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://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
```

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:
    g=nx.read_edgelist('data/after_eda/train_woheader.csv',delimiter=',',create_using=n
x.DiGraph(),nodetype=int)
    print(nx.info(g))
```

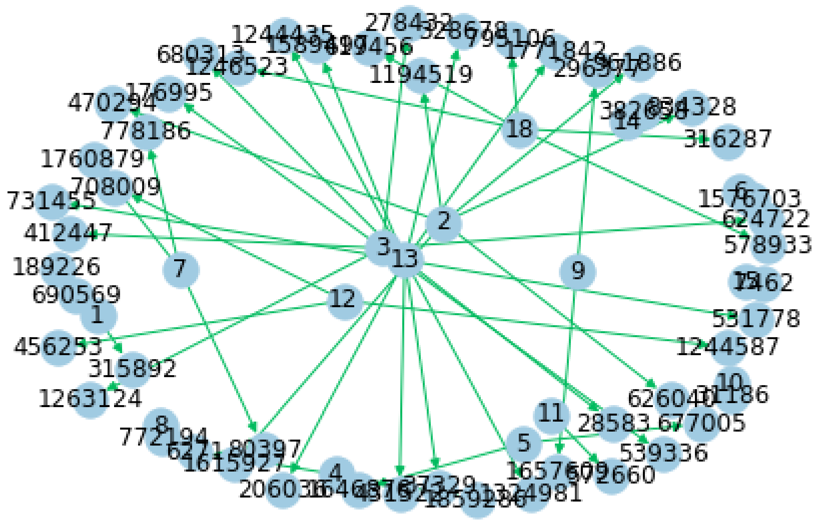
```
Name:
Type: DiGraph
Number of nodes: 1862220
Number of edges: 9437519
Average in degree:   5.0679
Average out degree:   5.0679
```

Displaying a sub graph

In [3]:

```
warnings.filterwarnings("ignore", category=UserWarning)

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

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]:

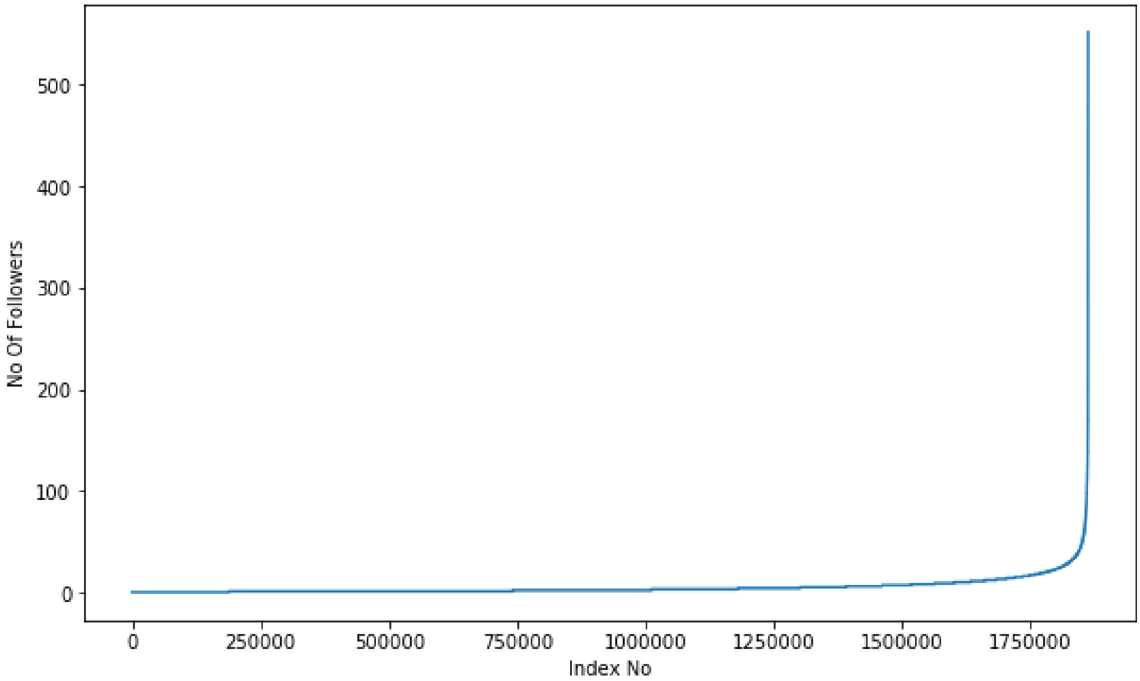
```
# No of Unique persons
print("The number of unique persons",len(g.nodes()))
```

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

1.1 No of followers for each person

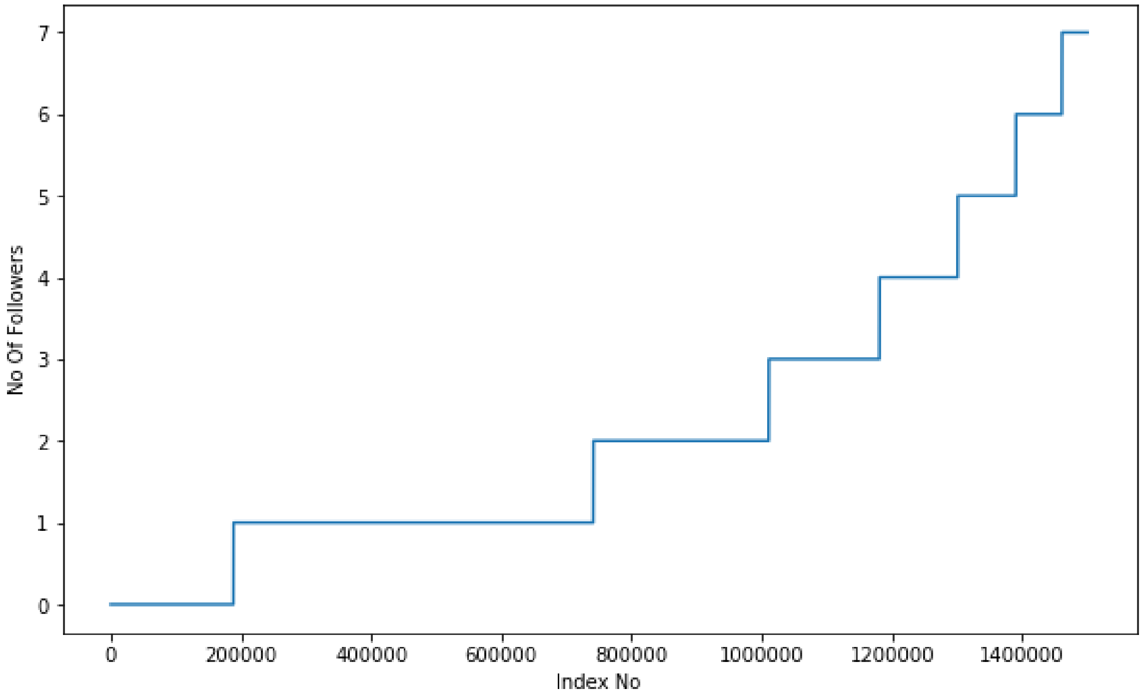
In [5]:

```
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()
```



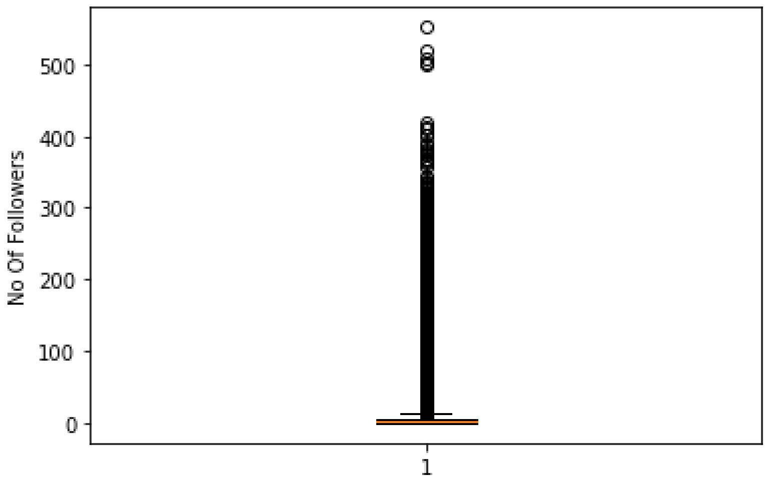
In [6]:

```
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()
```



In [7]:

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



In [8]:

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

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

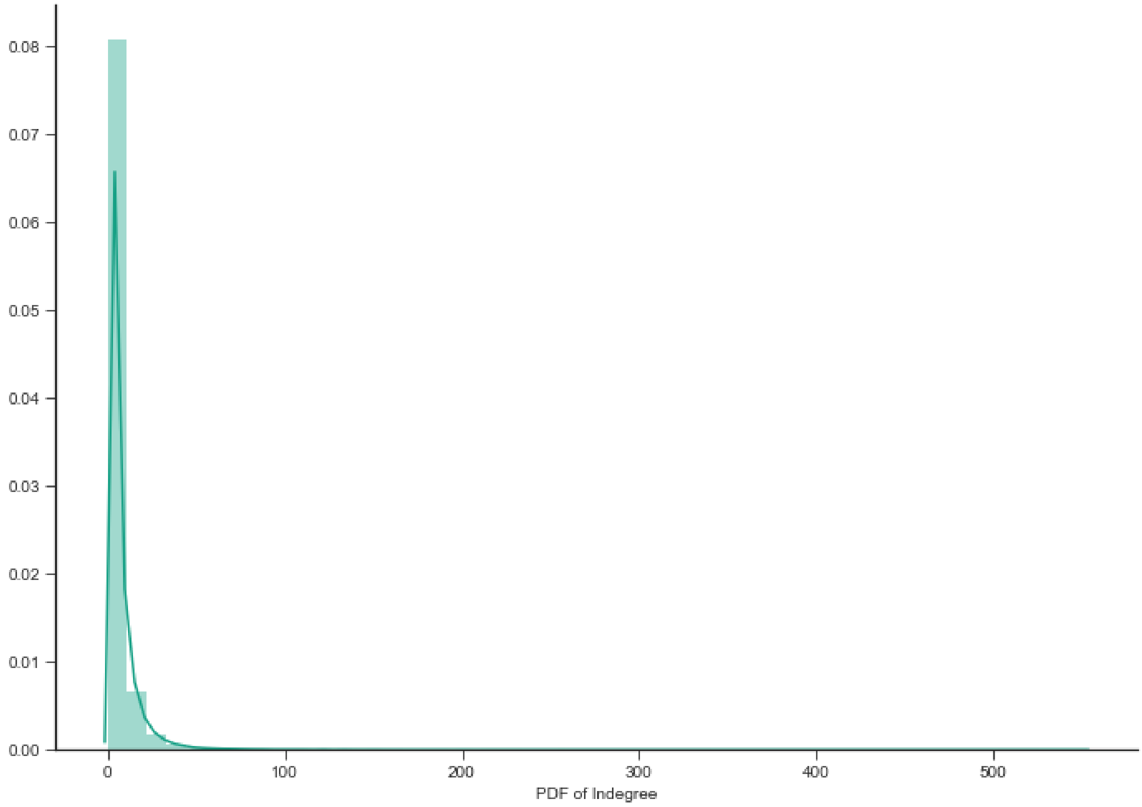
In [9]:

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

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

In [10]:

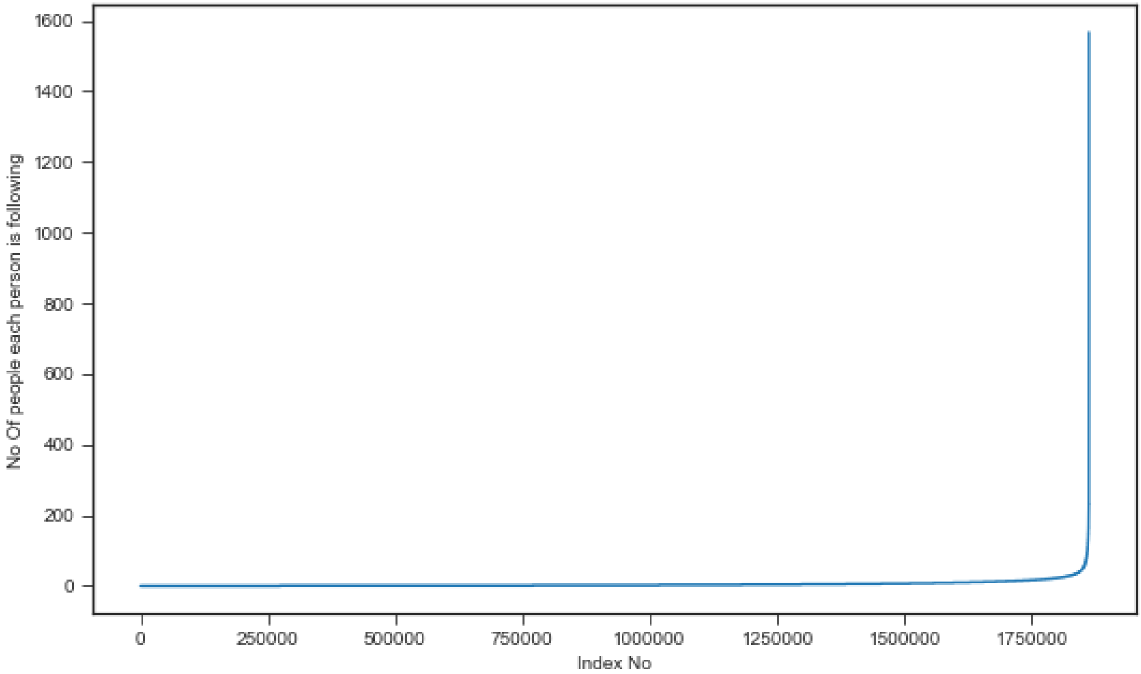
```
%matplotlib inline
sns.set_style('ticks')
fig, ax = plt.subplots()
fig.set_size_inches(11.7, 8.27)
sns.distplot(indegree_dist, color='#16A085')
plt.xlabel('PDF of Indegree')
sns.despine()
#plt.show()
```



1.2 No of followees each person has

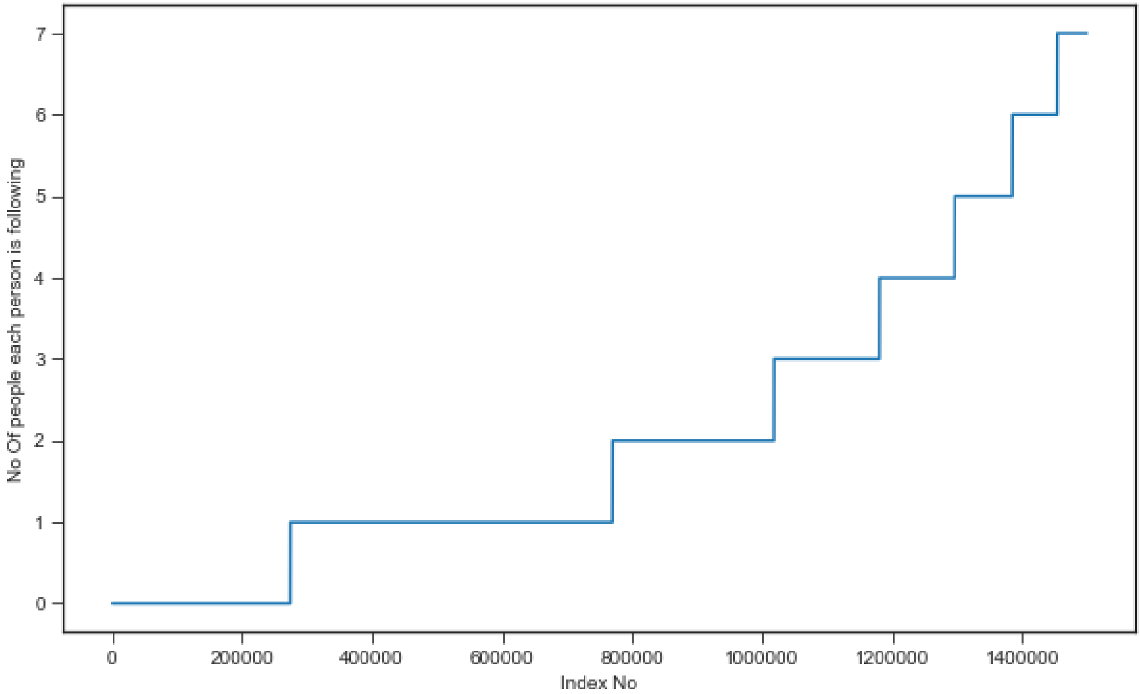
In [11]:

```
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()
```



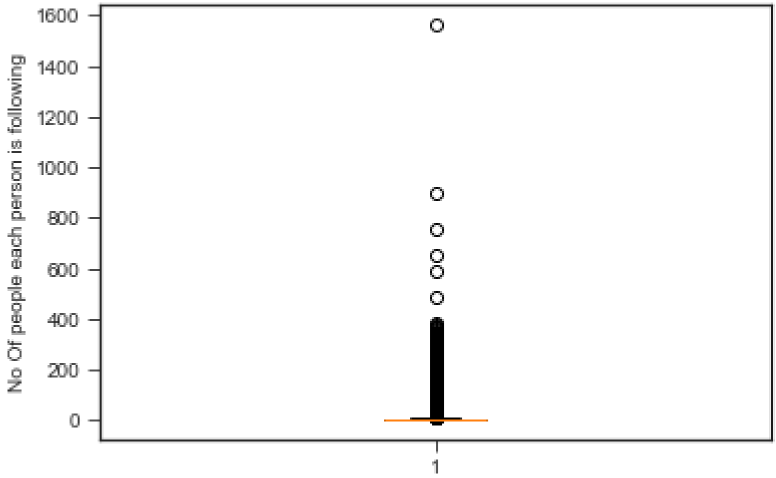
In [12]:

```
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 [13]:

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



In [14]:

```
### 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
```

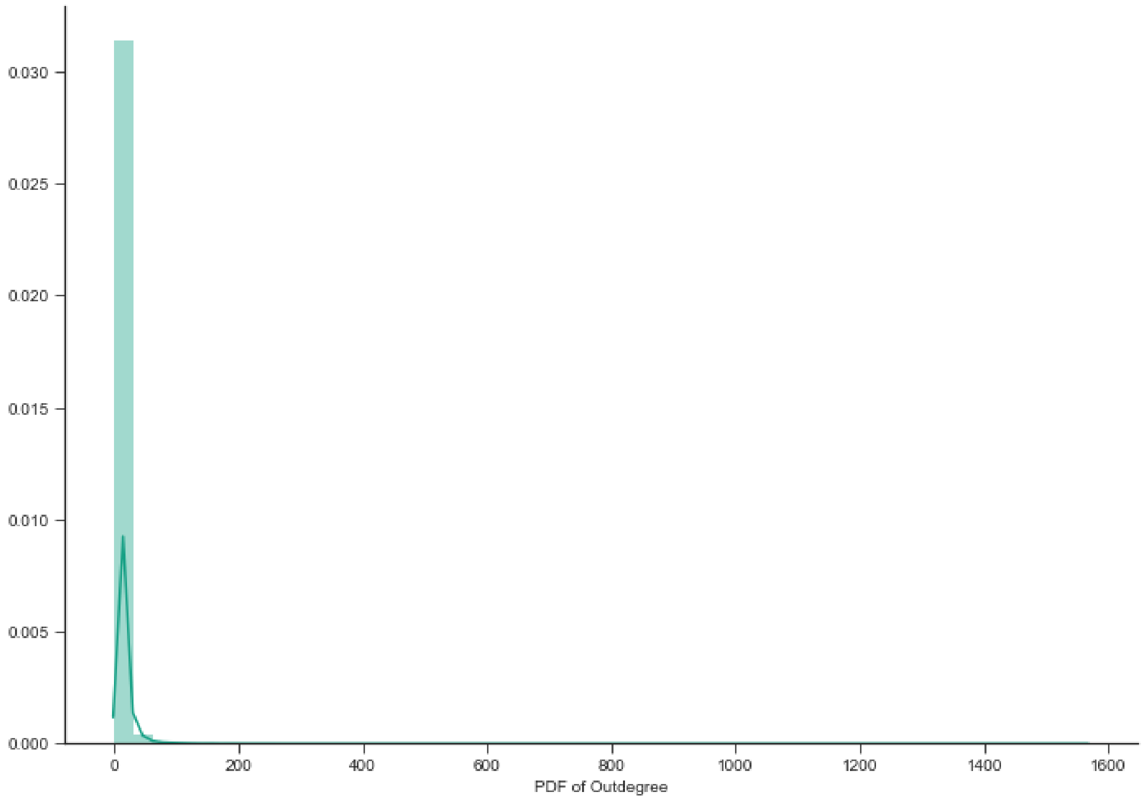
In [15]:

```
### 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 [16]:

```
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 [17]:

```
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 [18]:

```
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 [19]:

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

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

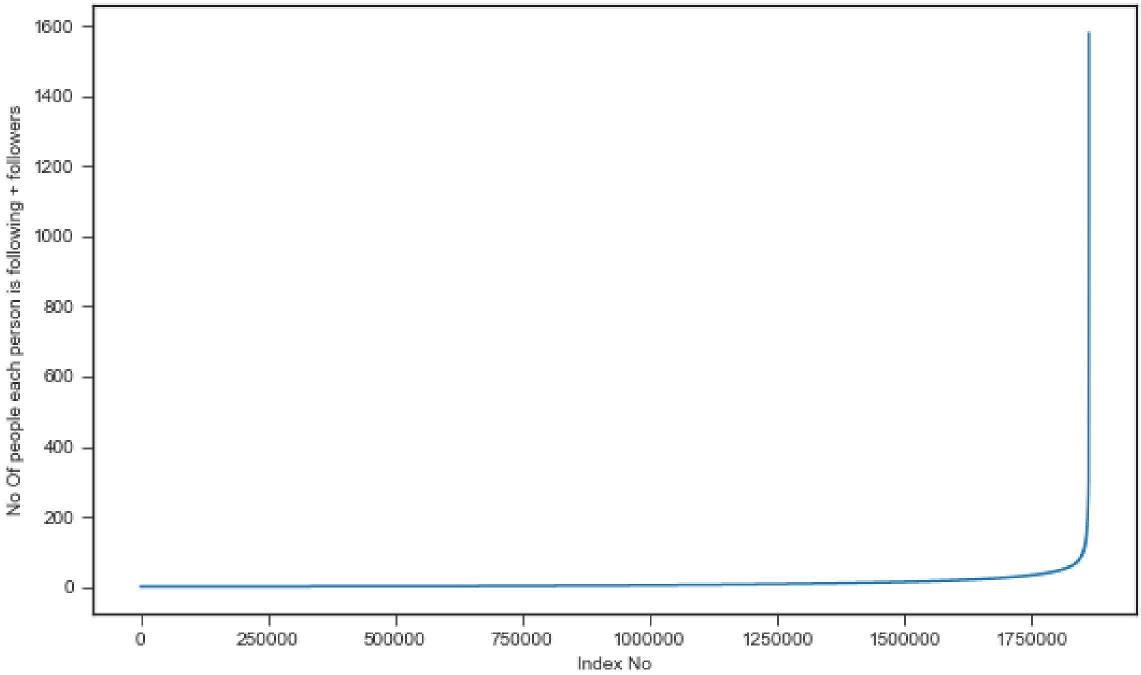
1.3 both followers + following

In [20]:

```
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()))
```

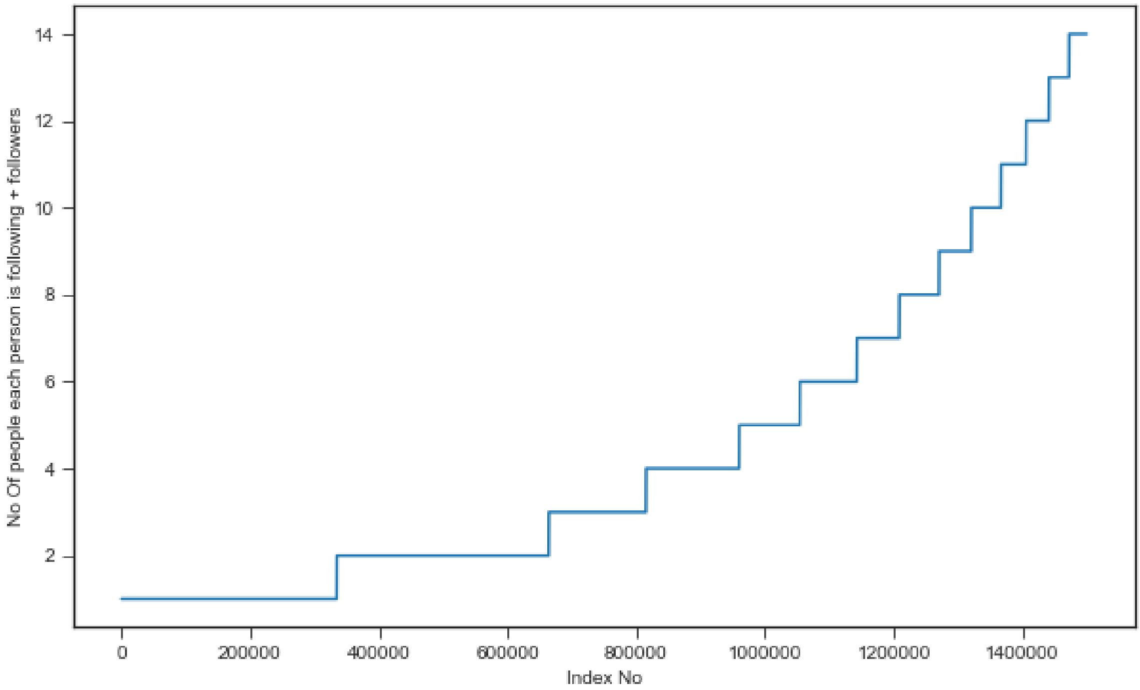
In [21]:

```
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 [22]:

```
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 [23]:

```
### 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 [24]:

```
### 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 [25]:

```
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 [26]:

```
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 [27]:

```
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 [28]:

```
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 [29]:

```
%%time
###generating bad edges from given graph
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


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

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

```
Wall time: 6.41 s
```

In [30]:

```
len(missing_edges)
```

Out[30]:

```
9437519
```

2.2 Training and Test data split:

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

In [31]:

```
from sklearn.model_selection import train_test_split

df_pos = pd.read_csv('data/train.csv')
df_neg = pd.DataFrame(list(missing_edges), columns=['source_node', 'destination_node'])

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

```
Number of nodes in the graph with edges 9437519
Number of nodes in the graph without edges 9437519
```

In [32]:

```
#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
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)
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=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])
```

```
============================================================
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 [33]:

```
del missing_edges
```

In [35]:

```
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 have a cold start problem here

In [36]:

```
#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('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 = pd.read_csv('data/after_eda/train_neg_after_eda.csv', names=['source_
node', 'destination_node'])
    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])

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

    X_train.to_csv('data/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)
```

In [38]:

```
X_train_pos = pd.read_csv('data/after_eda/train_pos_after_eda.csv', names=['source_nod
e', 'destination_node'])
X_test_pos = pd.read_csv('data/after_eda/test_pos_after_eda.csv', names=['source_node',
'destination_node'])
X_train_neg = pd.read_csv('data/after_eda/train_neg_after_eda.csv', names=['source_nod
e', 'destination_node'])
X_test_neg = pd.read_csv('data/after_eda/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('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=False)
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,)
```

3. Similarity measures

3.1 Jaccard Distance:

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

\begin{equation} j = \frac{|X\cap Y|}{|X \cup Y|} \end{equation}

In [40]:

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

In [41]:

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

```
0.0
```

In [44]:

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

```
0
```

3.2 Cosine distance

\begin{equation} CosineDistance = \frac{|X\cap Y|}{|X|\cdot|Y|} \end{equation}

In [42]:

```
#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.successo
rs(b)))))/\
                                    (math.sqrt(len(set(train_graph.successors(a)))*len
((set(train_graph.successors(b))))))
        return sim
    except:
        return 0
```

In [43]:

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

```
0.0
```

In [45]:

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

```
0
```

In [46]:

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

        if len(set(train_graph.predecessors(a))) == 0  | len(set(train_graph.predecesso
rs(b))) == 0:
            return 0
        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 [47]:

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

```
0.02886751345948129
```

In [48]:

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

```
0
```

4. 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.**

4.1 Page Ranking

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

In [50]:

```
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 [51]:

```
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 [52]:

```
#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
```

5. Other Graph Features

5.1 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 [53]:

```
#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 [54]:

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

Out[54]:

10

In [55]:

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

Out[55]:

-1

5.2 Checking for same community

In [56]:

```
#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 [57]:

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

Out[57]:

0

In [58]:

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

Out[58]:

0

5.3 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 [59]:

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

In [60]:

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

Out[60]:

0

In [61]:

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

Out[61]:

0

5.4 Is persion was following back:

In [62]:

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

In [63]:

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

Out[63]:

```
1
```

In [64]:

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

Out[64]:

```
0
```

5.5 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 [65]:

```
if not os.path.isfile('data/fea_sample/katz.p'):
    katz = nx.katz.katz_centrality(train_graph,alpha=0.005,beta=1)
    pickle.dump(katz,open('data/fea_sample/katz.p','wb'))
else:
    katz = pickle.load(open('data/fea_sample/katz.p','rb'))
```

In [66]:

```
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 [67]:

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

```
0.0007483800935562018
```

5.6 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)

In [68]:

```
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 [69]:

```
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
```

5.7 Preferential Attachment

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

In [79]:

```
# for followers
def pref_followers(a,b):
    try:
        if len(set(train_graph.predecessors(a))) == 0  | len(set(train_graph.predecesso
rs(b))) == 0:
            return 0
        flwr = (len(set(train_graph.predecessors(a)))*len(set(train_graph.predecessors(
b))))
        return flwr
    except:
        return 0
```

In [82]:

```
pref_followers(1,189226)
```

Out[82]:

```
9
```

In [83]:

```
pref_followers(669354,1635354)
```

Out[83]:

0

In [80]:

```
# for followees
def pref_followees(a,b):
    try:
        if len(set(train_graph.successors(a))) == 0  | len(set(train_graph.successors(b
))) == 0:
            return 0
        flwe = (len(set(train_graph.successors(a)))*len(set(train_graph.successors(b
))))
        return flwe
    except:
        return 0
```

In [189]:

```
pref_followees(1,189226)
```

Out[189]:

8

In [85]:

```
pref_followees(669354,1635354)
```

Out[85]:

0

6. Featurization

6. 1 Reading a sample of Data from both train and test

In [81]:

```
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 [86]:

```
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 [87]:

```
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 [88]:

```
df_final_train = pd.read_csv('data/after_eda/train_after_eda.csv', skiprows=skip_train,
names=['source_node', 'destination_node'])
df_final_train['indicator_link'] = pd.read_csv('data/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[88]:

	source_node	destination_node	indicator_link
0	273084	1505602	1
1	1492633	1370536	1

In [89]:

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

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

Out[89]:

	source_node	destination_node	indicator_link
0	848424	784690	1
1	908573	1205667	1

6.2 Adding a set of features

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

- jaccard_followers
- jaccard_followees
- cosine_followers
- cosine_followees
- num_followers_s
- num_followees_s
- num_followers_d
- num_followees_d
- inter_followers
- inter_followees

In [90]:

```
if not os.path.isfile('data/fea_sample/storage_sample_stage1.h5'):
    #mapping jaccrd followers to train and test data
    df_final_train['jaccard_followers'] = df_final_train.apply(lambda row:
                                            jaccard_for_followers(row['source_node'],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 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
['destination_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
['destination_node']),axis=1)
```

In [91]:

```
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_fo
llowers, inter_followees
```

In [94]:

```
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 = pd.read_hdf('data/fea_sample/storage_sample_stage1.h5', 'train_df'
,mode='r')
    df_final_test = pd.read_hdf('data/fea_sample/storage_sample_stage1.h5', 'test_df',m
ode='r')
```

6.3 Adding new set of features

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

- adar index
- is following back
- belongs to same weakly connect components
- shortest path between source and destination

In [95]:

```
if not os.path.isfile('data/fea_sample/storage_sample_stage2.h5'):
    #mapping adar index on train
    df_final_train['adar_index'] = df_final_train.apply(lambda row: calc_adar_in(row['s
ource_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['sou
rce_node'],row['destination_node']),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['destination_node']),axis=1)

    #mapping followback or not on test
    df_final_test['follows_back'] = df_final_test.apply(lambda row: follows_back(row['s
ource_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['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(ro
w['source_node'],row['destination_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_p
ath_length(row['source_node'],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 = pd.read_hdf('data/fea_sample/storage_sample_stage2.h5', 'train_df'
,mode='r')
    df_final_test = pd.read_hdf('data/fea_sample/storage_sample_stage2.h5', 'test_df',m
ode='r')
```

6.4 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

\begin{equation} W = \frac{1}{\sqrt{1+|X|}} \end{equation}

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

In [97]:

```
from tqdm import tqdm

#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 [00:41<00:00, 42936.36it/s]
```

In [98]:

```
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 [100]:

```
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 = pd.read_hdf('data/fea_sample/storage_sample_stage3.h5', 'train_df'
,mode='r')
    df_final_test = pd.read_hdf('data/fea_sample/storage_sample_stage3.h5', 'test_df',m
ode='r')
```

In [184]:

```
if not os.path.isfile('data/fea_sample/storage_sample_stage5.h5'):

    #mapping preferential followers to train and test data
    df_final_train['preferential_followers'] = df_final_train.apply(lambda row:
                                            pref_followers(row['source_node'],row['dest
ination_node']),axis=1)
    df_final_test['preferential_followers'] = df_final_test.apply(lambda row:
                                            pref_followers(row['source_node'],row['dest
ination_node']),axis=1)

    #mapping preferential followees to train and test data
    df_final_train['preferential_followees'] = df_final_train.apply(lambda row:
                                            pref_followees(row['source_node'],row['dest
ination_node']),axis=1)
    df_final_test['preferential_followees'] = df_final_test.apply(lambda row:
                                            pref_followees(row['source_node'],row['dest
ination_node']),axis=1)
    #=================================================================================
=====================================
    hdf = HDFStore('data/fea_sample/storage_sample_stage5.h5')
    hdf.put('train_df',df_final_train, format='table', data_columns=True)
    hdf.put('test_df',df_final_test, format='table', data_columns=True)
    hdf.close()
else:
    df_final_train = pd.read_hdf('data/fea_sample/storage_sample_stage5.h5', 'train_df'
,mode='r')
    df_final_test = pd.read_hdf('data/fea_sample/storage_sample_stage5.h5', 'test_df',m
ode='r')
```

In [193]:

```
df_final_train['preferential_followers'].head(2)
```

Out[193]:

```
0      66
1    1598
Name: preferential_followers, dtype: int64
```

In [195]:

```
if not os.path.isfile('data/fea_sample/storage_sample_stage5.h5'):


    df_final_train['preferential_followers'] = df_final_train.apply(lambda row:
                                            pref_followers(row['source_node'],row['dest
ination_node']),axis=1)
    df_final_test['preferential_followers'] = df_final_test.apply(lambda row:
                                            pref_followers(row['source_node'],row['dest
ination_node']),axis=1)

    #mapping preferential followees to train and test data
    df_final_train['preferential_followees'] = df_final_train.apply(lambda row:
                                            pref_followees(row['source_node'],row['dest
ination_node']),axis=1)
    df_final_test['preferential_followees'] = df_final_test.apply(lambda row:
                                            pref_followees(row['source_node'],row['dest
ination_node']),axis=1)

    hdf = HDFStore('data/fea_sample/storage_sample_stage5.h5')
    hdf.put('train_df',df_final_train, format='table', data_columns=True)
    hdf.put('test_df',df_final_test, format='table', data_columns=True)
    hdf.close()
else:
    df_final_train = pd.read_hdf('data/fea_sample/storage_sample_stage5.h5', 'train_df'
,mode='r')
    df_final_test = pd.read_hdf('data/fea_sample/storage_sample_stage5.h5', 'test_df',m
ode='r')
    #==============================================================================
============
```

6.5 Adding new set of features

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

SVD features for both source and destination

In [242]:

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

In [143]:

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

In [147]:

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

In [148]:

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

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

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

In [150]:

```
U
```

Out[150]:

```
array([[ 2.06284693e-15,  4.86280655e-12,  4.32863754e-13,
         6.34824893e-14, -1.18650990e-14,  1.85697417e-16],
       [ 5.39811427e-14,  7.68601834e-13,  1.32207398e-11,
         2.12945787e-13, -9.80313828e-13,  5.21482908e-16],
       [ 1.03323958e-11,  2.77939878e-11,  1.56793759e-09,
         2.35993372e-11, -3.87897324e-12,  6.99955332e-13],
       ...,
       [ 9.08915625e-15,  3.01082247e-14,  6.61712782e-13,
         1.86989050e-13, -2.39751698e-12,  1.01908637e-14],
       [ 4.04925854e-13,  1.66308678e-13,  1.36923604e-06,
         6.08333141e-14, -1.84662987e-13,  1.79395452e-12],
       [ 7.33016328e-15,  7.79262509e-15,  7.97715764e-14,
         2.89266504e-16, -1.21291971e-15,  2.25575583e-16]])
```

In [151]:

```
V
```

Out[151]:

```
array([[ 1.35264368e-15,  3.04926826e-13,  4.79660436e-12, ...,
         1.06138100e-13,  8.44181291e-13,  5.09694582e-16],
       [ 2.35858819e-10,  1.73676805e-11,  3.76124109e-12, ...,
         7.70091217e-13,  2.64078077e-14,  1.78740929e-15],
       [ 2.65961108e-12,  7.05455009e-11,  8.78160283e-10, ...,
         3.23522370e-12,  3.27472305e-07,  9.62865188e-15],
       [ 5.49863099e-14,  1.55584571e-13,  1.02312685e-10, ...,
         8.70916801e-14,  8.65388459e-14,  4.65720679e-15],
       [-2.64504798e-14, -4.03427281e-13, -2.52257595e-11, ...,
        -2.01629167e-12, -7.70333282e-14, -2.67517523e-15],
       [ 1.34507398e-17,  3.91916631e-16,  1.67583723e-13, ...,
         2.58442270e-14,  5.69272820e-14,  2.30977840e-17]])
```

In [152]:

```
s
```

Out[152]:

```
array([ 98.71963254, 100.36027594, 103.01815334, 104.62922418,
       125.15765247, 140.75447598])
```

In [149]:

```
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()
```

In [170]:

```
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components=20, n_iter=7, random_state=42)
svd_mat = svd.fit_transform(Adj)
```

In [171]:

```
def svd_dot(df):
    svd_dot = []
    for idx,temp in tqdm(df.iterrows(),total=df.shape[0]):
        source_index=sadj_dict.get(temp.destination_node,None)
        dest_index=sadj_dict.get(temp['source_node'])
        if ( source_index is not None and dest_index is not None):
            svd_temp = np.dot(svd_mat[source_index,:],svd_mat[dest_index,:])
            svd_dot.append(svd_temp)
        else:
            svd_dot.append(0)
    return svd_dot
```

In [172]:

```
df_final_train['svd_dot']=svd_dot(df_final_train)
df_final_test['svd_dot']=svd_dot(df_final_test)
```

```
100%|█████████████████████████████████████████████████████████████████████
███| 100002/100002 [00:47<00:00, 2125.58it/s]
100%|█████████████████████████████████████████████████████████████████████
█████| 50002/50002 [00:13<00:00, 3732.52it/s]
```

In [173]:

```
print(df_final_train['svd_dot'].head(2))
```

```
0    1.183292e-07
1    3.495259e+01
Name: svd_dot, dtype: float64
```

In [174]:

```
print(df_final_test['svd_dot'].head(2))
```

```
0    9.146824e-16
1    1.435302e-13
Name: svd_dot, dtype: float64
```

In [192]:

```
df_final_train.columns
```

Out[192]:

```
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_dot', 'preferential_followers',
       'preferential_followees'],
      dtype='object')
```

In [196]:

```
df_final_test.columns
```

Out[196]:

```
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_dot'],
      dtype='object')
```

In [197]:

```
df_final_train.head()
```

Out[197]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followee
0	273084	1505602	1	0	0.000000
1	832016	1543415	1	0	0.187135
2	1325247	760242	1	0	0.369565
3	1368400	1006992	1	0	0.000000
4	140165	1708748	1	0	0.000000

5 rows × 31 columns

In [198]:

```
df_final_test.head()
```

Out[198]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followee
0	848424	784690	1	0	0.0
1	483294	1255532	1	0	0.0
2	626190	1729265	1	0	0.0
3	947219	425228	1	0	0.0
4	991374	975044	1	0	0.2

5 rows × 31 columns

In [199]:

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

In [200]:

```
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)
```

7.1 Random Forest

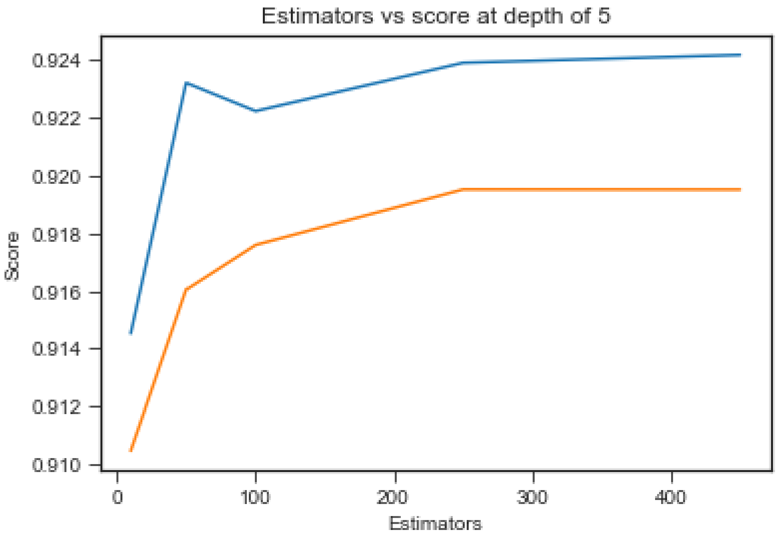
In [202]:

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

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,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))
    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 0.9145289240526332 test Score 0.9104742581790
516
Estimators =  50 Train Score 0.9231899419648011 test Score 0.9160505302294
14
Estimators =  100 Train Score 0.9222163920663239 test Score 0.917593997639
5211
Estimators =  250 Train Score 0.9238891979552636 test Score 0.919518417778
9024
Estimators =  450 Train Score 0.9241616653376422 test Score 0.919513583004
9106
```

Out[202]:

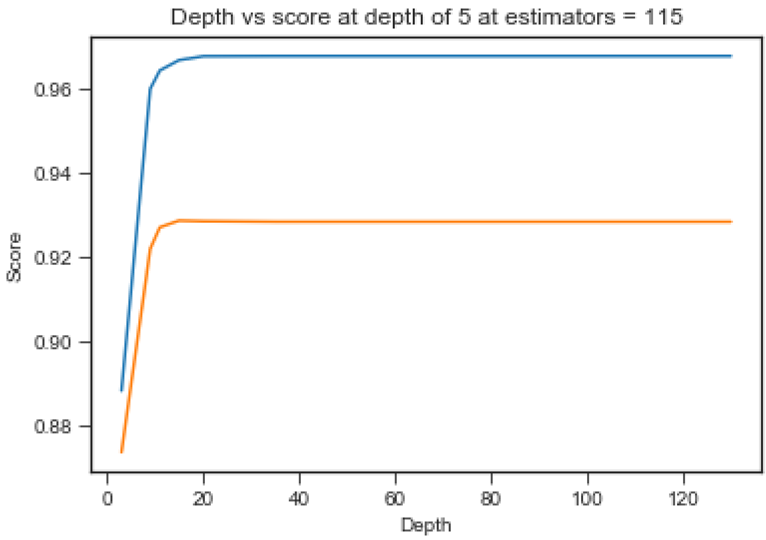
```
Text(0.5, 1.0, 'Estimators vs score at depth of 5')
```



In [203]:

```
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 0.888147616433729 test Score 0.8735248845561826
depth =  9 Train Score 0.9598355567766687 test Score 0.9217842893031308
depth =  11 Train Score 0.9641923521186698 test Score 0.9268999705919421
depth =  15 Train Score 0.9666514204401078 test Score 0.9284900645487913
depth =  20 Train Score 0.9675860598739177 test Score 0.9283514072323727
depth =  35 Train Score 0.96762666612525 test Score 0.9282248196126418
depth =  50 Train Score 0.96762666612525 test Score 0.9282248196126418
depth =  70 Train Score 0.96762666612525 test Score 0.9282248196126418
depth =  130 Train Score 0.96762666612525 test Score 0.9282248196126418
```



In [215]:

```
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
from xgboost.sklearn import XGBClassifier

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)

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 [0.96455638 0.96353094 0.96222878 0.96429569 0.96621159]

---------------------------------------------------------------------------
-
KeyError                                  Traceback (most recent call las
t)
<ipython-input-215-8cad5f91ce2c> in <module>()
     19 rf_random.fit(df_final_train,y_train)
     20 print('mean test scores',rf_random.cv_results_['mean_test_score'])
---> 21 print('mean train scores',rf_random.cv_results_['mean_train_score'
])

KeyError: 'mean_train_score'
```

In [ ]:

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

In [217]:

```
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 [218]:

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

In [219]:

```
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 0.9679461812425801
Test f1 score 0.9282791207863984
```

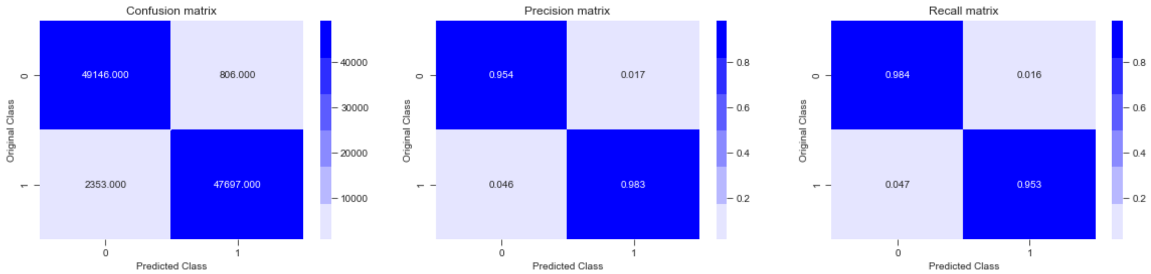
In [220]:

```
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()
```

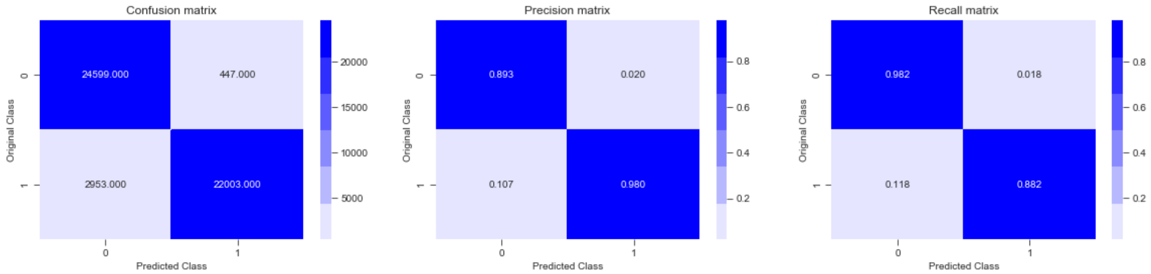
In [221]:

```
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

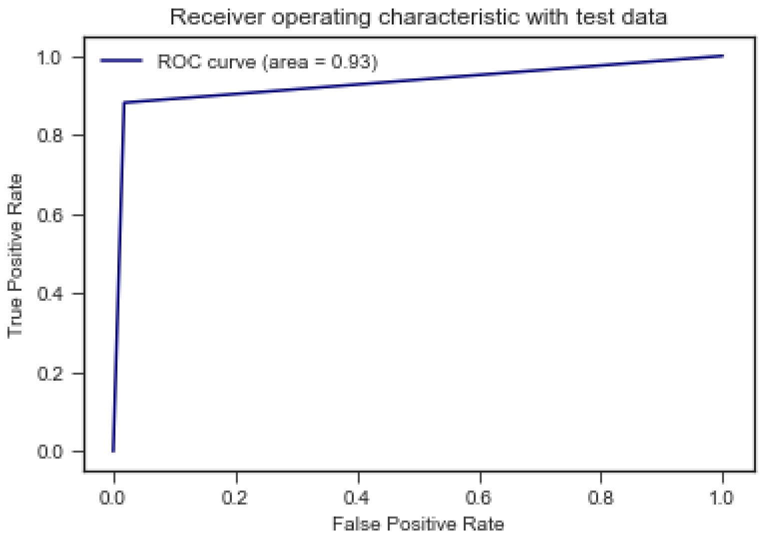


Test confusion_matrix



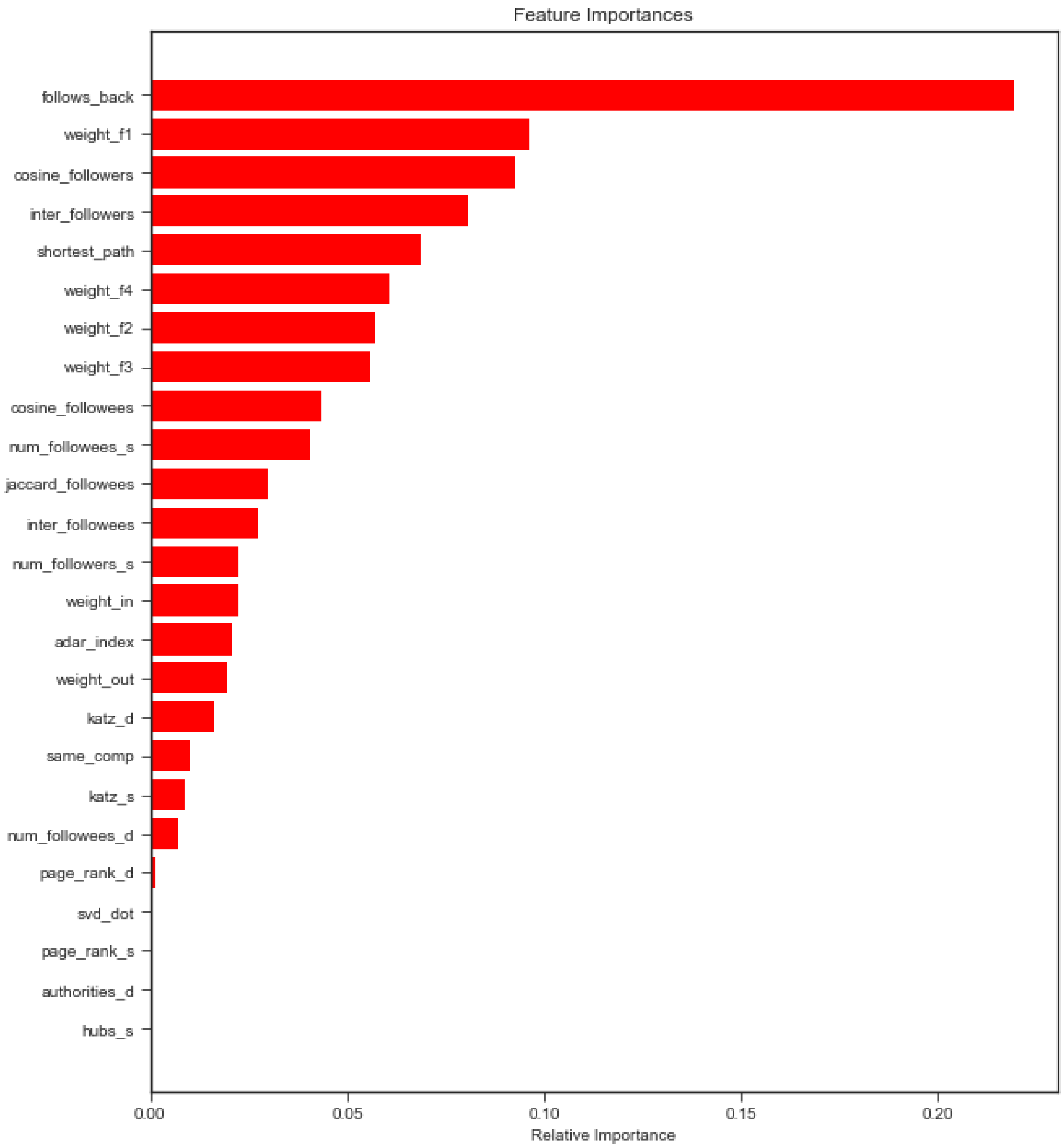
In [222]:

```
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 [223]:

```
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()
```



7.2 XGBoost

In [224]:

```
from xgboost.sklearn import XGBClassifier

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

clf = XGBClassifier(random_state=25,n_jobs=-1)

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

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

```
mean test scores [0.97769192 0.9779754  0.97842738 0.97823428 0.97769295]

--------------------------------------------------------------------------
-
KeyError                                  Traceback (most recent call las
t)
<ipython-input-224-f55d930703fa> in <module>()
     13 rf_random.fit(df_final_train,y_train)
     14 print('mean test scores',rf_random.cv_results_['mean_test_score'])
---> 15 print('mean train scores',rf_random.cv_results_['mean_train_score'
])

KeyError: 'mean_train_score'
```

In [225]:

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

```
XGBClassifier(base_score=0.5, booster=None, colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints=None,
              learning_rate=0.300000012, max_delta_step=0, max_depth=11,
              min_child_weight=1, min_samples_leaf=56, min_samples_split=1
79,
              missing=nan, monotone_constraints=None, n_estimators=106,
              n_jobs=-1, num_parallel_tree=1, objective='binary:logistic',
              random_state=25, reg_alpha=0, reg_lambda=1, scale_pos_weight
=1,
              subsample=1, tree_method=None, validate_parameters=False,
              verbosity=None)
```

In [234]:

```
clf = XGBClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=11, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=56, min_samples_split=179,
            min_weight_fraction_leaf=0.0, n_estimators=106, n_jobs=-1,
            oob_score=False, random_state=25, verbose=0, warm_start=False)
```

In [235]:

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

In [236]:

```
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 0.9999900098902087
Test f1 score 0.9247106702276485
```

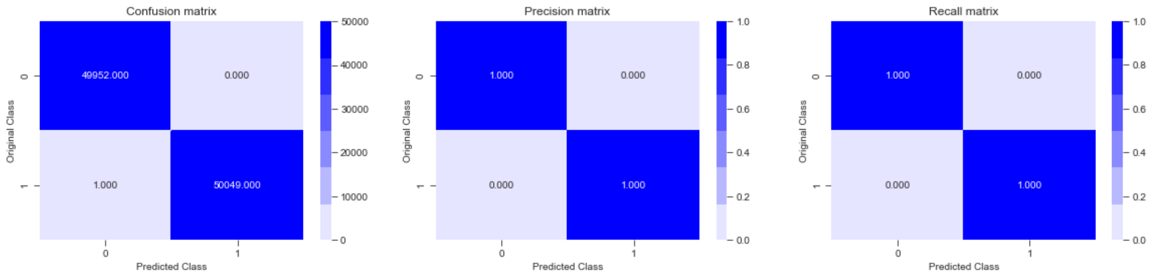
In [237]:

```
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()
```

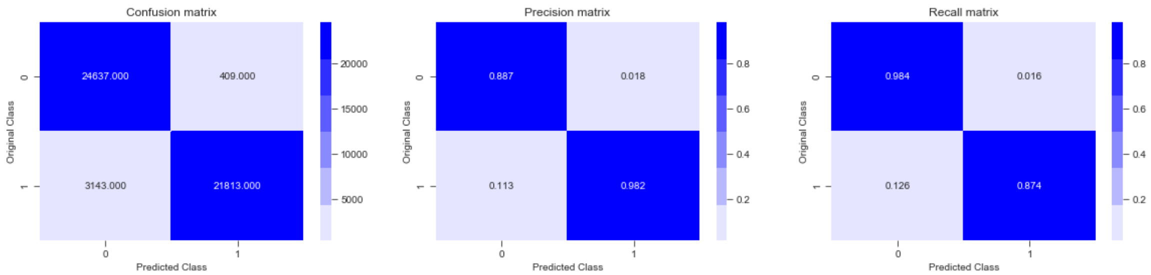
In [238]:

```
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

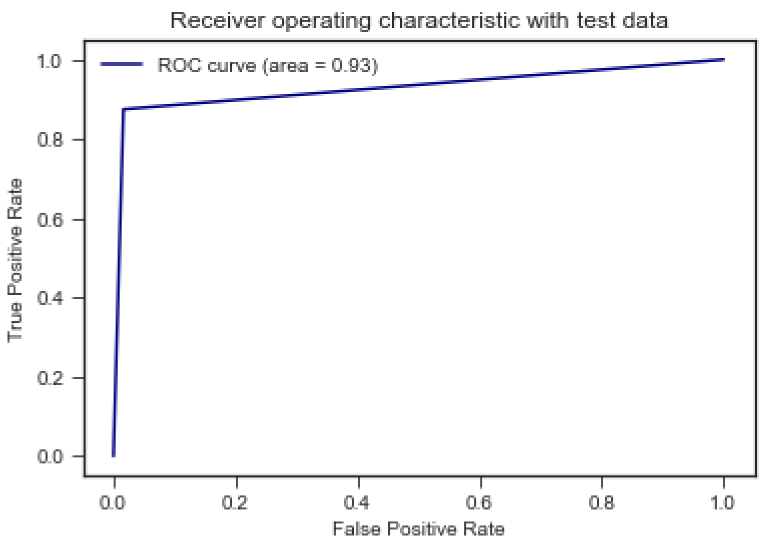


Test confusion_matrix



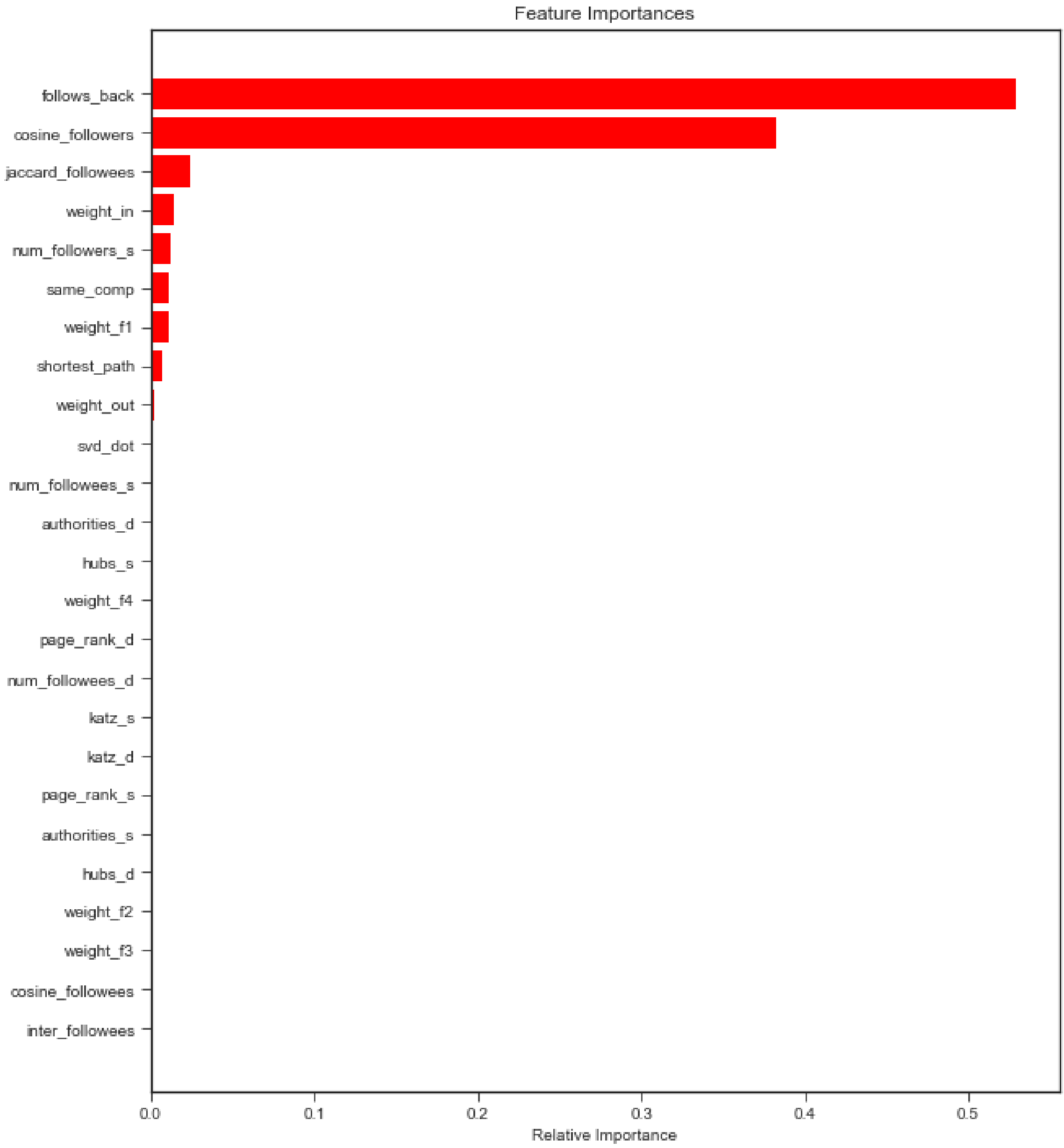
In [239]:

```
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 [240]:

```
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 [241]:

```
from prettytable import PrettyTable
ptable = PrettyTable()
ptable.field_names = ["Model", "max_depth", "Train f1-Score","Test f1-Score"]
ptable.add_row(['Random Forest','14','96.7','92.82'])
ptable.add_row(['XGBOOST','11','99.99','92.47'])
print(ptable)
```

```
+---------------+-----------+----------------+---------------+
|     Model     | max_depth | Train f1-Score | Test f1-Score |
+---------------+-----------+----------------+---------------+
| Random Forest |     14    |      96.7      |     92.82     |
|    XGBOOST    |     11    |     99.99      |     92.47     |
+---------------+-----------+----------------+---------------+
```

Observations

The given dataset is an imbalanced data. Number of nodes is 1.8M and number of edges are 9.3M.

While obtaining EDA observed that people who are followed by others and are following is only 12 for 90%.

After doing train-test split, performed the data featurization.

Featurization is the important task in this case study, Extracted various features like

- jaccord distance,
- cosine distance,
- weight features,
- adar index,
- preferential attachment,
- svd dot.

Trained using Random Forest, Xgboost machine learning models.