lambda x: svd(x, V.T)).apply(pd.Series)
    #================================================================================================

    hdf = HDFStore('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()
```

In [103]:

```
df_final_train
```

Out[103]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	num_fo
0	273084	1505602	1	0	0.000000	0.000000	0.000000	
1	832016	1543415	1	0	0.187135	0.028382	0.343828	
2	1325247	760242	1	0	0.369565	0.156957	0.566038	
3	1368400	1006992	1	0	0.000000	0.000000	0.000000	
4	140165	1708748	1	0	0.000000	0.000000	0.000000	
...	...	...	...	...	...	...	...	
99997	139353	893843	0	0	0.000000	0.000000	0.000000	
99998	910842	704068	0	0	0.000000	0.000000	0.000000	
99999	794228	1172755	0	0	0.000000	0.000000	0.000000	
100000	949992	1854931	0	0	0.000000	0.000000	0.000000	
100001	1642037	1090977	0	0	0.000000	0.000000	0.000000	

100002 rows x 54 columns

In [104]:

```
df_final_train =  df_final_train.drop(['num_followers_s','num_followees_d', 'num_followees_s' ], axis=1)
```

In [105]:

```
df_final_train = df_final_train.drop(['inter_followers', 'inter_followees'], axis=1)
df_final_train
```

Out[105]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	adar_inde
0	273084	1505602	1	0	0.000000	0.000000	0.000000	0.00000
1	832016	1543415	1	0	0.187135	0.028382	0.343828	16.36291
2	1325247	760242	1	0	0.369565	0.156957	0.566038	10.99182
3	1368400	1006992	1	0	0.000000	0.000000	0.000000	0.00000
4	140165	1708748	1	0	0.000000	0.000000	0.000000	0.00000
...	...	...	...	...	...	...	...	.
99997	139353	893843	0	0	0.000000	0.000000	0.000000	0.00000
99998	910842	704068	0	0	0.000000	0.000000	0.000000	0.00000
99999	794228	1172755	0	0	0.000000	0.000000	0.000000	0.00000
100000	949992	1854931	0	0	0.000000	0.000000	0.000000	0.00000
100001	1642037	1090977	0	0	0.000000	0.000000	0.000000	0.00000

100002 rows × 49 columns

In [106]:

```
df_final_train['num_followers_s'], df_final_train['num_followers_d'], \
df_final_train['num_followees_s'], df_final_train['num_followees_d'], \
df_final_train['inter_followers'], df_final_train['inter_followees']= compute_features_stage1(df_final_train)
df_final_train
```

Out[106]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	adar_in
0	273084	1505602	1	0	0.000000	0.000000	0.000000	0.000
1	832016	1543415	1	0	0.187135	0.028382	0.343828	16.362
2	1325247	760242	1	0	0.369565	0.156957	0.566038	10.991
3	1368400	1006992	1	0	0.000000	0.000000	0.000000	0.000
4	140165	1708748	1	0	0.000000	0.000000	0.000000	0.000
...	...	...	...	...	...	...	...	
99997	139353	893843	0	0	0.000000	0.000000	0.000000	0.000
99998	910842	704068	0	0	0.000000	0.000000	0.000000	0.000
99999	794228	1172755	0	0	0.000000	0.000000	0.000000	0.000
100000	949992	1854931	0	0	0.000000	0.000000	0.000000	0.000
100001	1642037	1090977	0	0	0.000000	0.000000	0.000000	0.000

100002 rows × 55 columns

In [107]:

```
df_final_test = df_final_test.drop(['num_followers_s','num_followees_d', 'num_followees_s', 'inter_followers', 'inter_followees' ], axis=1)
# df_final_test
```

In [108]:

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

Out[108]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	adar_index
0	848424	784690	1	0	0.0	0.029161	0.000000	0.000000
1	483294	1255532	1	0	0.0	0.000000	0.000000	0.000000
2	626190	1729265	1	0	0.0	0.000000	0.000000	0.000000
3	947219	425228	1	0	0.0	0.000000	0.000000	0.000000
4	991374	975044	1	0	0.2	0.042767	0.347833	6.136433
...	...	...	...	...	...	...	...	...
49997	1167544	310247	0	0	0.0	0.000000	0.000000	0.000000
49998	656027	1761965	0	0	0.0	0.000000	0.000000	0.000000
49999	1304926	958643	0	0	0.0	0.000000	0.000000	0.000000
50000	773347	1488855	0	0	0.0	0.000000	0.000000	0.000000
50001	1253866	281538	0	0	0.0	0.000000	0.000000	0.000000

50002 rows × 55 columns

In [109]:

```
df_final_train['PA_followers'] = df_final_train['num_followers_s'] * df_final_train['num_followers_d']
df_final_train['PA_followees'] = df_final_train['num_followees_s'] * df_final_train['num_followees_d']

# For Test Data
df_final_test['PA_followers'] = df_final_test['num_followers_s'] * df_final_test['num_followers_d']
df_final_test['PA_followees'] = df_final_test['num_followees_s'] * df_final_test['num_followees_d']
df_final_train
```

Out[109]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	adar_in
0	273084	1505602	1	0	0.000000	0.000000	0.000000	0.000
1	832016	1543415	1	0	0.187135	0.028382	0.343828	16.362
2	1325247	760242	1	0	0.369565	0.156957	0.566038	10.991
3	1368400	1006992	1	0	0.000000	0.000000	0.000000	0.000
4	140165	1708748	1	0	0.000000	0.000000	0.000000	0.000
...	...	...	...	...	...	...	...	
99997	139353	893843	0	0	0.000000	0.000000	0.000000	0.000
99998	910842	704068	0	0	0.000000	0.000000	0.000000	0.000
99999	794228	1172755	0	0	0.000000	0.000000	0.000000	0.000
100000	949992	1854931	0	0	0.000000	0.000000	0.000000	0.000
100001	1642037	1090977	0	0	0.000000	0.000000	0.000000	0.000

100002 rows × 57 columns

In [110]:

```
#for train datasets
s1,s2,s3,s4,s5,s6=df_final_train['svd_u_s_1'],df_final_train['svd_u_s_2'],df_final_train['svd_u_s_3'],df_final_tr
ain['svd_u_s_4'],df_final_train['svd_u_s_5'],df_final_train['svd_u_s_6']
s7,s8,s9,s10,s11,s12=df_final_train['svd_v_s_1'],df_final_train['svd_v_s_2'],df_final_train['svd_v_s_3'],df_final
_train['svd_v_s_4'],df_final_train['svd_v_s_5'],df_final_train['svd_v_s_6']

d1,d2,d3,d4,d5,d6=df_final_train['svd_u_d_1'],df_final_train['svd_u_d_2'],df_final_train['svd_u_d_3'],df_final_tr
ain['svd_u_d_4'],df_final_train['svd_u_d_5'],df_final_train['svd_u_d_6']
d7,d8,d9,d10,d11,d12=df_final_train['svd_v_d_1'],df_final_train['svd_v_d_2'],df_final_train['svd_v_d_3'],df_final
_train['svd_v_d_4'],df_final_train['svd_v_d_5'],df_final_train['svd_v_d_6']

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

In [111]:

```
#for test dataset
s1,s2,s3,s4,s5,s6=df_final_test['svd_u_s_1'],df_final_test['svd_u_s_2'],df_final_test['svd_u_s_3'],df_final_test[
'svd_u_s_4'],df_final_test['svd_u_s_5'],df_final_test['svd_u_s_6']
s7,s8,s9,s10,s11,s12=df_final_test['svd_v_s_1'],df_final_test['svd_v_s_2'],df_final_test['svd_v_s_3'],df_final_te
st['svd_v_s_4'],df_final_test['svd_v_s_5'],df_final_test['svd_v_s_6']

d1,d2,d3,d4,d5,d6=df_final_test['svd_u_d_1'],df_final_test['svd_u_d_2'],df_final_test['svd_u_d_3'],df_final_test[
'svd_u_d_4'],df_final_test['svd_u_d_5'],df_final_test['svd_u_d_6']
d7,d8,d9,d10,d11,d12=df_final_test['svd_v_d_1'],df_final_test['svd_v_d_2'],df_final_test['svd_v_d_3'],df_final_te
st['svd_v_d_4'],df_final_test['svd_v_d_5'],df_final_test['svd_v_d_6']

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

In [112]:

```
df_final_train
```

Out[112]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	adar_in
0	273084	1505602	1	0	0.000000	0.000000	0.000000	0.000
1	832016	1543415	1	0	0.187135	0.028382	0.343828	16.362
2	1325247	760242	1	0	0.369565	0.156957	0.566038	10.991
3	1368400	1006992	1	0	0.000000	0.000000	0.000000	0.000
4	140165	1708748	1	0	0.000000	0.000000	0.000000	0.000
...	...	...	...	...	...	...	...	
99997	139353	893843	0	0	0.000000	0.000000	0.000000	0.000
99998	910842	704068	0	0	0.000000	0.000000	0.000000	0.000
99999	794228	1172755	0	0	0.000000	0.000000	0.000000	0.000
100000	949992	1854931	0	0	0.000000	0.000000	0.000000	0.000
100001	1642037	1090977	0	0	0.000000	0.000000	0.000000	0.000

100002 rows × 58 columns

In [113]:

```
# #reading
# from pandas import read_hdf
# df_final_train = read_hdf('storage_sample_stage4.h5', 'train_df',mode='r')
# df_final_test = read_hdf('storage_sample_stage4.h5', 'test_df',mode='r')
```

In [114]:

```
df_final_train.columns
```

Out[114]:

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

In [115]:

```
y_train = df_final_train['indicator_link']
y_test = df_final_test['indicator_link']
```

In [116]:

```
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 [117]:

```
from scipy.sparse.linalg import svds, eigs
import gc
from tqdm import tqdm
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
```

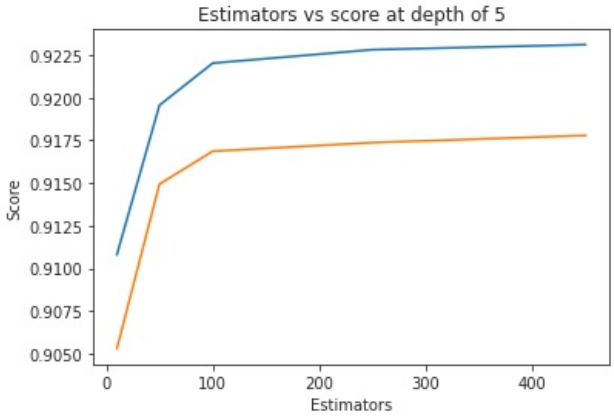
In [118]:

```
estimators = [10,50,100,250,450]
train_scores = []
test_scores = []
for i in estimators:
    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,verbose=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('Estimators = ',i,'Train Score',train_sc,'test Score',test_sc)
plt.plot(estimators,train_scores,label='Train Score')
plt.plot(estimators,test_scores,label='Test Score')
plt.xlabel('Estimators')
plt.ylabel('Score')
plt.title('Estimators vs score at depth of 5')
```

```
Estimators =  10 Train Score 0.9107890699506572 test Score 0.905276303465501
Estimators =  50 Train Score 0.9195609935409337 test Score 0.9149402978777267
Estimators =  100 Train Score 0.9220290037039364 test Score 0.9168663052158198
Estimators =  250 Train Score 0.9228184417590216 test Score 0.9173704394475993
Estimators =  450 Train Score 0.9231138374228008 test Score 0.9177909997485963
```

Out[118]:

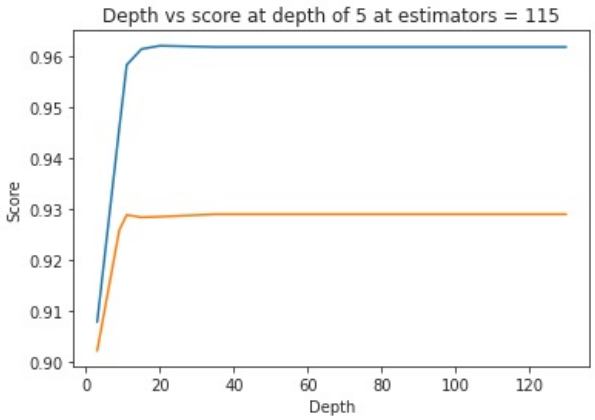
```
Text(0.5, 1.0, 'Estimators vs score at depth of 5')
```



In [119]:

```
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,verbose=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 0.9077319534707036 test Score 0.9020862136563423
depth =  9 Train Score 0.9458729405942635 test Score 0.9257943099859117
depth =  11 Train Score 0.9581981099340723 test Score 0.9287529135081787
depth =  15 Train Score 0.9612950489408029 test Score 0.9282773109243696
depth =  20 Train Score 0.9619464522040111 test Score 0.9284017074247745
depth =  35 Train Score 0.9617018675306416 test Score 0.9288794463493131
depth =  50 Train Score 0.9617018675306416 test Score 0.9288794463493131
depth =  70 Train Score 0.9617018675306416 test Score 0.9288794463493131
depth =  130 Train Score 0.9617018675306416 test Score 0.9288794463493131
```



In [120]:

```
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_train_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 [0.96036847 0.96011413 0.95603455 0.95935674 0.9622419 ]
mean train scores [0.96157077 0.96087157 0.95650894 0.9601801  0.96339082]
```

In [121]:

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

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=14, max_features='auto',
                       max_leaf_nodes=None, max_samples=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 [122]:

```
clf = RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=14, max_features='auto',
                       max_leaf_nodes=None, max_samples=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 [123]:

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

In [124]:

```
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 0.9632619037937624
Test f1 score 0.9303847929167891
```

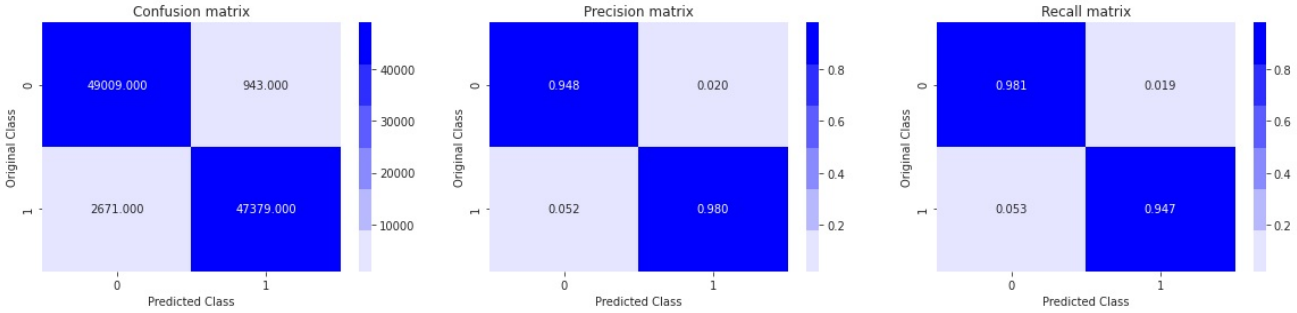
In [125]:

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

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

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

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

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

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

    plt.show()
```

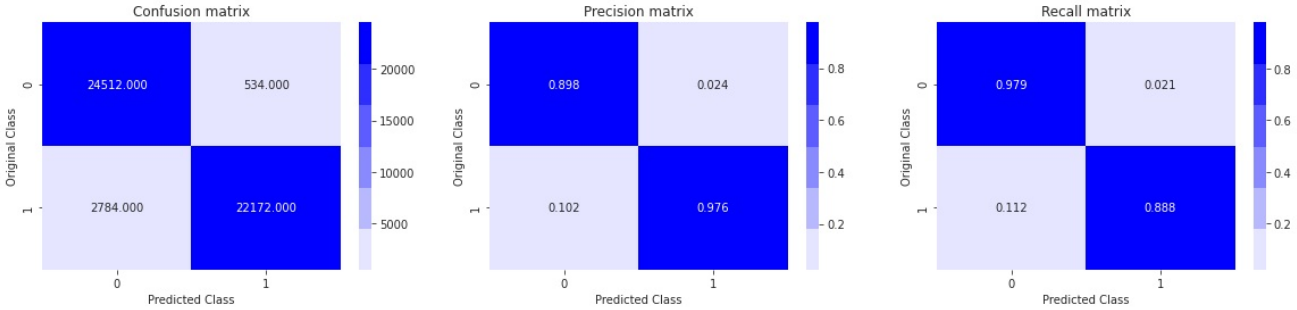
In [126]:

```
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
```

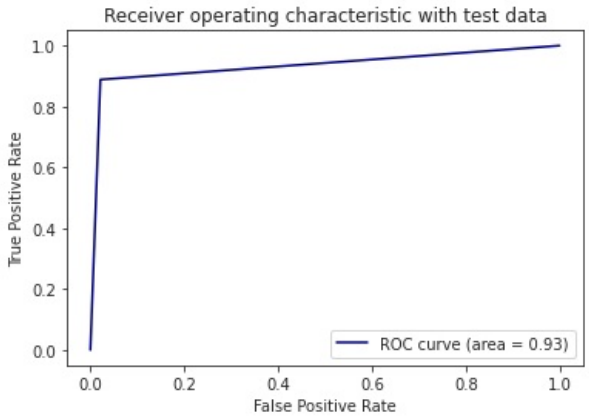



```
Test confusion_matrix
```



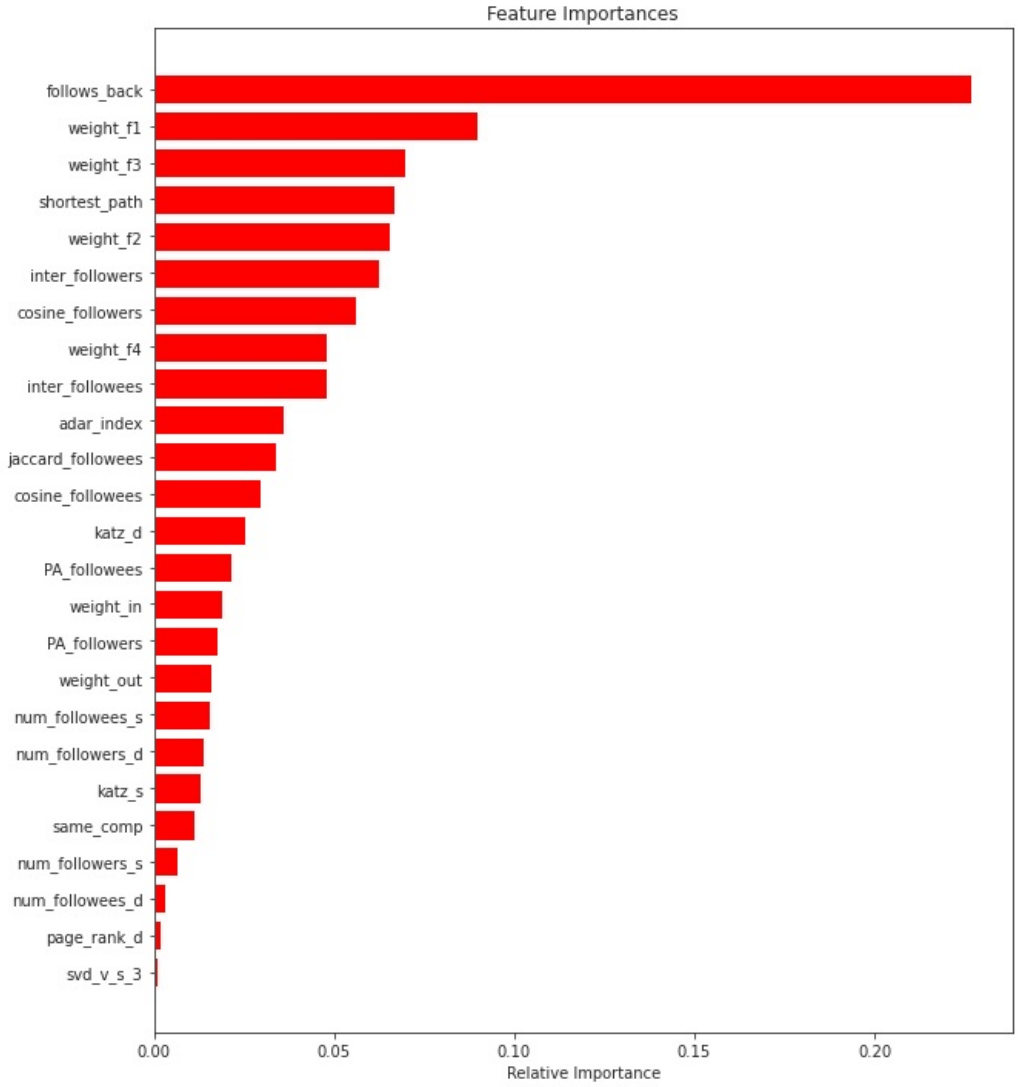
In [127]:

```
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 [128]:

```
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()
```



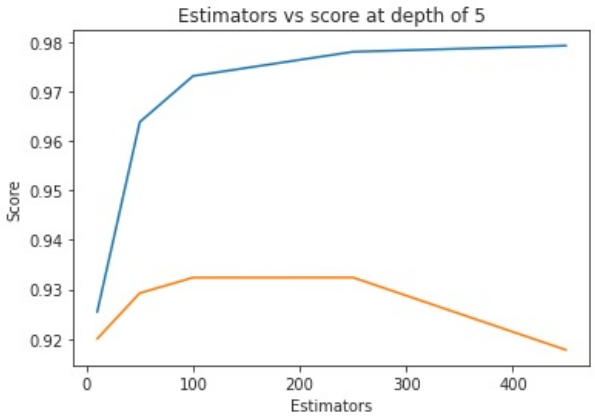
In [129]:

```
estimators = [10,50,100,250,450]
train_scores = []
test_scores = []
for i in estimators:
    clf = xgb.XGBClassifier(objective='binary:logistic', n_estimators=i)
    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('Estimators = ',i,'Train Score',train_sc,'test Score',test_sc)
plt.plot(estimators,train_scores,label='Train Score')
plt.plot(estimators,test_scores,label='Test Score')
plt.xlabel('Estimators')
plt.ylabel('Score')
plt.title('Estimators vs score at depth of 5')
```

```
Estimators =  10 Train Score 0.9254131685939068 test Score 0.9200093531449951
Estimators =  50 Train Score 0.9638144497841492 test Score 0.9292465739005958
Estimators =  100 Train Score 0.9731212731708799 test Score 0.9323831411161948
Estimators =  250 Train Score 0.9779977775532882 test Score 0.9324010308842791
Estimators =  450 Train Score 0.9792449020795478 test Score 0.9177554080102805
```

Out[129]:

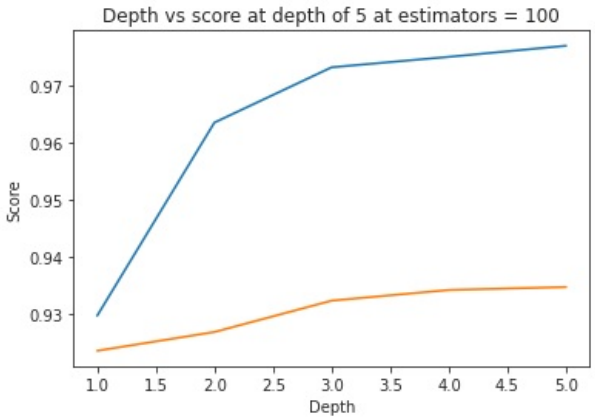
```
Text(0.5, 1.0, 'Estimators vs score at depth of 5')
```



In [130]:

```
depths = [1,2,3,4,5]
train_scores = []
test_scores = []
for i in depths:
    clf = xgb.XGBClassifier(objective='binary:logistic', n_estimators=100, max_depth=i)
    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 = 100')
plt.show()
```

```
depth =  1 Train Score 0.9297505925786085 test Score 0.9236288433536625
depth =  2 Train Score 0.9634714789125732 test Score 0.9269093204292026
depth =  3 Train Score 0.9731212731708799 test Score 0.9323831411161948
depth =  4 Train Score 0.9749443432503542 test Score 0.9342465753424658
depth =  5 Train Score 0.9768815355676462 test Score 0.9347601693665607
```



In [131]:

```
param_dist = {"n_estimators":sp_randint(105,125),
              "max_depth": sp_randint(10,15)
              }

clf = xgb.XGBClassifier(objective='binary:logistic', random_state=25,n_jobs=-1)

xgb_random = RandomizedSearchCV(clf, param_distributions=param_dist,
                                n_iter=5,cv=10,scoring='f1',random_state=25, return_train_score=True)

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

```
mean test scores [0.97991932 0.97993146 0.97953032 0.97985492 0.97988669]
mean train scores [0.99990342 0.99993339 0.99232804 0.99537968 0.99571373]
```

In [133]:

```
xgb_random.best_estimator_
```

Out[133]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=14,
              min_child_weight=1, missing=None, n_estimators=123, n_jobs=-1,
              nthread=None, objective='binary:logistic', random_state=25,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

In [134]:

```
clf = xgb.XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=14,
              min_child_weight=1, missing=None, n_estimators=123, n_jobs=-1,
              nthread=None, objective='binary:logistic', random_state=25,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

In [135]:

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

In [136]:

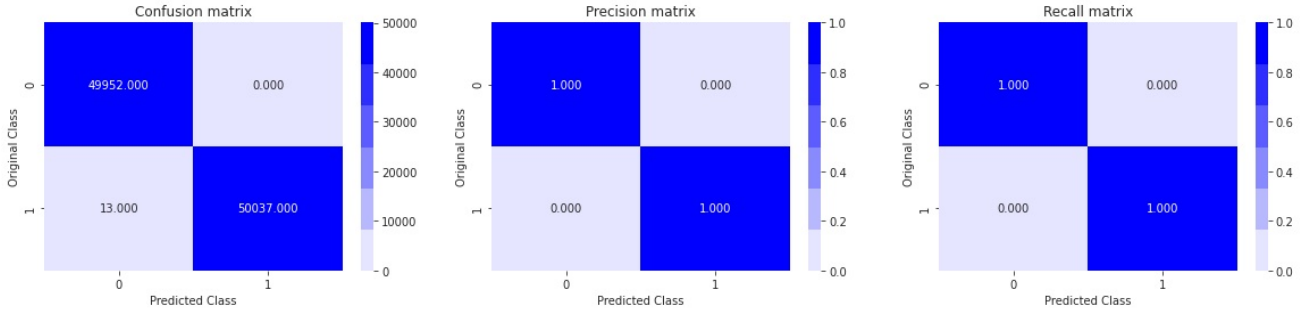
```
from sklearn.metrics import f1_score
print('Train f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))
```

```
Train f1 score 0.9998701130016886
Test f1 score 0.9331026771587096
```

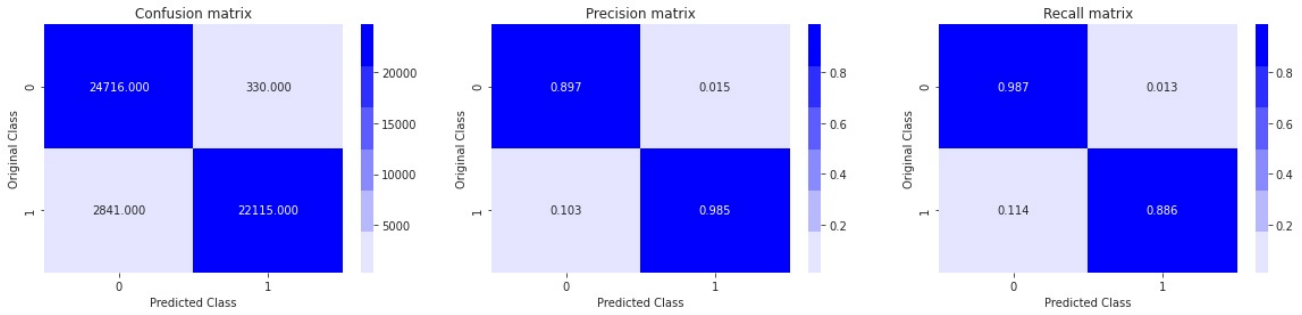
In [137]:

```
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
```

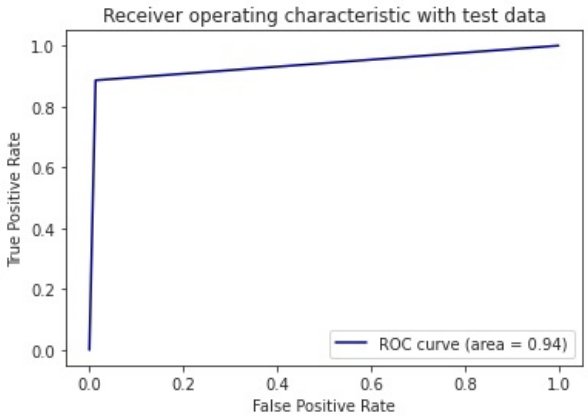



```
Test confusion_matrix
```



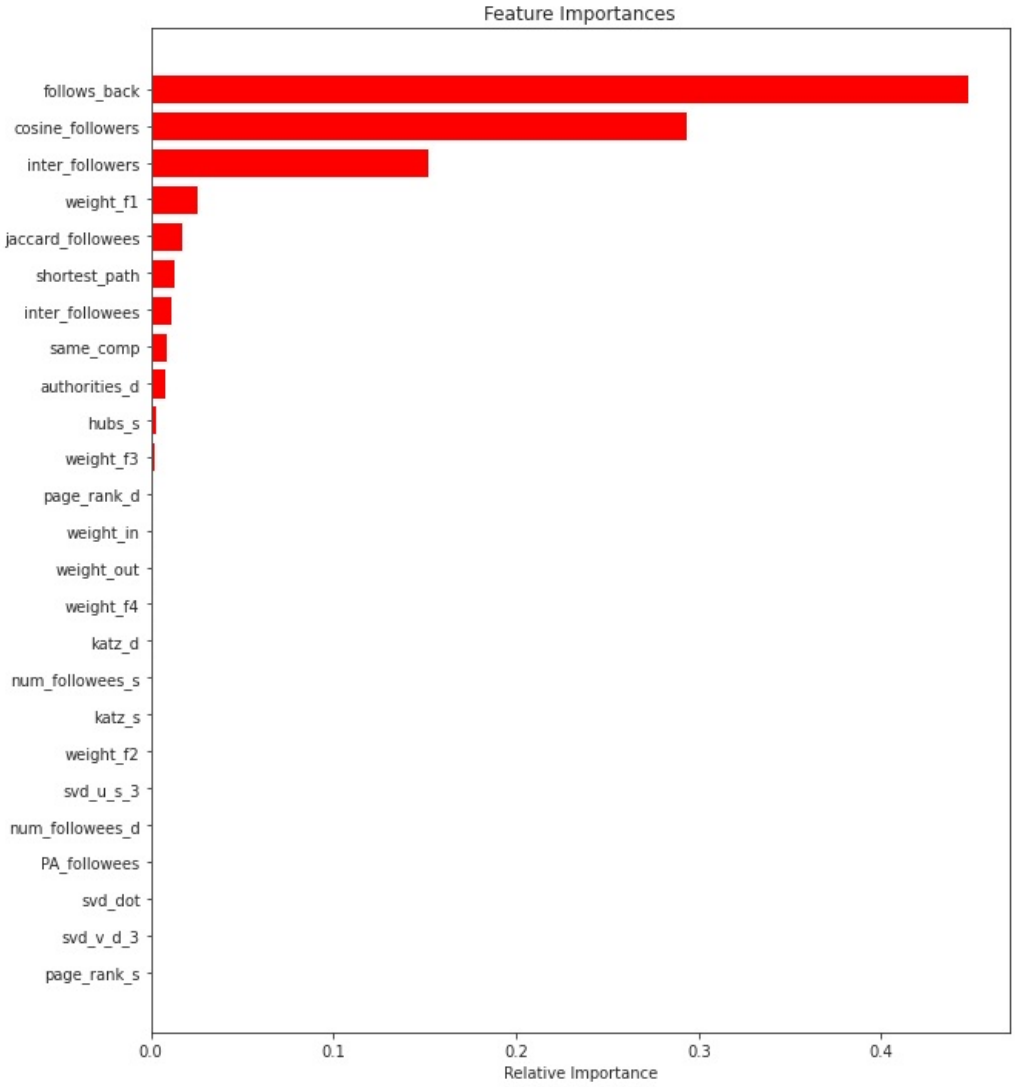
In [138]:

```
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 [139]:

```
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 [142]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model", "n_estimators", "max_depth", "Train f1-Score","Test f1-Score"]
x.add_row(['Random Forest','121','14','0.9632','0.9303'])
x.add_row(['XGBOOST','123','14','0.9998','0.9331'])
print(x)
```

```
+---------------+--------------+-----------+----------------+---------------+
|     Model     | n_estimators | max_depth | Train f1-Score | Test f1-Score |
+---------------+--------------+-----------+----------------+---------------+
| Random Forest |     121      |     14    |     0.9632     |     0.9303    |
|    XGBOOST    |     123      |     14    |     0.9998     |     0.9331    |
+---------------+--------------+-----------+----------------+---------------+
```

In [ ]: