Social network Graph Link Prediction - Facebook Challenge

Problem statement:

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

Data Overview

Taken data from facebook's recruting challenge on kaggle https://www.kaggle.com/c/FacebookRecruiting
data contains two columns source and destination eac edge in graph

```
- Data columns (total 2 columns):
- source_node          int64
- destination_node     int64
```

Mapping the problem into supervised learning problem:

- Generated training samples of good and bad links from given directed graph and for each link got some features like no of followers, is he followed back, page rank, katz score, adar index, some svd fetures of adj matrix, some weight features etc. and trained ml model based on these features to predict link.
- Some reference papers and videos :
  - https://www.cs.cornell.edu/home/kleinber/link-pred.pdf
  - https://www3.nd.edu/~dial/publications/lichtenwalter2010new.pdf
  - https://kaggle2.blob.core.windows.net/forum-message-attachments/2594/supervised_link_prediction.pdf
  - https://www.youtube.com/watch?v=2M77Hgy17cg

Business objectives and constraints:

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

Performance metric for supervised learning:

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

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 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=nx.DiGraph(),nod
etype=int)
    print(nx.info(g))
```

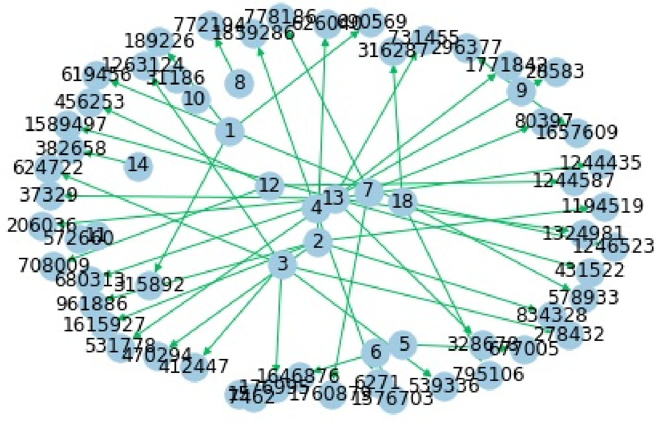
```
Name:
Type: DiGraph
Number of nodes: 1862220
Number of edges: 9437519
Average in degree:   5.0679
Average out degree:   5.0679
```

> Displaying a sub graph

In [3]:

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

subgraph=nx.read_edgelist('train_woheader_sample.csv',delimiter=',',create_using=nx.DiGraph(),node
type=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
```



1. Exploratory Data Analysis

In [4]:

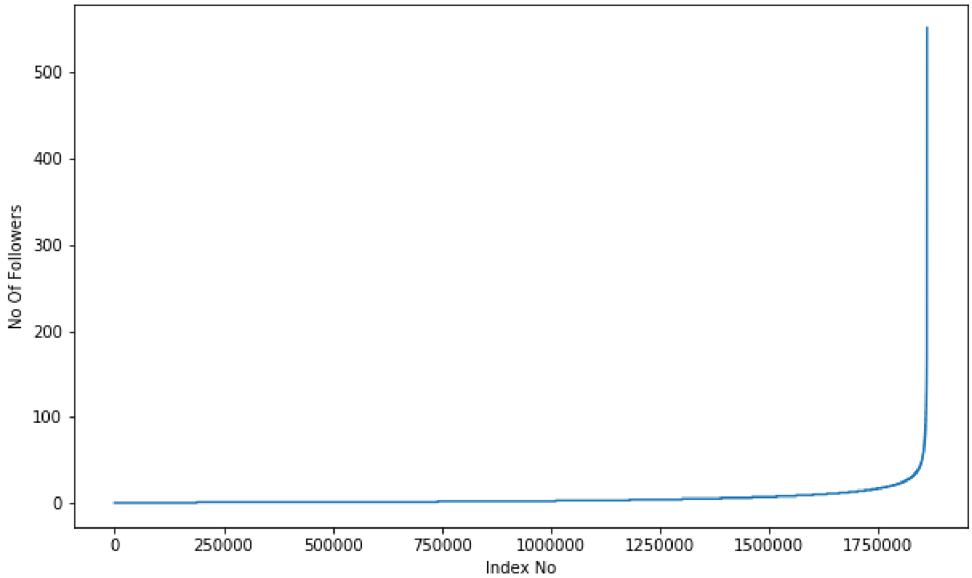
```
# No of Unique persons
print("The number of unique persons",len(g.nodes()))
```

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

1.1 No of followers for each person

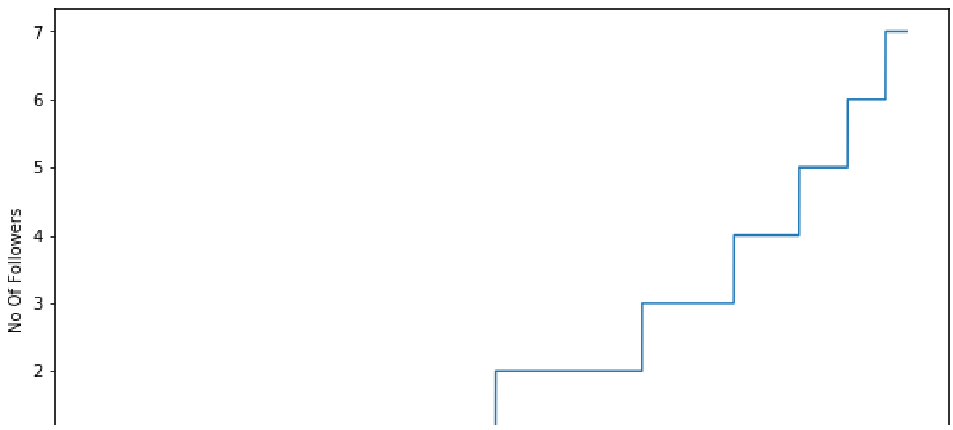
In [5]:

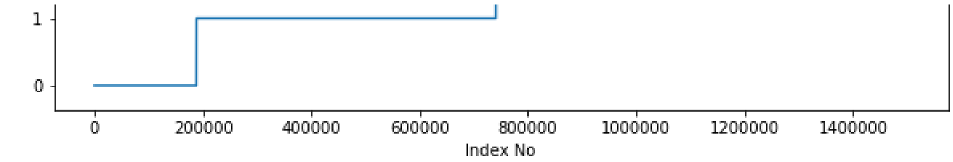
```
indegree_dist = list(dict(g.in_degree()).values())
indegree_dist.sort()
plt.figure(figsize=(10,6))
plt.plot(indegree_dist)
plt.xlabel('Index No')
plt.ylabel('No Of Followers')
plt.show()
```



In [6]:

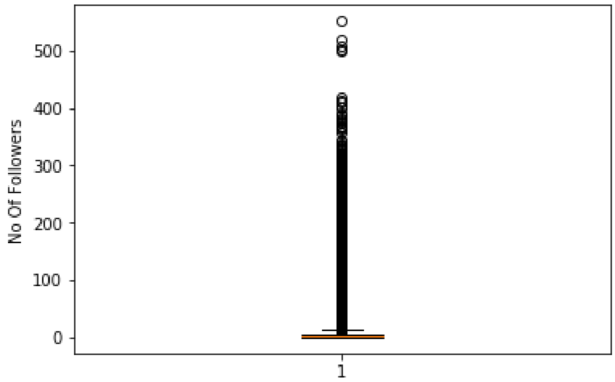
```
indegree_dist = list(dict(g.in_degree()).values())
indegree_dist.sort()
plt.figure(figsize=(10,6))
plt.plot(indegree_dist[0:1500000])
plt.xlabel('Index No')
plt.ylabel('No Of Followers')
plt.show()
```





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]:

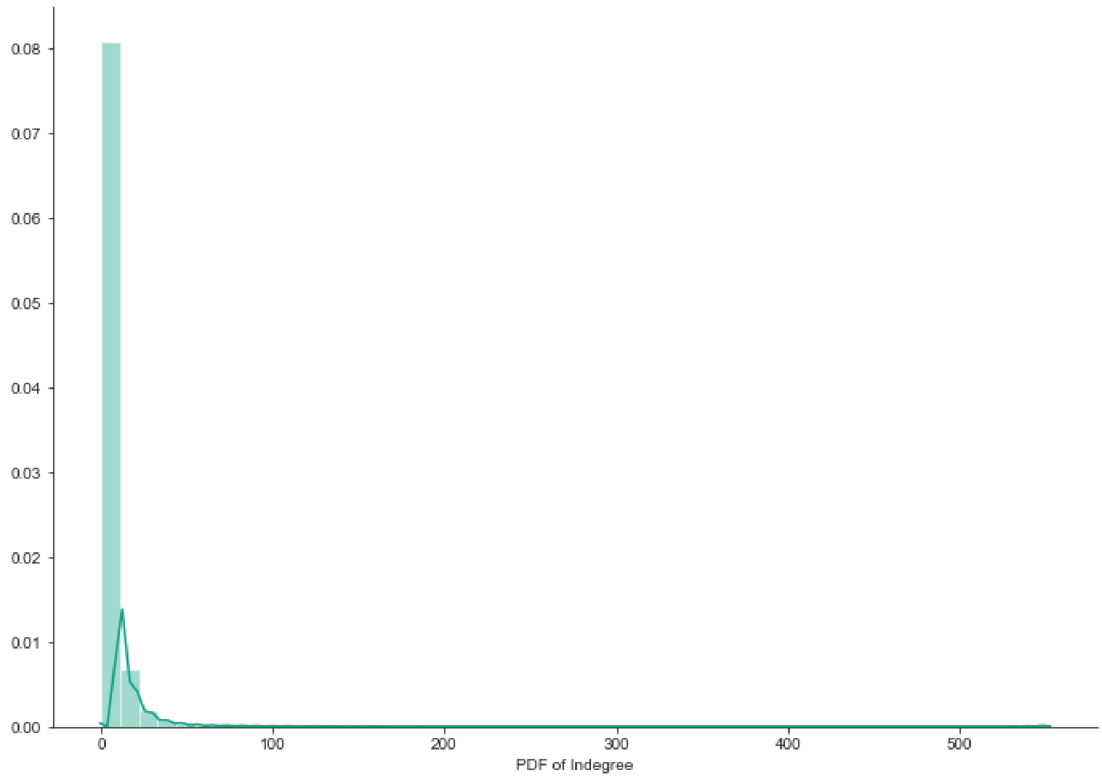
```
### 99-100 percentile
for i in range(10,110,10):
    print(99+(i/100),'percentile value is',np.percentile(indegree_dist,99+(i/100)))
```

```
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()
```

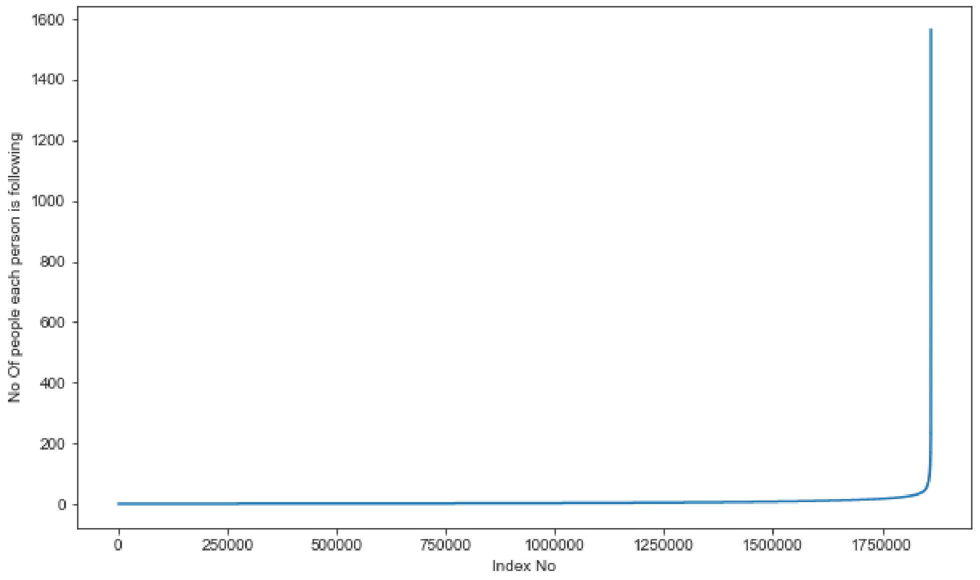
```
#plt.show()
```



1.2 No of people each person is following

In [11]:

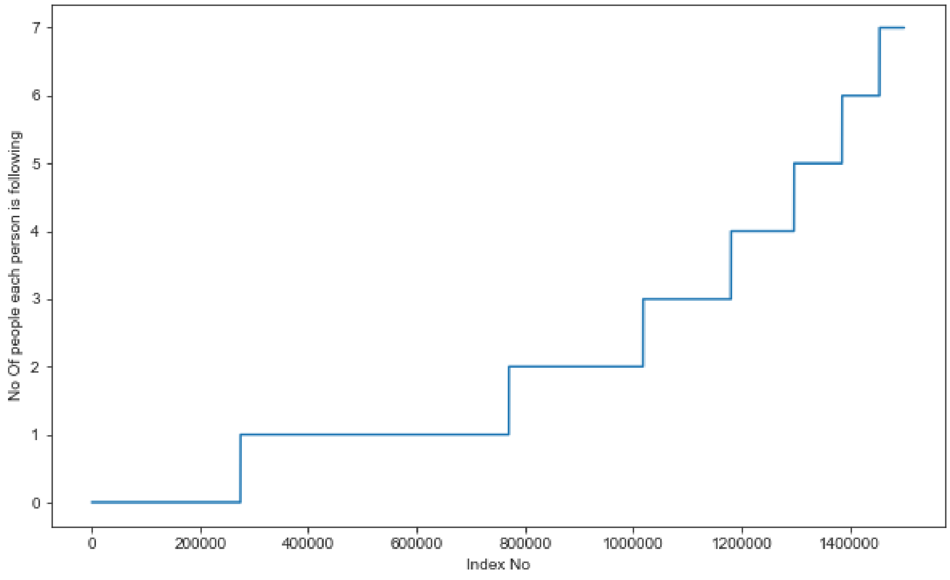
```
outdegree_dist = list(dict(g.out_degree()).values())
outdegree_dist.sort()
plt.figure(figsize=(10,6))
plt.plot(outdegree_dist)
plt.xlabel('Index No')
plt.ylabel('No Of people each person is following')
plt.show()
```



In [12]:

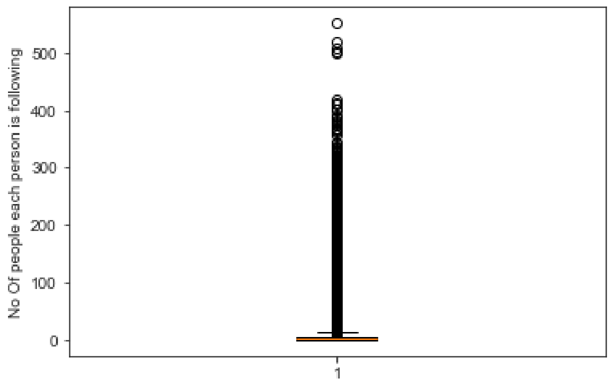
```
indegree_dist = list(dict(g.in_degree()).values())
indegree_dist.sort()
plt.figure(figsize=(10,6))
plt.plot(outdegree_dist[0:1500000])
plt.xlabel('Index No')
plt.ylabel('No Of people each person is following')
```

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



In [13]:

```
plt.boxplot(indegree_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]:

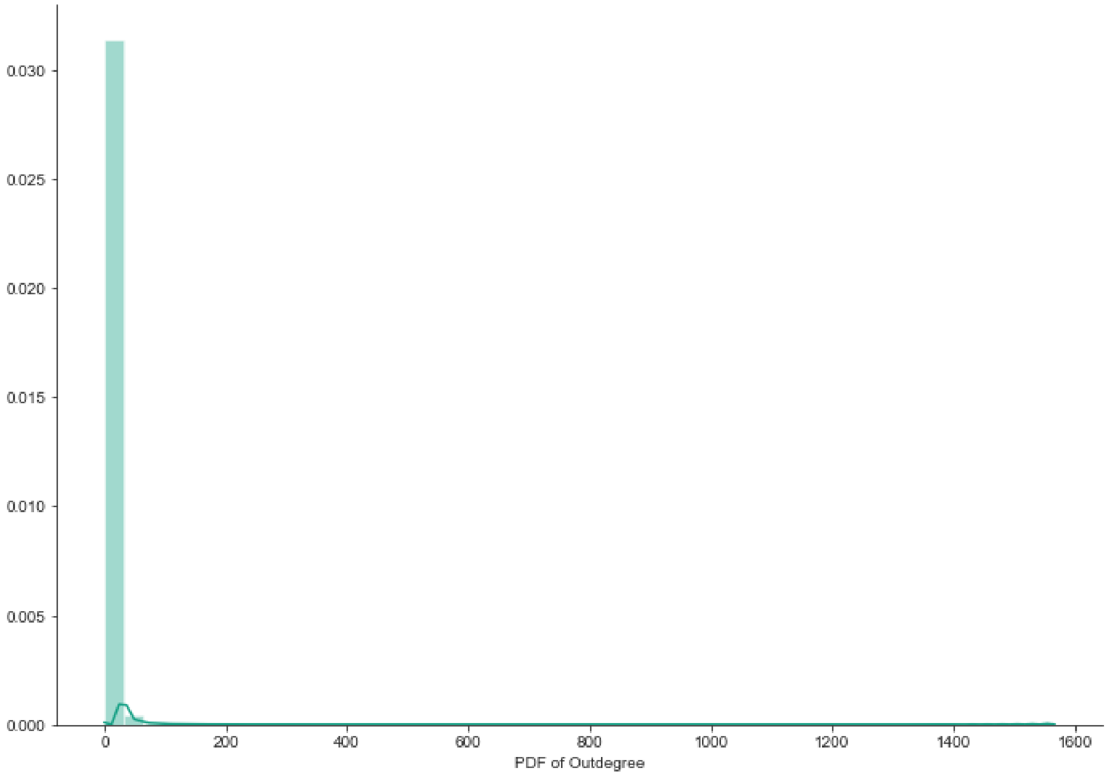
```
### 99-100 percentile
for i in range(10,110,10):
    print(99+(i/100),'percentile value is',np.percentile(outdegree_dist,99+(i/100)))
```

```
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 14.741115442858524
```

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 [20]:

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

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

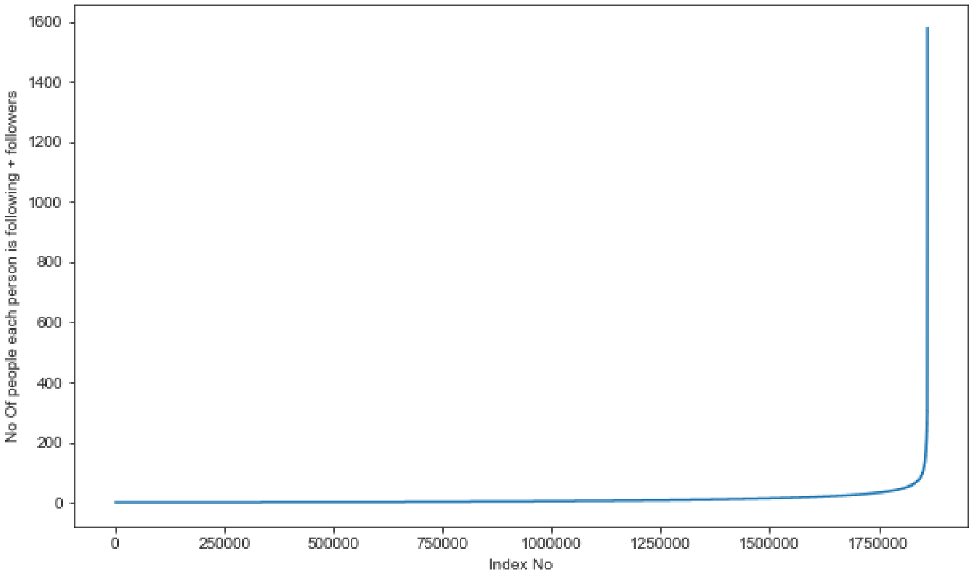
1.3 both followers + following

In [22]:

```python
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()))
```

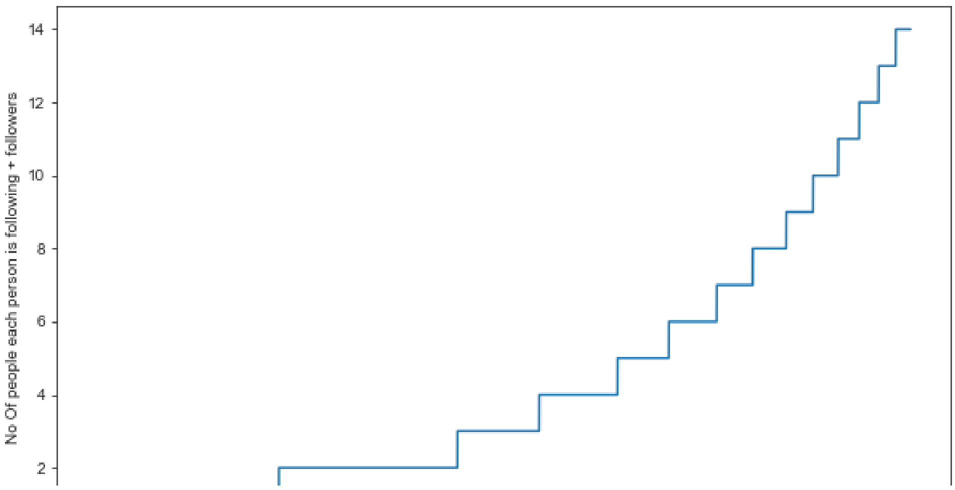
In [23]:

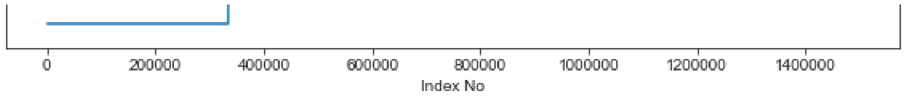
```python
in_out_degree_sort = sorted(in_out_degree)
plt.figure(figsize=(10,6))
plt.plot(in_out_degree_sort)
plt.xlabel('Index No')
plt.ylabel('No Of people each person is following + followers')
plt.show()
```



In [24]:

```python
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 [25]:

```
### 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 [26]:

```
### 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 [27]:

```
len(in_out_degree==in_out_degree.min())
```

Out[27]:

```
1862220
```

In [28]:

```
print('Min of no of followers + following is',in_out_degree.min())
print(np.sum(in_out_degree==in_out_degree.min()),' persons having minimum no of followers +
following')
```

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

In [29]:

```
print('Max of no of followers + following is',in_out_degree.max())
print(np.sum(in_out_degree==in_out_degree.max()),' persons having maximum no of followers +
following')
```

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

In [30]:

```
(in_out_degree[:10]<10)
```

Out[30]:

```
array([ True, False, False,  True, False, False, False, False, False,
       False])
```

In [31]:

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

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

In [32]:

```
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 [33]:

```
r = csv.reader(open('data/after_eda/train_woheader.csv','r'))
edges = dict()
for edge in r:
    edges[(edge[0], edge[1])] = 1
edges
```

Out[33]:

```
{('1', '690569'): 1,
 ('1', '315892'): 1,
 ('1', '189226'): 1,
 ('2', '834328'): 1,
 ('2', '1615927'): 1,
 ('2', '1194519'): 1,
 ('2', '470294'): 1,
 ('2', '961886'): 1,
 ('2', '626040'): 1,
 ('3', '176995'): 1,
 ('3', '624722'): 1,
 ('3', '412447'): 1,
 ('3', '1263124'): 1,
 ('3', '278432'): 1,
 ('3', '539336'): 1,
 ('4', '6271'): 1,
 ('4', '1859286'): 1,
 ('5', '1646876'): 1,
 ('5', '677005'): 1,
 ('6', '1576703'): 1,
 ('7', '1760879'): 1,
 ('7', '80397'): 1,
 ('7', '778186'): 1,
 ('8', '772194'): 1,
 ('9', '296377'): 1,
 ('9', '1657609'): 1,
 ('10', '31186'): 1,
 ('11', '572660'): 1,
 ('12', '1244587'): 1,
```

```
('12', '456253'): 1,
('12', '708009'): 1,
('13', '206036'): 1,
('13', '1244435'): 1,
('13', '28583'): 1,
('13', '1771842'): 1,
('13', '1589497'): 1,
('13', '731455'): 1,
('13', '328678'): 1,
('13', '1324981'): 1,
('13', '431522'): 1,
('13', '531778'): 1,
('13', '680313'): 1,
('13', '37329'): 1,
('14', '382658'): 1,
('15', '7462'): 1,
('18', '619456'): 1,
('18', '316287'): 1,
('18', '795106'): 1,
('18', '578933'): 1,
('18', '1246523'): 1,
('18', '1323975'): 1,
('18', '984426'): 1,
('18', '1003537'): 1,
('18', '1468933'): 1,
('18', '1688534'): 1,
('18', '1269182'): 1,
('18', '1753748'): 1,
('18', '248374'): 1,
('18', '1377966'): 1,
('18', '78146'): 1,
('19', '254810'): 1,
('19', '860701'): 1,
('19', '1205150'): 1,
('19', '220385'): 1,
('19', '401594'): 1,
('20', '493028'): 1,
('21', '212060'): 1,
('22', '57396'): 1,
('24', '322832'): 1,
('24', '423929'): 1,
('24', '1435'): 1,
('24', '1310872'): 1,
('25', '721394'): 1,
('25', '899566'): 1,
('25', '604764'): 1,
('25', '1465410'): 1,
('25', '1068517'): 1,
('25', '138314'): 1,
('25', '1094367'): 1,
('25', '422912'): 1,
('25', '788104'): 1,
('25', '992602'): 1,
('25', '942867'): 1,
('25', '1030016'): 1,
('25', '1526567'): 1,
('25', '95722'): 1,
('25', '1860295'): 1,
('25', '499524'): 1,
('25', '730230'): 1,
('25', '177460'): 1,
('25', '1199618'): 1,
('25', '1229587'): 1,
('25', '4067'): 1,
('25', '1098536'): 1,
('25', '1200994'): 1,
('25', '210114'): 1,
('25', '1593893'): 1,
('25', '1634916'): 1,
('25', '1130945'): 1,
('25', '958524'): 1,
('25', '1625018'): 1,
('25', '396162'): 1,
('26', '803083'): 1,
('27', '1662028'): 1,
('28', '1682122'): 1,
('29', '720972'): 1,
```

```
('30', '917075'): 1,
('31', '679692'): 1,
('31', '742710'): 1,
('31', '538636'): 1,
('31', '577282'): 1,
('32', '1324739'): 1,
('32', '214896'): 1,
('32', '1116131'): 1,
('33', '700696'): 1,
('33', '1593284'): 1,
('34', '1518929'): 1,
('34', '1125104'): 1,
('36', '881549'): 1,
('36', '1242489'): 1,
('37', '575581'): 1,
('37', '743846'): 1,
('37', '874856'): 1,
('38', '1084554'): 1,
('38', '926898'): 1,
('38', '27623'): 1,
('38', '389596'): 1,
('38', '1125093'): 1,
('38', '811551'): 1,
('38', '1070165'): 1,
('38', '384225'): 1,
('39', '1652563'): 1,
('40', '1273789'): 1,
('41', '1229344'): 1,
('42', '186751'): 1,
('43', '406191'): 1,
('44', '899392'): 1,
('44', '81763'): 1,
('44', '826890'): 1,
('44', '611366'): 1,
('44', '1244292'): 1,
('45', '1844351'): 1,
('47', '1028853'): 1,
('49', '1854076'): 1,
('49', '1053836'): 1,
('49', '450855'): 1,
('49', '527674'): 1,
('49', '1206006'): 1,
('49', '1819413'): 1,
('49', '306893'): 1,
('49', '1315486'): 1,
('49', '1186111'): 1,
('50', '1065918'): 1,
('50', '1566117'): 1,
('51', '694124'): 1,
('51', '262606'): 1,
('51', '1060463'): 1,
('52', '1524285'): 1,
('53', '175439'): 1,
('53', '1134531'): 1,
('53', '989733'): 1,
('53', '1159365'): 1,
('53', '1066109'): 1,
('53', '321039'): 1,
('53', '1141033'): 1,
('53', '204553'): 1,
('53', '1364025'): 1,
('53', '813204'): 1,
('53', '528206'): 1,
('53', '1611430'): 1,
('53', '482808'): 1,
('53', '1386771'): 1,
('53', '1711782'): 1,
('53', '1007233'): 1,
('54', '946584'): 1,
('54', '867555'): 1,
('54', '1757684'): 1,
('54', '237014'): 1,
('54', '127561'): 1,
('54', '766708'): 1,
('54', '978393'): 1,
('54', '1844797'): 1,
('54', '1834300'): 1,
```

```
('54', '494483'): 1,
('54', '1101058'): 1,
('54', '761994'): 1,
('54', '679014'): 1,
('54', '609116'): 1,
('54', '577031'): 1,
('54', '1786857'): 1,
('54', '1194176'): 1,
('54', '1130353'): 1,
('54', '444771'): 1,
('54', '1336174'): 1,
('54', '230597'): 1,
('55', '279010'): 1,
('55', '1007648'): 1,
('55', '13589'): 1,
('56', '1707687'): 1,
('56', '1429792'): 1,
('57', '373871'): 1,
('58', '916140'): 1,
('59', '1389095'): 1,
('59', '1778028'): 1,
('60', '83366'): 1,
('60', '878202'): 1,
('60', '1168174'): 1,
('60', '296792'): 1,
('61', '1685730'): 1,
('61', '871534'): 1,
('61', '1705344'): 1,
('61', '208925'): 1,
('61', '1737228'): 1,
('61', '1031786'): 1,
('61', '1286823'): 1,
('61', '260181'): 1,
('61', '1408376'): 1,
('61', '803963'): 1,
('61', '1483341'): 1,
('61', '1499185'): 1,
('61', '122289'): 1,
('61', '1473016'): 1,
('61', '1740009'): 1,
('61', '89946'): 1,
('61', '817445'): 1,
('62', '1822914'): 1,
('62', '1601368'): 1,
('62', '402932'): 1,
('63', '268939'): 1,
('63', '1788531'): 1,
('63', '1630186'): 1,
('63', '1330009'): 1,
('63', '1765456'): 1,
('63', '1382381'): 1,
('63', '758777'): 1,
('63', '258969'): 1,
('63', '1732515'): 1,
('63', '732009'): 1,
('63', '1168913'): 1,
('63', '259286'): 1,
('64', '1709106'): 1,
('64', '1313246'): 1,
('64', '947828'): 1,
('65', '602159'): 1,
('65', '1812382'): 1,
('65', '171318'): 1,
('65', '442729'): 1,
('65', '76799'): 1,
('65', '1705481'): 1,
('65', '1119900'): 1,
('65', '1156069'): 1,
('65', '880042'): 1,
('65', '1673827'): 1,
('65', '1454964'): 1,
('65', '341285'): 1,
('65', '1405545'): 1,
('65', '557790'): 1,
('65', '1774222'): 1,
('65', '717601'): 1,
('66', '1741841'): 1,
```

('66', '1672048'): 1,
('67', '566600'): 1,
('67', '451245'): 1,
('67', '1541531'): 1,
('68', '1752782'): 1,
('68', '1045772'): 1,
('69', '131924'): 1,
('70', '871386'): 1,
('71', '746824'): 1,
('71', '1854232'): 1,
('71', '1575746'): 1,
('71', '1315204'): 1,
('71', '1674297'): 1,
('71', '1300876'): 1,
('71', '1105698'): 1,
('72', '1756934'): 1,
('73', '611306'): 1,
('73', '489181'): 1,
('73', '217164'): 1,
('73', '471806'): 1,
('73', '202447'): 1,
('74', '1061504'): 1,
('74', '1434864'): 1,
('74', '1709589'): 1,
('74', '1704181'): 1,
('74', '370494'): 1,
('74', '237909'): 1,
('74', '720583'): 1,
('74', '1400239'): 1,
('74', '142356'): 1,
('74', '1576900'): 1,
('74', '1481411'): 1,
('75', '1145079'): 1,
('75', '1247899'): 1,
('75', '512330'): 1,
('75', '1102064'): 1,
('75', '56206'): 1,
('75', '13121'): 1,
('76', '1183714'): 1,
('78', '1400162'): 1,
('79', '1595558'): 1,
('79', '419087'): 1,
('79', '1181051'): 1,
('79', '459884'): 1,
('79', '1716876'): 1,
('79', '253105'): 1,
('79', '792819'): 1,
('79', '973169'): 1,
('79', '348828'): 1,
('79', '1444240'): 1,
('79', '609109'): 1,
('79', '783907'): 1,
('79', '906140'): 1,
('79', '1041727'): 1,
('79', '756378'): 1,
('80', '361048'): 1,
('81', '1022657'): 1,
('81', '997656'): 1,
('81', '380597'): 1,
('81', '1698750'): 1,
('81', '1738080'): 1,
('81', '1490625'): 1,
('81', '869268'): 1,
('81', '719978'): 1,
('81', '711543'): 1,
('81', '167186'): 1,
('81', '276421'): 1,
('81', '98471'): 1,
('81', '980119'): 1,
('81', '1813867'): 1,
('81', '1398225'): 1,
('81', '1600230'): 1,
('82', '611092'): 1,
('82', '112566'): 1,
('82', '288597'): 1,
('83', '1521498'): 1,
('83', '19406'): 1,

('83', '1768082'): 1,
('83', '1711525'): 1,
('84', '628304'): 1,
('84', '1831501'): 1,
('84', '495318'): 1,
('84', '344428'): 1,
('84', '219656'): 1,
('84', '1812006'): 1,
('84', '489480'): 1,
('84', '1182549'): 1,
('84', '946366'): 1,
('84', '164496'): 1,
('84', '941898'): 1,
('84', '447831'): 1,
('84', '1362510'): 1,
('84', '1354719'): 1,
('84', '1498955'): 1,
('84', '1125968'): 1,
('84', '244887'): 1,
('84', '636422'): 1,
('84', '751223'): 1,
('84', '1354664'): 1,
('84', '597299'): 1,
('86', '907649'): 1,
('87', '1194997'): 1,
('87', '1003093'): 1,
('87', '109627'): 1,
('87', '1854354'): 1,
('87', '454942'): 1,
('87', '521421'): 1,
('87', '1159153'): 1,
('87', '508650'): 1,
('87', '231172'): 1,
('87', '1234791'): 1,
('87', '441051'): 1,
('87', '3347'): 1,
('87', '295799'): 1,
('87', '80780'): 1,
('87', '1184475'): 1,
('88', '201540'): 1,
('89', '948959'): 1,
('89', '878526'): 1,
('89', '1755151'): 1,
('89', '1820119'): 1,
('89', '412572'): 1,
('89', '436243'): 1,
('89', '1436178'): 1,
('89', '236707'): 1,
('89', '874621'): 1,
('89', '704901'): 1,
('89', '1024220'): 1,
('89', '627987'): 1,
('91', '800472'): 1,
('93', '232270'): 1,
('94', '1571211'): 1,
('94', '785283'): 1,
('95', '236966'): 1,
('96', '453195'): 1,
('98', '472174'): 1,
('98', '1144628'): 1,
('99', '237694'): 1,
('99', '1723995'): 1,
('99', '582267'): 1,
('101', '1058638'): 1,
('102', '124268'): 1,
('102', '429822'): 1,
('102', '837022'): 1,
('102', '531989'): 1,
('102', '793303'): 1,
('104', '644509'): 1,
('105', '1625115'): 1,
('105', '578479'): 1,
('105', '566519'): 1,
('105', '45974'): 1,
('105', '1154097'): 1,
('105', '1766921'): 1,
('105', '1407805'): 1,

```
('105', '1418762'): 1,
('105', '1391542'): 1,
('106', '266064'): 1,
('106', '1122174'): 1,
('106', '1784331'): 1,
('106', '1266933'): 1,
('106', '237219'): 1,
('107', '615816'): 1,
('107', '1321486'): 1,
('107', '933251'): 1,
('107', '486260'): 1,
('108', '358350'): 1,
('109', '1003858'): 1,
('110', '412863'): 1,
('111', '184581'): 1,
('111', '1519665'): 1,
('111', '297069'): 1,
('111', '1276690'): 1,
('111', '1390248'): 1,
('111', '1226594'): 1,
('111', '1850951'): 1,
('111', '1723555'): 1,
('111', '1178224'): 1,
('111', '1847174'): 1,
('111', '468743'): 1,
('111', '524048'): 1,
('111', '233077'): 1,
('111', '1724509'): 1,
('111', '1861931'): 1,
('111', '1121338'): 1,
('111', '1382582'): 1,
('111', '1384121'): 1,
('111', '806470'): 1,
('111', '1233289'): 1,
('111', '1775946'): 1,
('111', '1850147'): 1,
('111', '648943'): 1,
('111', '1273025'): 1,
('111', '671541'): 1,
('111', '747365'): 1,
('111', '175669'): 1,
('112', '675562'): 1,
('112', '821470'): 1,
('113', '547644'): 1,
('115', '1270945'): 1,
('116', '1394223'): 1,
('117', '245746'): 1,
('117', '1315537'): 1,
('117', '323398'): 1,
('117', '259972'): 1,
('117', '1618429'): 1,
('117', '296239'): 1,
('117', '981622'): 1,
('117', '1452096'): 1,
('117', '185553'): 1,
('117', '416110'): 1,
('117', '1082885'): 1,
('117', '1519107'): 1,
('117', '1405921'): 1,
('118', '146690'): 1,
('118', '707932'): 1,
('119', '1495706'): 1,
('119', '255000'): 1,
('119', '852555'): 1,
('119', '809936'): 1,
('119', '875039'): 1,
('119', '93610'): 1,
('119', '86004'): 1,
('119', '779396'): 1,
('119', '327613'): 1,
('119', '1293375'): 1,
('119', '261599'): 1,
('120', '862294'): 1,
('121', '103376'): 1,
('121', '487812'): 1,
('121', '385535'): 1,
('121', '508962'): 1,
```

```
('121', '508902'): 1,
('122', '1739931'): 1,
('122', '862700'): 1,
('122', '1405347'): 1,
('122', '1500867'): 1,
('122', '1171449'): 1,
('122', '581797'): 1,
('122', '252892'): 1,
('122', '1049126'): 1,
('122', '1638581'): 1,
('122', '1586443'): 1,
('122', '1687614'): 1,
('122', '1502201'): 1,
('122', '113017'): 1,
('122', '1032567'): 1,
('122', '603098'): 1,
('122', '1601261'): 1,
('122', '1789585'): 1,
('122', '1757805'): 1,
('122', '922781'): 1,
('122', '721778'): 1,
('122', '1646238'): 1,
('122', '1163641'): 1,
('122', '1418295'): 1,
('123', '525674'): 1,
('125', '1254433'): 1,
('125', '30404'): 1,
('125', '1171122'): 1,
('125', '344575'): 1,
('125', '652927'): 1,
('125', '666374'): 1,
('125', '671301'): 1,
('125', '1824545'): 1,
('125', '199458'): 1,
('125', '1608293'): 1,
('125', '1304583'): 1,
('125', '134946'): 1,
('126', '193651'): 1,
('127', '580880'): 1,
('127', '880649'): 1,
('127', '1049839'): 1,
('127', '1612915'): 1,
('127', '1073850'): 1,
('127', '178165'): 1,
('128', '857471'): 1,
('130', '499097'): 1,
('130', '282933'): 1,
('130', '904365'): 1,
('130', '953250'): 1,
('130', '202772'): 1,
('130', '485035'): 1,
('130', '4072'): 1,
('130', '1570182'): 1,
('132', '1730988'): 1,
('132', '482928'): 1,
('132', '982571'): 1,
('133', '1210717'): 1,
('133', '1336080'): 1,
('133', '37527'): 1,
('133', '819390'): 1,
('133', '345681'): 1,
('133', '1101843'): 1,
('133', '1325424'): 1,
('135', '612535'): 1,
('135', '593597'): 1,
('135', '1166489'): 1,
('135', '1746180'): 1,
('135', '168795'): 1,
('135', '347708'): 1,
('135', '36562'): 1,
('135', '428269'): 1,
('135', '161908'): 1,
('135', '717660'): 1,
('135', '684382'): 1,
('135', '1146696'): 1,
('135', '1623787'): 1,
('135', '839774'): 1,
('135', '100378'): 1,
```

```
('135', '100578'): 1,
('135', '71740'): 1,
('135', '1498540'): 1,
('135', '808799'): 1,
('135', '1647182'): 1,
('135', '1578593'): 1,
('135', '1025388'): 1,
('135', '1769211'): 1,
('135', '956320'): 1,
('135', '1643728'): 1,
('135', '925682'): 1,
('135', '564165'): 1,
('135', '951962'): 1,
('135', '962111'): 1,
('135', '969308'): 1,
('135', '607835'): 1,
('135', '1832076'): 1,
('135', '508098'): 1,
('135', '1084361'): 1,
('135', '618863'): 1,
('135', '1646757'): 1,
('135', '1617835'): 1,
('135', '250674'): 1,
('135', '1465671'): 1,
('135', '491567'): 1,
('135', '740672'): 1,
('135', '1231569'): 1,
('135', '953125'): 1,
('135', '242147'): 1,
('135', '1835285'): 1,
('135', '1844669'): 1,
('135', '822004'): 1,
('135', '506857'): 1,
('136', '433374'): 1,
('136', '1527751'): 1,
('136', '1222229'): 1,
('136', '1541805'): 1,
('136', '1483192'): 1,
('136', '1721537'): 1,
('136', '61698'): 1,
('136', '1427267'): 1,
('136', '754689'): 1,
('136', '1146329'): 1,
('137', '1768787'): 1,
('138', '479543'): 1,
('138', '1007606'): 1,
('139', '1102059'): 1,
('139', '1364631'): 1,
('140', '1122310'): 1,
('140', '100170'): 1,
('140', '625577'): 1,
('141', '368227'): 1,
('142', '694465'): 1,
('142', '1068238'): 1,
('143', '1608536'): 1,
('143', '29868'): 1,
('143', '30678'): 1,
('143', '20832'): 1,
('143', '385744'): 1,
('143', '1003694'): 1,
('143', '1784446'): 1,
('143', '283299'): 1,
('143', '1579486'): 1,
('145', '1790320'): 1,
('146', '44123'): 1,
('146', '1804972'): 1,
('146', '882846'): 1,
('146', '1829224'): 1,
('147', '775423'): 1,
('147', '1065996'): 1,
('148', '1315276'): 1,
('148', '1165746'): 1,
('148', '143737'): 1,
('148', '1773940'): 1,
('148', '679568'): 1,
('149', '1165815'): 1,
('150', '1455822'): 1,
('151', '1741895'): 1,
```

('151', '1741895'): 1,
('152', '1319086'): 1,
('152', '629024'): 1,
('152', '150431'): 1,
('152', '1760659'): 1,
('152', '1536764'): 1,
('152', '588919'): 1,
('152', '376997'): 1,
('152', '1675448'): 1,
('152', '936670'): 1,
('153', '527826'): 1,
('153', '166580'): 1,
('153', '626388'): 1,
('153', '1045001'): 1,
('154', '1273853'): 1,
('154', '962460'): 1,
('154', '93121'): 1,
('154', '1304780'): 1,
('154', '380280'): 1,
('154', '736969'): 1,
('154', '767212'): 1,
('154', '591467'): 1,
('154', '1857731'): 1,
('154', '27544'): 1,
('154', '1812401'): 1,
('154', '652098'): 1,
('154', '1823990'): 1,
('154', '813453'): 1,
('154', '1714716'): 1,
('154', '296977'): 1,
('154', '1739769'): 1,
('154', '248243'): 1,
('154', '1359304'): 1,
('154', '280203'): 1,
('154', '26862'): 1,
('154', '496603'): 1,
('154', '1030041'): 1,
('154', '1088677'): 1,
('154', '371906'): 1,
('154', '745099'): 1,
('154', '1269699'): 1,
('154', '1622333'): 1,
('154', '1534571'): 1,
('154', '830762'): 1,
('154', '667918'): 1,
('154', '234148'): 1,
('154', '1692748'): 1,
('154', '1271971'): 1,
('154', '908717'): 1,
('154', '505050'): 1,
('154', '1081991'): 1,
('155', '611954'): 1,
('155', '990325'): 1,
('155', '1594405'): 1,
('155', '391505'): 1,
('155', '1259916'): 1,
('155', '50571'): 1,
('156', '332995'): 1,
('156', '1474531'): 1,
('156', '1709921'): 1,
('156', '1118944'): 1,
('156', '1164808'): 1,
('156', '1365638'): 1,
('157', '1110246'): 1,
('159', '217322'): 1,
('160', '873150'): 1,
('160', '504273'): 1,
('160', '935558'): 1,
('160', '1080124'): 1,
('160', '1065503'): 1,
('160', '1223149'): 1,
('161', '612043'): 1,
('161', '1542729'): 1,
('163', '1579539'): 1,
('163', '1614761'): 1,
('163', '1772229'): 1,
('163', '1746941'): 1,

```
('163', '742658'): 1,
('163', '928359'): 1,
('163', '705565'): 1,
('163', '1533384'): 1,
('163', '315604'): 1,
('163', '810753'): 1,
('163', '1294153'): 1,
('164', '1478790'): 1,
('164', '1188073'): 1,
('164', '1847742'): 1,
('165', '139728'): 1,
('166', '1281467'): 1,
('166', '536060'): 1,
('166', '663272'): 1,
('167', '597139'): 1,
('168', '952523'): 1,
('168', '1063176'): 1,
('168', '204709'): 1,
('168', '942370'): 1,
('168', '1608317'): 1,
('168', '1149931'): 1,
('168', '1166908'): 1,
('170', '930854'): 1,
('170', '177989'): 1,
('170', '789700'): 1,
('170', '1778092'): 1,
('170', '509793'): 1,
('170', '1861582'): 1,
('170', '1252327'): 1,
('170', '1672986'): 1,
('170', '705856'): 1,
('170', '676802'): 1,
('170', '545836'): 1,
('170', '358857'): 1,
('171', '652545'): 1,
('171', '495157'): 1,
('171', '956716'): 1,
('171', '1784396'): 1,
('171', '665233'): 1,
('171', '1773998'): 1,
('171', '124205'): 1,
('171', '1686016'): 1,
('171', '167372'): 1,
('171', '1072213'): 1,
('171', '1210546'): 1,
('171', '382635'): 1,
('171', '391144'): 1,
('171', '1405597'): 1,
('171', '1622184'): 1,
('171', '429565'): 1,
('171', '1262028'): 1,
('171', '651379'): 1,
('171', '1131806'): 1,
('171', '183303'): 1,
('171', '1730708'): 1,
('171', '255580'): 1,
('171', '597356'): 1,
('171', '1071272'): 1,
('171', '143917'): 1,
('171', '1170552'): 1,
('171', '1471278'): 1,
('171', '1756610'): 1,
('171', '127181'): 1,
('171', '1571476'): 1,
('171', '1267832'): 1,
('171', '1825254'): 1,
('171', '927274'): 1,
('171', '912681'): 1,
('171', '799586'): 1,
('171', '187805'): 1,
('171', '1400484'): 1,
('171', '864239'): 1,
('172', '4769'): 1,
('172', '906059'): 1,
('172', '372997'): 1,
('172', '1208963'): 1,
('172', '346991'): 1,
```

('172', '1172153'): 1,
('172', '161309'): 1,
('172', '200086'): 1,
('172', '642842'): 1,
('172', '153393'): 1,
('172', '625473'): 1,
('172', '1453902'): 1,
('172', '703254'): 1,
('172', '1591526'): 1,
('172', '94719'): 1,
('172', '1851039'): 1,
('172', '1556441'): 1,
('172', '1359168'): 1,
('172', '1233403'): 1,
('172', '891719'): 1,
('174', '1454201'): 1,
('175', '262454'): 1,
('175', '945369'): 1,
('175', '979716'): 1,
('176', '1410138'): 1,
('177', '1337513'): 1,
('178', '185820'): 1,
('178', '31865'): 1,
('178', '336157'): 1,
('179', '1483220'): 1,
('179', '271130'): 1,
('179', '1084031'): 1,
('179', '861489'): 1,
('179', '781383'): 1,
('179', '1228513'): 1,
('179', '1567328'): 1,
('179', '96588'): 1,
('179', '1497030'): 1,
('179', '462643'): 1,
('179', '330270'): 1,
('179', '1786977'): 1,
('179', '310769'): 1,
('179', '232801'): 1,
('179', '1503576'): 1,
('179', '486560'): 1,
('179', '707486'): 1,
('179', '934657'): 1,
('179', '529357'): 1,
('180', '1272584'): 1,
('181', '541439'): 1,
('181', '337912'): 1,
('181', '409469'): 1,
('181', '446899'): 1,
('181', '1673573'): 1,
('182', '24725'): 1,
('182', '1418688'): 1,
('183', '455180'): 1,
('185', '1357074'): 1,
('186', '1265993'): 1,
('186', '327167'): 1,
('186', '711951'): 1,
('186', '83430'): 1,
('186', '1442515'): 1,
('186', '311232'): 1,
('186', '974920'): 1,
('186', '699686'): 1,
('186', '236128'): 1,
('186', '388013'): 1,
('186', '1631284'): 1,
('186', '1458006'): 1,
('188', '153986'): 1,
('189', '1749543'): 1,
('190', '64429'): 1,
('190', '1048960'): 1,
('190', '526223'): 1,
('190', '1289987'): 1,
('190', '271006'): 1,
('190', '1285203'): 1,
('191', '683870'): 1,
('191', '585770'): 1,
('191', '1157784'): 1,
('191', '1414695'): 1,

```
('191', '572524'): 1,
('192', '1009108'): 1,
('192', '576042'): 1,
('192', '1193622'): 1,
('192', '1519661'): 1,
('192', '1032428'): 1,
('192', '1566459'): 1,
('192', '1152050'): 1,
('192', '1370403'): 1,
('192', '377436'): 1,
('192', '1725345'): 1,
('192', '395920'): 1,
('192', '1272965'): 1,
('192', '135660'): 1,
('192', '13971'): 1,
('192', '656285'): 1,
('192', '128661'): 1,
('192', '1466700'): 1,
('192', '1289600'): 1,
('192', '1524820'): 1,
('192', '144188'): 1,
('192', '1654697'): 1,
('193', '6569'): 1,
('193', '1677350'): 1,
('193', '1026303'): 1,
('193', '479169'): 1,
('193', '1611937'): 1,
('193', '1571677'): 1,
('193', '1153484'): 1,
('193', '704075'): 1,
('193', '781535'): 1,
('193', '659531'): 1,
('193', '1760872'): 1,
('194', '1561139'): 1,
('195', '316120'): 1,
('195', '667575'): 1,
('197', '1682373'): 1,
('197', '500407'): 1,
('197', '1125507'): 1,
('197', '1423204'): 1,
('197', '257310'): 1,
('197', '916907'): 1,
('198', '1186809'): 1,
('199', '157525'): 1,
('200', '860695'): 1,
('201', '625238'): 1,
('202', '684918'): 1,
('202', '1759109'): 1,
('202', '1127522'): 1,
('202', '427502'): 1,
('203', '1388870'): 1,
('203', '974831'): 1,
('203', '1628507'): 1,
('203', '252539'): 1,
('203', '917890'): 1,
('204', '119966'): 1,
('204', '541063'): 1,
('205', '1006837'): 1,
('205', '946356'): 1,
('205', '66945'): 1,
('205', '1469189'): 1,
('206', '964836'): 1,
('206', '1559022'): 1,
('206', '1353017'): 1,
('206', '1075632'): 1,
('206', '1467586'): 1,
('206', '1058463'): 1,
('206', '292683'): 1,
('207', '848686'): 1,
('207', '1468946'): 1,
('207', '1086203'): 1,
('207', '495569'): 1,
('207', '229654'): 1,
('207', '1668614'): 1,
('207', '55206'): 1,
('208', '779848'): 1,
('209', '1605441'): 1,
```

```
 ('209', '1680525'): 1,
 ('209', '766605'): 1,
 ('209', '1532595'): 1,
 ('209', '1677577'): 1,
 ('211', '479718'): 1,
 ('211', '771626'): 1,
 ('211', '854473'): 1,
 ('211', '888971'): 1,
 ('211', '1791428'): 1,
 ('211', '1376720'): 1,
 ('211', '1605669'): 1,
 ('211', '1562874'): 1,
 ('211', '319655'): 1,
 ('211', '1756947'): 1,
 ('212', '1145629'): 1,
 ('212', '791075'): 1,
 ('212', '404628'): 1,
 ('212', '137141'): 1,
 ('212', '1691472'): 1,
 ('213', '1650053'): 1,
 ('213', '1850048'): 1,
 ('213', '479034'): 1,
 ('213', '1365529'): 1,
 ('213', '1709815'): 1,
 ('214', '595373'): 1,
 ('214', '837744'): 1,
 ('214', '472064'): 1,
 ('214', '290758'): 1,
 ('214', '838744'): 1,
 ('214', '1383691'): 1,
 ('214', '23720'): 1,
 ('214', '929669'): 1,
 ('214', '271427'): 1,
 ('215', '1310257'): 1,
 ('216', '1780520'): 1,
 ('216', '29376'): 1,
 ('216', '1352408'): 1,
 ('216', '18447'): 1,
 ('216', '385559'): 1,
 ('216', '850654'): 1,
 ('216', '863514'): 1,
 ('216', '1239300'): 1,
 ('216', '995717'): 1,
 ('216', '1805592'): 1,
 ('216', '1216771'): 1,
 ('216', '1198938'): 1,
 ('216', '530173'): 1,
 ('216', '1287580'): 1,
 ...}
```

In [34]:

```
%%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: 11.5 s
Parser   : 101 ms
```

In [35]:

```
missing_edges = pickle.load(open('data/after_eda/missing_edges_final.p','rb'))
len(missing_edges)
```

Out[35]:

```
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 [36]:

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

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

    #Trian test split
    #Spiltted data into 80-20
    #positive links and negative links seperatly because we need positive training data only for c
reating 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_pc
s.shape[0])
    print("Number of nodes in the train data graph without edges", X_train_neg.shape[0],"=", y_trai
n_neg.shape[0])
    print('='*60)
    print("Number of nodes in the test data graph with edges", X_test_pos.shape[0],"=",y_test_pos.s
hape[0])
    print("Number of nodes in the test data graph without edges",
X_test_neg.shape[0],"=",y_test_neg.shape[0])

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

In [ ]:

```
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=n
x.DiGraph(),nodetype=int)
```

```
    test_graph=nx.read_edgelist('data/after_eda/test_pos_after_eda.csv',delimiter=',',create_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(te
Y_trN/len(test_nodes_pos)*100))
```

> we have a cold start problem here

In [3]:

```
#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', 'dest
ination_node'])
    X_test_pos = pd.read_csv('data/after_eda/test_pos_after_eda.csv', names=['source_node', 'destin
ation_node'])
    X_train_neg = pd.read_csv('data/after_eda/train_neg_after_eda.csv', names=['source_node', 'dest
ination_node'])
    X_test_neg = pd.read_csv('data/after_eda/test_neg_after_eda.csv', names=['source_node', 'destin
ation_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)
```

In [4]:

```
X_train = pd.read_csv('data/after_eda/train_after_eda.csv')
X_test = pd.read_csv('data/after_eda/test_after_eda.csv')
y_train = pd.read_csv('data/train_y.csv')
y_test = pd.read_csv('data/test_y.csv')
```

In [5]:

```
print("Data points in train data",X_train.shape)
print("Data points in test data",X_test.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 (15100029, 2)
Data points in test data (3775007, 2)
Shape of traget variable in train (15100029, 1)
Shape of traget variable in test (3775007, 1)
```

In [6]:

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

2nd Notebook:

Social network Graph Link Prediction - Facebook Challenge

In [7]:

```
#Importing Libraries
# please do go through this python notebook:
import warnings
warnings.filterwarnings("ignore")

import csv
import pandas as pd#pandas to create small dataframes
import datetime #Convert to unix time
import time #Convert to unix time
# if numpy is not installed already : pip3 install numpy
import numpy as np#Do aritmetic operations on arrays
# matplotlib: used to plot graphs
import matplotlib
import matplotlib.pylab as plt
import seaborn as sns#Plots
from matplotlib import rcParams#Size of plots
from sklearn.cluster import MiniBatchKMeans, KMeans#Clustering
import math
import pickle
import os
# to install xgboost: pip3 install xgboost
import xgboost as xgb

import warnings
import networkx as nx
import pdb
import pickle
from pandas import HDFStore,DataFrame
from pandas import read_hdf
from scipy.sparse.linalg import svds, eigs
import gc
from tqdm import tqdm
```

1. Reading Data

In [8]:

```
if os.path.isfile('data/after_eda/train_pos_after_eda.csv'):

train_graph=nx.read_edgelist('data/after_eda/train_pos_after_eda.csv',delimiter=',',create_using=n
x.DiGraph(),nodetype=int)
    print(nx.info(train_graph))
else:
    print("please run the FB_EDA.ipynb or download the files from drive")
```

```
Name:
Type: DiGraph
Number of nodes: 1780722
```

```
Number of edges: 7550015
Average in degree:   4.2399
Average out degree:   4.2399
```

2. Similarity measures

2.1 Jaccard Distance:

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

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

In [9]:

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

(len(set(train_graph.successors(a)).union(set(train_graph.successors(b)))))
    except:
        return 0
    return sim
```

In [10]:

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

```
0.0
```

In [11]:

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

```
0.0
```

In [12]:

```
#for followers
def jaccard_for_followers(a,b):
    try:
        if len(set(train_graph.predecessors(a))) == 0  | len(set(g.predecessors(b))) == 0:
            return 0
        sim = (len(set(train_graph.predecessors(a)).intersection(set(train_graph.predecessors(b))))
)/\
                                 (len(set(train_graph.predecessors(a)).union(set(train_graph.predec
ssors(b)))))
        return sim
    except:
        return 0
```

In [13]:

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

```
0
```

In [14]:

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

0

2.2 Cosine distance

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

In [15]:

```
#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.successors(b))))))
        return sim
    except:
        return 0
```

In [16]:

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

0.0

In [17]:

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

0

In [18]:

```
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(tra
n_graph.predecessors(b)))))
        return sim
    except:
        return 0
```

In [19]:

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

0.02886751345948129

In [20]:

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

0

3. Ranking Measures

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

PageRank computes a ranking of the nodes in the graph G based on the structure of the incoming links.

Mathematical PageRanks for a simple network, expressed as percentages. (Google uses a logarithmic scale.) Page C has a higher PageRank than Page E, even though there are fewer links to C; the one link to C comes from an important page and hence is of high value. If web surfers who start on a random page have an 85% likelihood of choosing a random link from the page they are currently visiting, and a 15% likelihood of jumping to a page chosen at random from the entire web, they will reach Page E 8.1% of the time. **(The 15% likelihood of jumping to an arbitrary page corresponds to a damping factor of 85%.) Without damping, all web surfers would eventually end up on Pages A, B, or C, and all other pages would have PageRank zero. In the presence of damping, Page A effectively links to all pages in the web, even though it has no outgoing links of its own.**

3.1 Page Ranking

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

In [21]:

```
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 [22]:

```
pr[min(pr, key=pr.get)]
```

Out[22]:

```
1.6556497245737814e-07
```

In [23]:

```
(pr.get)
```

Out[23]:

```
<function dict.get(key, default=None, /)>
```

In [24]:

```
min(pr,key=pr.get)
```

Out[24]:

```
527014
```

In [25]:

```
pr.get(3.459962832379924e-07)
```

In [26]:

```
pr
```

Out[26]:

```
{273084: 2.0452904537613205e-06,
 1505602: 3.459962832379924e-07,
 912810: 1.039181158882892e-06,
 1678443: 1.7938059019480253e-06,
 365429: 1.033021623853361e-06,
 1523458: 3.096855642103832e-06,
 527014: 1.6556497245737814e-07,
 1605979: 6.428994469008903e-07,
```

1228116: 8.348032485214042e-07,
471233: 2.6658762149907754e-06,
866691: 1.5084114559632217e-06,
535232: 1.2802253685038755e-06,
660560: 1.4090493321512168e-06,
1272982: 1.2307217955919252e-06,
1409846: 1.7424392231291614e-06,
845593: 1.3746924926171e-06,
628879: 4.3605450797536837e-07,
858706: 8.815666841226465e-07,
1114859: 1.6556497245737814e-07,
813966: 9.482419757909467e-07,
976987: 1.0694172129407271e-06,
182524: 2.848985910682562e-07,
1408148: 2.0895901558860652e-07,
973346: 2.265836660088316e-06,
1085939: 5.465731937112257e-07,
569150: 8.059887256690902e-07,
396322: 7.515436589181799e-06,
149376: 1.841840696914121e-06,
117851: 5.042451471815843e-07,
598891: 5.042451471815843e-07,
1046713: 1.738626852658722e-06,
1790645: 1.5043745222060861e-06,
1038318: 5.7262520483004e-07,
1593467: 3.472432137074377e-06,
70574: 7.166902835298391e-07,
1328148: 3.764618765214357e-07,
1814022: 2.848985910682562e-07,
1791177: 8.815666841226464e-07,
1757093: 4.6962691525976323e-07,
912379: 8.759933076103603e-07,
1570978: 3.4361830816249777e-07,
1499086: 2.251325164653729e-06,
333578: 3.246764639385488e-07,
879520: 8.02010938382061e-07,
463464: 1.2926047037823373e-06,
745738: 2.451207181979635e-07,
791618: 3.3263203851260736e-07,
1356611: 8.454049815132893e-07,
546636: 5.254600127124071e-07,
283651: 4.823673171029233e-07,
882823: 2.0144008977942957e-06,
1306607: 1.1781184753069518e-06,
436949: 6.066111421036863e-07,
36283: 6.367136221548575e-07,
1355641: 1.781264059953653e-07,
642709: 1.8959361279336687e-06,
1566025: 2.5814077847798203e-06,
1225769: 1.0362028841473486e-06,
1722386: 2.0749028702314266e-06,
948062: 8.461432781851012e-07,
679765: 6.832455750979015e-07,
1773023: 7.628013208384869e-07,
1802245: 1.189750489646295e-06,
686760: 7.798644083864676e-07,
989325: 5.95014146564786e-07,
562993: 3.146912988925782e-06,
1722833: 7.821220019469146e-07,
544361: 6.230105104657439e-07,
334572: 1.6556497245737814e-07,
1169048: 6.428994469008903e-07,
424175: 1.021500239860429e-06,
1404855: 5.965698498511969e-07,
1770842: 7.202208584352477e-07,
1477860: 6.178540272856873e-07,
1687319: 1.6556497245737814e-07,
750411: 1.247523114529339e-06,
1106259: 7.257700153806667e-07,
154784: 2.6103186734608056e-07,
1662480: 1.6556497245737814e-07,
1551903: 2.419220784801087e-06,
562675: 4.996991045678367e-07,
219692: 5.832326375954513e-07,
708359: 4.042322096791342e-07,
888200: 4.476262528103626e-07,
730672: 2.4736383816731444e-06,

1840032: 1.2340505567282848e-06,
1825190: 8.815666841226464e-07,
1686004: 3.246764639385488e-07,
501189: 1.6556497245737814e-07,
61: 7.697806619025419e-07,
203305: 1.3070334880443694e-06,
725502: 3.0442591047730896e-07,
301556: 4.661720402914471e-07,
1756878: 2.008391167217788e-06,
275557: 3.4881855022556557e-07,
839980: 3.5996888127604563e-07,
842723: 8.392197408416998e-07,
524527: 4.6623760821376076e-07,
276086: 3.1598550011814876e-07,
661895: 6.057957167445669e-07,
1542325: 1.7689887569574746e-06,
1079302: 7.7578831024002e-06,
1791610: 4.64866588865202e-07,
1321724: 2.6103186734608056e-07,
832016: 2.3534578623285186e-07,
1543415: 6.427659659497299e-07,
1831478: 1.3905512582785386e-06,
561277: 1.327078860269924 3e-06,
1122952: 1.2869393703735384e-06,
720171: 1.7899297789489588e-06,
1452769: 1.5547856576293684e-06,
154251: 2.536077154958819e-06,
132017: 7.6674996682671e-07,
970018: 1.249417298951734e-06,
1402038: 2.02690987136318e-07,
182580: 1.1041951426576642e-06,
1284805: 5.733451532438204e-06,
1252082: 2.202787454691224e-07,
1172937: 8.321154338013055e-07,
1098851: 2.1866172860334186e-07,
65493: 7.948262736117935e-07,
26839: 4.225280159962275e-07,
531012: 1.0641552758865631e-06,
614833: 1.3197570975227974e-06,
438469: 1.6556497245737814e-07,
1286875: 2.8110776459952012e-06,
1077631: 3.3793575489531305e-07,
1517929: 8.815666841226464e-07,
321689: 7.477187627975572e-07,
1539523: 1.0073475100616797e-06,
1252341: 1.1141725311927388e-06,
1374700: 1.9966029206048615e-07,
1553030: 1.920835543709066e-07,
1283053: 7.383663417895928e-07,
1561705: 1.419008593659695e-06,
648602: 1.1281479656717669e-06,
227087: 2.1329841990172935e-07,
1736008: 7.3398184333227e-07,
392731: 1.6556497245737814e-07,
267971: 7.980331510950318e-07,
1268337: 2.1827760818409467e-07,
956952: 3.715055823757338e-07,
1492633: 2.4185251443122455e-06,
1370536: 2.817230465954177e-07,
151196: 6.362698014225082e-07,
384061: 1.2233544400933129e-06,
983414: 1.653005229404363e-06,
1490571: 2.2485362191794104e-06,
1728893: 6.4891084325695e-07,
1575103: 5.700405589759644e-07,
322123: 4.042322096791342e-07,
1677099: 6.042735124304612e-07,
1622310: 1.7700358497352313e-06,
1645746: 4.19148912005494e-07,
1496122: 5.216218387387488e-07,
716889: 1.1257193943781065e-06,
131993: 1.885185950446591e-06,
1130115: 2.2773304739462613e-06,
443328: 1.1998395345481853e-06,
1193183: 1.1049577344821517e-06,
331160: 2.5922929334947988e-06,
1036556: 2.6754933082706956e-06,

79771: 1.5498404501570903e-06,
209829: 2.223861878345837e-06,
1325247: 6.211018987265075e-07,
760242: 5.179801024857924e-07,
1612865: 6.524547962713255e-07,
1659365: 2.066893260650331e-06,
364413: 5.757305626997536e-07,
591806: 3.288211822288675e-06,
1201562: 2.2834028668974645e-06,
550101: 5.639000350466027e-07,
1092078: 3.904907626875786e-07,
1019460: 9.420481251901643e-07,
696941: 5.474783360241401e-07,
1289468: 9.888494997249078e-07,
1368400: 2.9981529339461594e-07,
1006992: 1.704245418485518e-06,
335122: 7.698270230597332e-07,
1021192: 3.7564724475935674e-07,
826445: 4.125339475602225e-07,
1364513: 3.6705674309919015e-07,
1579986: 6.786995324841537e-07,
393085: 3.530892302744722e-07,
85816: 9.59846812499545e-07,
1506520: 1.9068783953335247e-07,
1538840: 2.826081903894172e-06,
1141034: 2.4952606144707123e-07,
583350: 5.766029921170691e-07,
751732: 7.58261963581524e-07,
1400831: 1.3155578687017503e-06,
1754364: 2.6279977280698244e-07,
1431711: 8.971162162446698e-07,
672511: 1.1054791057099567e-06,
427712: 4.349179973219314e-07,
363634: 1.7936164835057857e-06,
258962: 4.4539014834246463e-07,
496866: 2.371163353791835e-06,
1811820: 1.4127329781098869e-06,
1395145: 5.418906808771705e-07,
1677755: 7.135970949014525e-07,
298631: 1.4392591471209348e-06,
739876: 6.428994469008903e-07,
852055: 1.3597601322430988e-06,
201818: 9.293001315669976e-07,
992764: 3.7240991138290006e-07,
1711901: 1.282919568738907e-06,
1539921: 2.1131807970856097e-06,
1163581: 2.2387571791496628e-07,
1222329: 9.60361970528947e-07,
554602: 4.953597002547138e-07,
513262: 1.9075812497478914e-06,
1222366: 2.655779099598283e-07,
1493194: 9.102134379460421e-07,
536820: 2.555559523795208e-07,
226930: 7.900775765209734e-07,
150725: 6.408407675154979e-07,
1426561: 1.2077021972523907e-06,
1136749: 1.3285529010625231e-06,
143756: 1.8088836521618618e-06,
1796497: 8.229697396803144e-06,
1321708: 9.475696430755246e-06,
175024: 2.5336743373354695e-06,
205843: 5.42545556202409e-07,
369819: 2.2161692526708737e-06,
486651: 2.4960119288939756e-07,
24884: 1.4153650742119117e-06,
1515472: 1.7377089017463304e-06,
1014884: 1.018137380977318e-06,
740353: 8.152702293388252e-07,
198080: 1.9539837711009766e-07,
851163: 1.44419409750919e-06,
570124: 9.794713671454386e-07,
220362: 8.661376910093207e-07,
606966: 1.7653728856021655e-06,
1224294: 9.35310846962992e-07,
1316661: 1.3634497392930281e-06,
1170389: 1.7947329317527943e-06,
1431257: 3.802376151660755e-07,

671321: 2.601114057162728e-07,
1163375: 6.835546262405028e-07,
278734: 5.519785946259356e-07,
101504: 3.0459780758883195e-07,
1691337: 1.467965742491569e-06,
1116394: 1.6556497245737814e-07,
1030314: 6.428994469008903e-07,
529989: 6.518484064252917e-07,
99844: 5.764289983900896e-07,
1801683: 7.238256907823848e-07,
1245657: 2.7826894558987406e-07,
764724: 1.9171464909743544e-06,
584026: 1.804802354238633e-06,
398432: 6.898460792774775e-07,
1244349: 6.997249795727369e-07,
742520: 2.1329841990172935e-07,
124581: 2.1140369141765626e-06,
130375: 2.417427481840292e-07,
209755: 4.824578617601297e-06,
1381070: 3.799263094420106e-07,
117351: 2.5001645639738415e-07,
425049: 6.138007055971836e-06,
1307553: 5.57604996180994e-07,
464647: 6.554010640886966e-07,
590147: 4.5602136570949614e-07,
1668806: 2.2781362530993315e-06,
1430666: 9.029769964387037e-07,
569324: 6.275565530794917e-07,
1194578: 4.77051679279797e-07,
587091: 1.425334465908727e-06,
298888: 5.191394183766263e-07,
1829309: 6.135894353122536e-07,
1310254: 2.848985910682562e-07,
1785474: 1.3367272793837176e-06,
1478421: 4.290168843137012e-07,
1488737: 5.85318889669068e-07,
457662: 1.4008324913535024e-06,
605761: 6.35384498114929e-07,
1236701: 2.4739373950483736e-07,
873749: 1.5271789071177992e-06,
93650: 7.363705182030693e-07,
1144584: 4.2014335882725124e-07,
1298861: 4.996991045678367e-07,
959250: 6.826773197711829e-07,
1189845: 4.042322096791342e-07,
707628: 1.354913567587685e-06,
681447: 1.2873449954359037e-06,
1050142: 1.8793070887401622e-06,
773379: 1.8606123294609527e-06,
1336855: 2.6755240602541587e-06,
1290325: 9.378043664565084e-07,
819936: 4.880936696906324e-07,
1783381: 3.259664665893349e-06,
1076063: 2.848985910682562e-07,
58393: 7.582552782247392e-07,
1331983: 3.903472378950631e-07,
270629: 2.9938594145607136e-06,
1430596: 5.976690657675505e-06,
400599: 7.277832990242691e-07,
1457021: 2.0144437518254062e-06,
59618: 4.85429238881857e-07,
196674: 4.151069821857665e-07,
421096: 1.2401845752821743e-06,
868419: 1.6425801992990353e-06,
144309: 2.451207181979635e-07,
768675: 1.6556497245737814e-07,
338516: 1.0802561895399825e-06,
669549: 1.6556497245737814e-07,
719049: 4.996991045678367e-07,
38611: 1.962804429281559e-06,
1664439: 2.2773879055716332e-07,
1371356: 1.035429451672084e-06,
325548: 3.514500322166305e-07,
132660: 3.1483177299406166e-07,
449350: 1.241862417236259e-06,
686204: 1.6869528814133762e-06,
716722: 2.4877474738814806e-06,

1082463: 1.205489858506713e-06,
640572: 1.8726199402299234e-07,
423597: 3.725304717561864e-07,
1169977: 3.3134799274291216e-06,
21320: 4.6299969116824117e-07,
1134485: 6.54267412251264e-07,
1090680: 1.4987226439902762e-06,
1258804: 1.4787911110633345e-06,
1008202: 5.655875811683727e-07,
960995: 1.6088490830165409e-06,
1462461: 1.4384569043067437e-06,
1600294: 1.9210950951329615e-06,
940996: 1.6180255875497794e-06,
1780134: 5.678897437740526e-07,
679604: 6.680503209770881e-07,
1001122: 3.451408112338334e-06,
381406: 2.848985910682562e-07,
181524: 8.897240305156123e-07,
1076923: 1.315183472558079e-06,
1402350: 4.7254955639535513e-07,
1218356: 1.3303729191939423e-06,
128623: 1.5622092386907159e-06,
944692: 4.1572470808363677e-07,
559417: 4.305785930088086e-07,
132222: 2.3736396588582715e-06,
1858489: 1.13367459795037e-06,
413266: 1.4217278118853202e-06,
387141: 4.774596574630821e-07,
130676: 1.0862949548798042e-06,
632613: 4.166022882040599e-07,
335520: 1.6556497245737814e-07,
573770: 2.6843553795385293e-06,
1443670: 7.813264444895089e-07,
271786: 2.848985910682562e-07,
1011666: 3.1672088936449034e-07,
450550: 5.805109936622207e-07,
170193: 1.4482185675059502e-06,
1034617: 2.053428453276708e-07,
95630: 7.774149706761755e-07,
1626277: 5.689066635567434e-07,
1295556: 3.058380666657837e-07,
963526: 8.828118305781181e-07,
1226438: 1.1088688148100331e-06,
714629: 6.11633482490954e-07,
910865: 1.2103273141085217e-06,
1597016: 2.5269745588754305e-07,
1814537: 6.228469431756712e-07,
613441: 2.1215428225446808e-06,
397744: 1.1809721053172119e-06,
1238388: 5.266204285810144e-07,
690085: 4.042322096791342e-07,
1100574: 7.622330655117683e-07,
1344889: 2.106006999139705e-06,
154548: 6.464983973034406e-07,
180103: 2.953104309959461e-07,
516777: 1.974487460744753e-06,
658286: 3.070622152481846e-07,
1287262: 5.465026796593757e-07,
428941: 5.418150286562157e-06,
1316097: 5.467271186224247e-07,
1320959: 3.9627663510507573e-07,
1310495: 3.0765513938328194e-06,
1215945: 7.4757805971745e-07,
175973: 4.440100825494269e-07,
1411900: 2.765396231675111e-06,
1315420: 2.3375561166359416e-07,
1641402: 9.232387414153339e-07,
1162798: 1.584033587096984e-06,
766587: 4.415185015015074e-07,
612474: 9.607435929787526e-07,
10432: 4.964670696093447e-07,
153962: 6.497185108215119e-07,
884929: 1.0163873617544212e-06,
1851310: 9.551764545923658e-07,
733044: 8.815666841226464e-07,
1115294: 6.826773197711829e-07,
53024: 1.711219461131608e-06,

470337: 9.779806712225463e-07,
1618992: 7.124160152027828e-07,
157861: 3.1406903117313744e-07,
1191128: 1.0804560484741096e-06,
1653707: 6.428994469008903e-07,
314714: 2.451207181979635e-07,
259983: 8.815666841226464e-07,
637303: 4.933522528201219e-07,
68370: 5.19001048053502e-06,
1218786: 2.370730576774298e-06,
1770244: 6.551827370761489e-07,
294519: 7.502997036506805e-07,
1351298: 2.5109184274832154e-06,
222126: 1.7909646253144328e-06,
1387912: 5.247023389434492e-07,
336155: 3.5312654622603567e-07,
231810: 9.102015842130724e-07,
1794113: 1.8978338339303247e-06,
1593902: 2.2275713443118368e-06,
1499199: 8.612925761243842e-07,
1096475: 6.90310509552083e-07,
626646: 4.201433588272513e-07,
614513: 1.9752841973598824e-06,
409669: 4.859638923525929e-07,
775250: 1.0874075054994108e-06,
726427: 6.724139205612108e-07,
128572: 8.726450754931666e-07,
1283364: 1.6773307798291252e-06,
664879: 2.1860213628443504e-07,
1430336: 1.8439638796321369e-06,
300365: 3.952764435312316e-07,
34503: 2.533534765851714e-06,
437532: 1.4090032304610087e-06,
1658696: 4.584894225495387e-07,
1426508: 2.4291083637183615e-07,
1804087: 3.022959464554831e-07,
202280: 2.71864144618233e-06,
1219779: 1.0425532914966784e-06,
706355: 1.2447060006554393e-06,
156424: 1.881361791307902e-06,
1758591: 5.260971474726672e-07,
1582569: 4.042322096791342e-07,
1655182: 1.1202339213444023e-06,
1492413: 1.236011639198694e-06,
20010: 3.102515210295416e-07,
1235227: 1.920835543709066e-07,
160369: 1.8239234342641041e-06,
1071699: 4.492114865133545e-07,
800041: 1.6594842936762339e-06,
471874: 6.031154908075581e-07,
1256320: 5.181710229551157e-07,
212986: 1.5180126500473293e-06,
778666: 2.848985910682562e-07,
27317: 7.902008651699291e-06,
603112: 4.206849716058284e-06,
1072684: 2.7007632463784843e-06,
458008: 5.311597858379772e-07,
599127: 4.686397784585253e-07,
1831260: 6.706737538720624e-07,
1511622: 2.3020062517056226e-06,
1352161: 1.7533712739904472e-06,
1573105: 3.763179129280516e-07,
1150488: 5.195182552611053e-07,
1533080: 6.800254615798302e-07,
553980: 5.733880659211021e-07,
1420459: 4.042322096791342e-07,
6065: 4.837879554197195e-07,
233058: 1.6556497245737814e-07,
1622037: 4.042322096791342e-07,
139595: 3.7068555039147856e-07,
766895: 1.0132098727215367e-06,
567090: 1.376741210227281 9e-06,
351885: 1.0803938295402215e-06,
960433: 2.509565337098201e-06,
1371463: 1.3282861939224626e-06,
220723: 6.820653524962553e-07,
849050: 5.036768918548659e-07,

1243504: 9.54806485283262e-07,
1053312: 4.848599760152817e-07,
169495: 4.6454044518293876e-07,
1354251: 2.979037956242245e-07,
205195: 1.3601573019199571e-06,
79777: 1.8298771549854421e-06,
1613640: 1.7219461793576026e-07,
1313162: 1.6725791480500487e-06,
81598: 6.267024826135591e-07,
1353944: 2.5046837008320644e-07,
591299: 3.825395802231487e-07,
1552488: 8.621762172723408e-07,
1808831: 2.6103186734608056e-07,
1173705: 2.9443990880329947e-06,
1764008: 1.7850997686836787e-06,
713641: 1.1128472365525842e-06,
1733241: 7.880862144751073e-07,
674485: 7.454365713129734e-07,
1110877: 3.6207968134853146e-06,
1801859: 2.0364601209126604e-07,
535958: 2.861547344220549e-07,
992930: 3.485431876607245e-07,
901833: 7.007260948087101e-07,
1416532: 5.751119420638322e-07,
140165: 4.3491799732193146e-07,
1708748: 2.0895901558860652e-07,
41281: 2.4556611280124743e-06,
1139928: 2.3078403821089897e-06,
1107040: 9.119995288689036e-06,
835997: 4.544836741018556e-07,
805612: 4.445132018900788e-07,
1310090: 3.5730831670600226e-06,
1160801: 1.8362356330096707e-06,
71690: 4.042322096791342e-07,
829118: 5.748803309899641e-07,
1209271: 2.9857645704423495e-07,
1007826: 1.1202339213444023e-06,
1757704: 6.428994469008903e-07,
327433: 5.354991901511e-07,
1363519: 1.162916210329034e-06,
124045: 1.3755929412944144e-06,
578411: 5.072930621158016e-07,
939582: 6.731669067232698e-07,
1411821: 2.053428453276708e-07,
1336711: 1.2259294121140373e-06,
111141: 2.7694301649419764e-07,
974254: 6.465156171618259e-07,
215735: 5.189760737280554e-07,
1118480: 2.34804426394077e-06,
132008: 2.1329841990172935e-07,
316236: 1.1049669482751565e-06,
1677011: 4.191697745262301e-07,
389782: 1.6012562472115408e-06,
1415429: 2.1349412337612146e-06,
1013979: 9.514501678685972e-07,
1628559: 4.990897865018915e-07,
1221960: 1.6556497245737814e-07,
1610659: 1.0730687292267699e-06,
674707: 9.445029955912224e-07,
1326729: 6.623019223366488e-07,
711219: 2.6008911695991627e-06,
91841: 5.342001687074407e-07,
1104580: 6.428994469008903e-07,
1570516: 6.428994469008903e-07,
30447: 1.3321931582103904e-06,
792190: 3.9883378407530885e-07,
5411: 1.5817748059600964e-06,
815927: 1.5625526130914836e-06,
367871: 4.6384318687077223e-07,
1214389: 1.1562011445857285e-06,
1665942: 1.4980536490062022e-06,
1757866: 1.0037376055536431e-06,
1031590: 8.815666841226464e-07,
1656574: 4.042322096791342e-07,
545029: 5.977355018034399e-07,
1119764: 2.3148185471861252e-06,
1377733: 5.942343349917684e-07,

375423: 1.1433875403428527e-06,
1567186: 2.665219417066131e-06,
1005023: 1.0334337684032637e-06,
1580187: 4.837879554197195e-07,
612293: 8.02010938382061e-07,
1740713: 5.122007217556429e-07,
1136962: 4.485561251631746e-07,
491547: 8.545543799536566e-07,
135396: 1.9242286016661525e-06,
1536027: 1.407771116663947e-06,
1500498: 5.931322435503887e-07,
538664: 2.1238486109896667e-07,
52205: 2.490053320890804e-07,
1610529: 3.776135664263393e-07,
235333: 1.0845231338603522e-06,
648114: 4.042322096791342e-07,
1545191: 4.042322096791342e-07,
1245581: 3.863321668875025e-07,
742541: 8.324169904711276e-07,
1587544: 2.054498949746397e-06,
175227: 5.048893232604555e-07,
399979: 1.9364347095405532e-07,
1071433: 2.805948529760495e-06,
1675985: 4.531363771276947e-07,
918120: 2.6653066295205816e-07,
1487059: 2.457476115039869e-07,
177707: 3.013584005318256e-07,
831251: 2.020810623355921e-06,
1136007: 2.7656417960971867e-07,
197515: 7.415573140088579e-07,
805550: 4.555646075260357e-07,
1368835: 1.6556497245737814e-07,
1397351: 2.0483842883178986e-06,
1210429: 4.803784234594087e-07,
1684048: 5.052094592511672e-07,
993658: 4.6457124616526645e-07,
643063: 6.082925800680327e-07,
1503975: 2.451207181979635e-07,
493782: 8.707698329149955e-07,
1432233: 1.8850387375396097e-06,
311527: 2.6670319813754473e-06,
937184: 8.920116127383585e-07,
259011: 7.425534111488368e-07,
1440399: 6.929317454396935e-07,
525049: 5.205424185388819e-07,
1116286: 3.455847048942508e-07,
918841: 1.419587272392153e-06,
231222: 2.264384410230174e-06,
270635: 5.317093619962777e-07,
695719: 3.407072457870277e-07,
1789385: 2.635788732016461e-06,
900537: 3.8031303161910764e-07,
1562772: 5.556456296993328e-07,
625125: 6.269882977527732e-07,
643419: 4.169749048843362e-07,
312300: 4.561500827111396e-07,
591658: 8.184903428568966e-07,
660809: 5.416789802773919e-07,
169657: 3.9861225723072593e-07,
281745: 8.947890493067474e-06,
1513924: 6.181517095890449e-06,
122637: 2.28983586193646e-06,
7211: 1.6389419334054957e-06,
733917: 2.231930758847963e-06,
335868: 9.295067698676225e-07,
376990: 3.292225065522966e-07,
417088: 6.034429322153626e-07,
1535778: 6.694180288144188e-07,
827566: 4.042322096791342e-07,
1778965: 1.627460299197613e-06,
1602457: 2.4620379557048018e-06,
367827: 2.583546380095427e-06,
1705661: 2.0895901558860652e-07,
871552: 2.2938682143073953e-06,
1464896: 2.250132220217716e-07,
267343: 2.848985910682562e-07,
1183878: 9.213445569929391e-07,

1684743: 3.6292860003069285e-06,
108325: 2.4921382471933393e-06,
1150809: 4.27662664151621e-07,
1440757: 7.419850764769781e-07,
1015566: 1.6939100349042436e-06,
1720226: 9.911027459943496e-07,
1163878: 1.698852580581098e-06,
1629981: 1.0779692983366879e-06,
1156541: 1.1386520207412659e-06,
1827687: 2.106911307556093e-07,
349107: 2.101710391239334e-06,
1772442: 2.9767412935920197e-06,
374687: 5.412977154210629e-07,
1482135: 5.840442981353953e-07,
1506186: 6.424635398975157e-07,
1064116: 1.0159485055133254e-06,
307571: 1.1224252327350161e-06,
1455540: 1.6133346955390251e-06,
1144943: 1.0303869468452595e-06,
1077332: 6.002220147993931e-07,
854944: 2.3375561166359416e-07,
470394: 2.883600103088727e-06,
1059200: 1.0851968079480617e-06,
1664635: 3.8251087772521125e-07,
1175730: 3.3793575489531305e-07,
689828: 7.889927254426925e-07,
984700: 4.060048045444417e-07,
1048062: 2.5501744108796754e-06,
1568074: 1.4146533200653146e-06,
711175: 4.874041256806553e-07,
1637605: 8.855248763984095e-07,
230395: 1.521803942157203e-06,
680861: 4.1637220984084694e-07,
56785: 1.1648242914792487e-06,
1664821: 7.422819153947133e-07,
922569: 7.451854057102143e-07,
360739: 5.79254850308422e-07,
990102: 5.307777768743219e-07,
38688: 7.593942154456565e-07,
353776: 4.069390603214598e-07,
992126: 7.20466298997961e-07,
1128784: 4.042322096791342e-07,
84132: 1.367805168468623e-06,
1218973: 2.723022646593302e-07,
1018891: 2.3398361082050507e-06,
1770392: 1.920835543709066e-07,
1435169: 1.4942406355462737e-06,
617793: 2.5532017278008984e-07,
129579: 1.3689891424416126e-06,
1742982: 7.524353667269785e-07,
897335: 1.2364552492427917e-06,
942803: 7.903315640570298e-07,
760492: 7.799121201207873e-07,
849609: 1.657328618929199e-06,
1816176: 2.17049430409683e-07,
1009171: 1.1899800391144326e-06,
7821: 2.601468919798821e-06,
1573412: 2.8024560094698617e-06,
280551: 1.8631864525926998e-07,
379305: 2.2826235964040174e-06,
1014803: 1.8521467821577877e-06,
1636728: 2.451207181979635e-07,
1457449: 3.282926341994846e-07,
183522: 6.428994469008903e-07,
373526: 6.253007516310033e-07,
311931: 1.0207600978698647e-06,
228163: 5.73493718806689e-07,
1484017: 2.8465419563486694e-07,
884709: 1.160011794214695e-06,
963385: 6.428994469008903e-07,
18517: 9.411045540654395e-07,
1166989: 2.080211804479281e-06,
767438: 1.9072556132024269e-06,
1689705: 6.784240147499872e-07,
1255532: 1.1483600742157898e-06,
1398871: 1.3044682799015475e-06,
644613: 2.13890663333168e-06,

1689027: 1.3093720001980013e-06,
1167599: 2.3515682389211672e-06,
793422: 9.336560168617612e-07,
1285415: 6.769947665039983e-07,
1359864: 2.6103186734608056e-07,
72624: 3.1191564256662536e-07,
363188: 8.262049528657345e-07,
1135381: 6.113831322421199e-07,
1128500: 1.8270760230031361e-06,
579094: 9.705642106764e-07,
110619: 1.5524868065349163e-06,
1499870: 5.328090519222583e-07,
1706396: 4.144316642612605e-07,
1531486: 1.6556497245737814e-07,
1476606: 5.172375303333749e-07,
1708719: 1.141517965784565e-06,
1722495: 1.423249263783422e-06,
1409029: 1.4903400407278685e-06,
1406493: 4.8517617638211215e-06,
1143011: 1.3040741957121205e-06,
416792: 2.0550139337962807e-06,
811702: 1.7111829286923374e-07,
1594917: 1.6442558125609009e-06,
1555548: 1.6556497245737814e-07,
461075: 7.683849470649877e-07,
898978: 1.1416554251758502e-06,
1668491: 4.447386150195787e-07,
648133: 6.288426948291202e-07,
760360: 1.3300459919680655e-06,
15671: 1.973872707536123e-07,
1177344: 1.5931885373376287e-06,
933781: 1.7992307446987706e-06,
125939: 1.677748542835055e-07,
621529: 1.9863870355927323e-06,
1305021: 1.973872707536123e-07,
1963: 9.93381045437264e-07,
1304862: 4.4418640353439926e-07,
424313: 1.3944826844083762e-06,
702888: 7.846567197858298e-07,
1363018: 2.7365258052181943e-06,
752401: 2.661762882965794e-06,
703123: 5.657675958581607e-07,
1617399: 7.447873524620298e-07,
242790: 1.8021784965708087e-06,
1124220: 3.0861138940858584e-06,
731869: 1.2053273709551661e-06,
804351: 3.4058761308666595e-07,
708341: 8.87249237389831e-07,
1771379: 1.2968563417072689e-06,
1796152: 5.889940048298614e-07,
40442: 1.953501076502341e-06,
604072: 8.179550789055839e-07,
503899: 4.2514128055760247e-07,
8250: 2.1329841990172935e-07,
1140863: 2.4668063478111224e-07,
127405: 6.629486604794701e-07,
537931: 8.046683145022971e-07,
623878: 4.4035867780902565e-07,
481358: 7.980948289027894e-07,
1811099: 1.0277394389339195e-06,
1756739: 7.866652589953354e-07,
885869: 1.0618930411346397e-06,
713457: 5.324857149336536e-07,
1445857: 5.845585666911277e-07,
1277246: 2.9270117382358283e-07,
1722634: 8.050864663068706e-07,
647399: 3.6341909578985373e-07,
753961: 4.042322096791342e-07,
1843752: 6.428994469008903e-07,
1330719: 1.4437233741894308e-06,
540860: 1.6925920141696547e-06,
1539379: 3.686661115833431e-07,
1371321: 1.0247670264557e-06,
1632610: 7.648849237031211e-07,
1653720: 2.848985910682562e-07,
1206163: 3.8322398244893636e-07,
1815486: 2.404085630753835e-06,

629075: 2.8774768582230785e-06,
397167: 4.655928460266432e-06,
401549: 3.301404574646879e-07,
896709: 1.7136178667109762e-06,
785326: 1.1757334035892338e-06,
1006657: 3.8585862078190375e-07,
1284381: 1.9165078745969877e-06,
1017398: 2.5648582473233286e-07,
640756: 2.4505402301609534e-07,
1813874: 1.2687503477257234e-06,
443939: 2.6103186734608056e-07,
535546: 1.277175710669011 6e-06,
1480505: 1.3824276533887997e-06,
438127: 1.4402351396290269e-06,
521886: 2.7598777725322644e-07,
292052: 1.981995815351555e-06,
1410093: 1.8869963325603323e-06,
992569: 1.8064788117000207e-06,
1715317: 7.993590801907083e-07,
879385: 2.759984717495023e-06,
1314093: 9.51083252424539e-07,
789938: 6.310972208844298e-07,
322321: 1.498976454896918e-06,
1062432: 7.531118389854668e-07,
1691811: 5.0395557585854e-07,
651378: 7.903914535610229e-07,
211414: 1.6556497245737814e-07,
1097610: 3.07572478303882e-06,
1533744: 6.428994469008903e-07,
1127761: 1.1202339213444023e-06,
203054: 2.3375561166359416e-07,
849904: 6.383534042871425e-07,
627341: 1.5769098641433899e-06,
1448767: 1.738987476231447e-06,
1295758: 8.147124437188127e-07,
734119: 3.060436708572013e-07,
1564484: 5.083910997897043e-07,
1811810: 7.821220019469146e-07,
782699: 1.0491728642057882e-06,
1021154: 1.4733544667240429e-06,
602034: 1.2288903214479916e-06,
1200986: 7.796304208989952e-07,
634838: 1.4755736870566132e-06,
1292151: 7.023476964652837e-07,
1326077: 3.5877178354165686e-07,
1842965: 9.363991053314144e-07,
1206110: 1.6556497245737814e-07,
1334999: 7.10239964079867e-07,
148065: 1.1202339213444023e-06,
12244: 5.748836554875105e-07,
1487275: 1.1321672832054901e-06,
1248963: 1.1158945170312798e-06,
319610: 4.077094519675319e-06,
967102: 5.48366607326551e-07,
664304: 2.848985910682562e-07,
206540: 1.3589011585661585e-06,
1600366: 1.967046287234423e-06,
1118409: 3.0991725319029543e-07,
485658: 2.163222582013715e-07,
1707361: 3.0080974021637326e-07,
582634: 3.75861617800876e-06,
1048523: 3.807158923774155e-07,
363171: 3.685622993816848e-07,
451817: 9.427021104834355e-07,
1390733: 4.90956714926014e-07,
542070: 2.0678235061259096e-06,
475612: 6.144877206990074e-07,
1513618: 3.9123306641981684e-07,
287011: 1.5532238164738118e-06,
509247: 6.132365188461864e-07,
412537: 5.180230597556032e-07,
1359237: 2.6103186734608056e-07,
1654384: 1.077046516513799e-06,
516912: 8.672739712967593e-07,
1087783: 1.4974000380251655e-06,
10553: 4.1872272051045514e-07,
280719: 5.541253369713165e-07,

1478518: 7.021176335799726e-07,
89511: 1.424310337488963e-06,
931446: 2.2523178176281716e-07,
980701: 2.2612819596915932e-06,
846172: 3.945238995209232e-06,
1173091: 8.02010938382061e-07,
681801: 4.837879554197196e-07,
617497: 1.3233350604703674e-06,
991515: 3.0442591047730896e-07,
1594774: 5.051931996971268e-07,
2493: 5.604230918933292e-07,
346372: 3.940464740365464e-07,
559481: 3.56498762234783e-07,
182341: 5.089756056780467e-07,
601785: 7.11303839493239e-07,
255616: 1.53133406495473e-06,
799585: 2.702435852739378e-07,
1728920: 4.6950289889874325e-06,
499883: 8.60555393791269e-07,
1417605: 4.4477675692104433e-07,
1230379: 5.524609428165883e-07,
756045: 8.938600314847837e-07,
1766188: 5.09927700448769e-07,
111238: 4.0356786343178366e-07,
98644: 7.73212478385469e-07,
923375: 2.0895901558860652e-07,
415223: 1.988355861986366e-06,
542584: 2.4276540748463086e-07,
1431345: 7.66591070030576e-07,
1554188: 2.217646886317267e-06,
555799: 9.066895511986207e-07,
697117: 2.0106276450240393e-06,
573724: 2.0215570524866703e-06,
1270943: 9.91427005468669e-07,
1097752: 2.1860213628443504e-07,
1384386: 6.655014670062616e-07,
639133: 1.5876834023119957e-06,
911225: 7.790201432606186e-07,
175345: 5.295175465186745e-07,
1083846: 1.6556497245737814e-07,
1141252: 1.6771241415284999e-06,
1859230: 2.1329841990172935e-07,
521884: 4.042322096791342e-07,
1662727: 3.246764639385488e-07,
537157: 3.246764639385488e-07,
1275484: 1.5869852796692071e-06,
701935: 3.3172963730511257e-07,
1774821: 1.0889798783748868e-06,
1241870: 2.7826894558987406e-07,
129970: 1.7627125776567735e-06,
1593220: 9.599357790339078e-07,
1211768: 2.2620188217267624e-06,
689907: 5.233846896177373e-07,
343678: 5.905966976609451e-07,
1832188: 6.646485707262414e-07,
836552: 3.193225286342606e-07,
504642: 1.2855125799448252e-06,
721519: 9.40034993934405e-07,
1787289: 5.835828068960415e-07,
1803738: 1.2956062575593633e-06,
1087393: 7.837228165245231e-07,
1197429: 1.3321420266948496e-06,
733972: 4.215106828670391e-06,
949699: 1.8316207109623816e-06,
495204: 6.74517756105482e-07,
1073320: 6.843820857513384e-07,
1053296: 5.122007217556429e-07,
17271: 1.2787168381139094e-06,
360050: 4.60988384908492e-07,
1392993: 7.44995987267975e-07,
344557: 6.915972064148244e-07,
741161: 8.648880295231166e-07,
1644408: 1.0823603989847395e-06,
1430171: 4.460927791057253e-07,
1024577: 2.2176159906006297e-06,
1832376: 3.628995209185708e-06,
1394586: 1.9687400787786658e-07

```
 842187: 1.102358585306106e-06,
 555555: 8.340672241657674e-07,
 751546: 2.5010943652702766e-06,
 108673: 2.735334845338868e-07,
 906283: 3.98274138677783e-07,
 1065400: 3.4194707559077715e-07,
 1535940: 1.1397612407534554e-06,
 1533415: 2.2423899056236712e-06,
 1122776: 2.9560085840959883e-06,
 959167: 3.437698429162893e-07,
 728944: 7.677554618565443e-07,
 1534498: 7.374821551840873e-07,
 304984: 5.480008073389063e-07,
 1641628: 1.075123190792598e-06,
 1266290: 1.2442738597124617e-06,
 1773719: 8.467033887933341e-07,
 1601865: 4.28383061064669e-07,
 1499731: 6.208445517566577e-07,
 496945: 2.4155075628208767e-06,
 81393: 2.0647935598110775e-07,
 568199: 1.4959824716476666e-06,
 77972: 8.721347897065644e-07,
 942866: 3.0460741828707085e-06,
 312503: 1.9677894525041907e-06,
 1240018: 2.7374012283451177e-07,
 893254: 1.906678956947926e-06,
 1210368: 5.440110986114564e-07,
 25555: 2.848985910682562e-07,
 982702: 4.710553895429796e-07,
 1133519: 3.4979933101452315e-07,
 281441: 5.139227075941838e-07,
 1557796: 1.3625173288270945e-06,
 1585307: 2.3567575122837858e-06,
 290047: 2.751393234733585e-07,
 59507: 8.646309534005976e-07,
 1635312: 1.766544145029e-06,
 1344467: 3.67275978471361e-07,
 917597: 9.541476503404468e-07,
 1785341: 5.506871052470301e-07,
 1843022: 2.272450605028098e-06,
 1242208: 5.235658282900123e-07,
 394556: 2.3330219244062385e-06,
 1743132: 1.6556497245737814e-07,
 1777224: 1.040678175603817e-06,
 1354482: 6.324986524361039e-07,
 948669: 8.149086123127317e-07,
 1330050: 4.0592462767840465e-06,
 879593: 3.253409688922204e-07,
 387658: 1.1194432052103223e-06,
 759275: 4.906929228211694e-07,
 1791073: 7.022046391802357e-07,
 931027: 1.6221153039163137e-06,
 812080: 2.927581910747146e-07,
 9285: 7.013156027244751e-07,
 1535925: 2.6216038116125805e-07,
 686248: 3.832290471131673e-07,
 182631: 3.246764639385488e-07,
 1379691: 7.502997036506806e-07,
 1128159: 2.7479571461885817e-06,
 1476900: 3.5107520586040114e-06,
 730964: 1.5895566200599707e-06,
 622268: 4.7674790331648517e-07,
 902595: 6.636285476116692e-07,
 667845: 4.827764363435862e-07,
 1262789: 6.747834988623608e-07,
 539098: 9.712631379603083e-07,
 1334753: 1.2318941152934374e-06,
 1725849: 8.434553382625463e-07,
 ...}
```

In [27]:

```
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 [28]:

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

```
5.615699699389075e-07
```

4. Other Graph Features

4.1 Shortest path:

Getting Shortest path between twoo nodes, if nodes have direct path i.e directly connected then we are removing that edge and calculating path.

In [29]:

```
#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 [30]:

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

Out[30]:

```
10
```

In [31]:

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

Out[31]:

```
-1
```

4.2 Checking for same community

In [32]:

```
#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 [33]:

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

Out[33]:

0

In [34]:

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

Out[34]:

0

4.3 Adamic/Adar Index:

Adamic/Adar measures is defined as inverted sum of degrees of common neighbours for given two vertices. $$A(x,y)=\sum_{u \in N(x) \cap N(y)}\frac{1}{log(|N(u)|)}$$

In [35]:

```
#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 [36]:

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

Out[36]:

0

In [37]:

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

Out[37]:

```
0
```

4.4 Is persion was following back:

In [38]:

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

In [39]:

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

Out[39]:

```
1
```

In [40]:

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

Out[40]:

```
0
```

4.5 Katz Centrality:

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

https://www.geeksforgeeks.org/katz-centrality-centrality-measure/ Katz centrality computes the centrality for a node based on the centrality of its neighbors. It is a generalization of the eigenvector centrality. The Katz centrality for node `i` is

$$x_i = \alpha \sum_{j} A_{ij} x_j + \beta,$$ where `A` is the adjacency matrix of the graph G with eigenvalues $$\lambda$$.

The parameter $$\beta$$ controls the initial centrality and

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

In [41]:

```
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 [42]:

```
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 [43]:

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

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

```
0.0007483800935562018
```

4.6 Hits Score

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

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

In [44]:

```
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 [45]:

```
print('min',hits[0][min(hits[0], key=hits[0].get)])
print('max',hits[0][max(hits[0], key=hits[0].get)])
print('mean',float(sum(hits[0].values())) / len(hits[0]))
```

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

5. Featurization

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

In [46]:

```
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 [47]:

```
len(skip_train)
```

Out[47]:

```
15000028
```

In [48]:

```
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 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 [49]:

```
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 [50]:

```
df_final_train = pd.read_csv('data/after_eda/train_after_eda.csv', skiprows=skip_train, names=['sou
rce_node', 'destination_node'])
df_final_train['indicator_link'] = pd.read_csv('data/train_y.csv', skiprows=skip_train, names=['ind
icator_link'])
print("Our train matrix size ",df_final_train.shape)
df_final_train.head(2)
```

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

Out[50]:

	source_node	destination_node	indicator_link
0	273084	1505602	1
1	1831478	561277	1

In [51]:

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

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

Out[51]:

	source_node	destination_node	indicator_link
0	848424	784690	1
1	1153009	338609	1

5.2 Adding a set of features

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

1. jaccard_followers
2. jaccard_followees
3. cosine_followers
4. cosine_followees
5. num_followers_s
6. num_followees_s
7. num_followers_d
8. num_followees_d
9. inter_followers
10. inter_followees

In [52]:

```
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'],row['destination_node']),axis=1)
    df_final_test['jaccard_followers'] = df_final_test.apply(lambda row:

jaccard_for_followers(row['source_node'],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_for_followees(row['source_node'],row['destination_node']),axis=1)
    df_final_test['jaccard_followees'] = df_final_test.apply(lambda row:

jaccard_for_followees(row['source_node'],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['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 [53]:

```
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, int
er_followees
```

In [54]:

```
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_features_stage1(d
f_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('data/fea_sample/storage_sample_stage1.h5')
    hdf.put('train_df',df_final_train, format='table', data_columns=True)
    hdf.put('test_df',df_final_test, format='table', data_columns=True)
    hdf.close()
else:
    df_final_train = read_hdf('data/fea_sample/storage_sample_stage1.h5', 'train_df',mode='r')
    df_final_test = read_hdf('data/fea_sample/storage_sample_stage1.h5', 'test_df',mode='r')
```

5.3 Adding new set of features

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

1. adar index
2. is following back
3. belongs to same weakly connect components
4. shortest path between source and destination

In [55]:

```
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['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'],r
ow['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['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['destination_node']),axis=1)

    ##mapping same component of wcc or not on train
    df_final_test['same_comp'] = df_final_test.apply(lambda row: belongs_to_same_wcc(row['source_no
de'],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_path_length(r
ow['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 = read_hdf('data/fea_sample/storage_sample_stage2.h5', 'train_df',mode='r')
    df_final_test = read_hdf('data/fea_sample/storage_sample_stage2.h5', 'test_df',mode='r')
```

5.4 Adding new set of features

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

1. Weight Features
    - weight of incoming edges
    - weight of outgoing edges
    - weight of incoming edges + weight of outgoing edges
    - weight of incoming edges * weight of outgoing edges
    - 2*weight of incoming edges + weight of outgoing edges
    - weight of incoming edges + 2*weight of outgoing edges
2. Page Ranking of source
3. Page Ranking of dest
4. katz of source
5. katz of dest
6. hubs of source
7. hubs of dest
8. authorities_s of source
9. authorities_s of dest

Weight Features

In order to determine the similarity of nodes, an edge weight value was calculated between nodes. Edge weight decreases as the neighbor count goes up. Intuitively, consider one million people following a celebrity on a social network then chances are most of them never met each other or the celebrity. On the other hand, if a user has 30 contacts in his/her social network, the chances are higher that many of them know each other. `credit` - Graph-based Features for Supervised Link Prediction William Cukierski, Benjamin Hamner, Bo Yang

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

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

In [56]:

```
#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
[00:15<00:00, 112202.40it/s]
```

In [57]:

```
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: Weight_in.get(x,m
ean_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,mea
n_weight_in))
    df_final_test['weight_out'] = df_final_test.source_node.apply(lambda x: Weight_out.get(x,mean_w
```

```
eight_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 [58]:

```
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.get(x,mean_pr
))

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

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

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

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

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

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

5.5 Adding new set of features

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

1. SVD features for both source and destination

In [59]:

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

In [60]:

```
#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 [61]:

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

In [62]:

```
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 [63]:

```
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('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 [64]:

```
# prepared and stored the data from machine learning models
# pelase check the FB_Models.ipynb
```

3rd Notebook:

Social network Graph Link Prediction - Facebook Challenge

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

In [2]:

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

In [3]:

```
df_final_train.columns
```

Out[3]:

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

In [4]:

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

In [5]:

```
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)
```

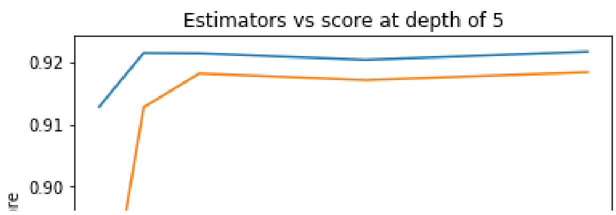
In [6]:

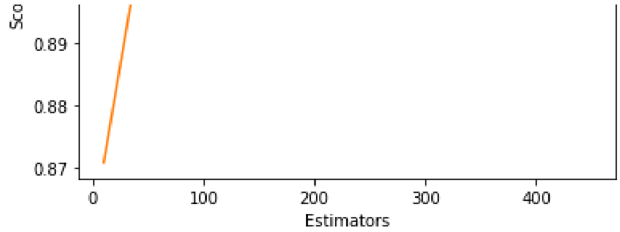
```
estimators = [10,50,100,250,450]
train_scores = []
test_scores = []
for i in estimators:
    clf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=5, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            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 0.9128303959052149 test Score 0.8707937877480586
Estimators =  50 Train Score 0.9214447615849564 test Score 0.912747695242132
Estimators =  100 Train Score 0.9213881316837911 test Score 0.9181440647595758
Estimators =  250 Train Score 0.9203382194235182 test Score 0.9170826225226933
Estimators =  450 Train Score 0.9216463286786598 test Score 0.9183548706705736
```

Out[6]:

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

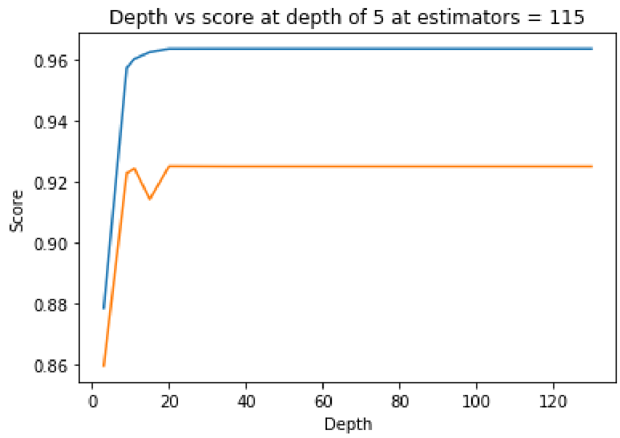




In [7]:

```
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,war
m_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 0.8786209385515186 test Score 0.8596559642965466
depth =  9 Train Score 0.9574483286552693 test Score 0.9228689701782546
depth =  11 Train Score 0.9603514373747953 test Score 0.9244026800387679
depth =  15 Train Score 0.9625799898425597 test Score 0.9143051800123485
depth =  20 Train Score 0.9636710892577768 test Score 0.9251697810773188
depth =  35 Train Score 0.963683290655611 test Score 0.92511552338953
depth =  50 Train Score 0.963683290655611 test Score 0.92511552338953
depth =  70 Train Score 0.963683290655611 test Score 0.92511552338953
depth =  130 Train Score 0.963683290655611 test Score 0.92511552338953
```



In [8]:

```
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 [0.96204957 0.96128263 0.95981394 0.96123824 0.96335889]
mean train scores [0.96303618 0.96221892 0.96039219 0.96181684 0.96448908]
```

In [9]:

```
sp_randint(105,125)
```

Out[9]:

```
<scipy.stats._distn_infrastructure.rv_frozen at 0x16074934eb8>
```

In [10]:

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

```
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 [11]:

```
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 [12]:

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

In [13]:

```
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 0.9653868439510842
Test f1 score 0.9263349027722062
```

In [14]:

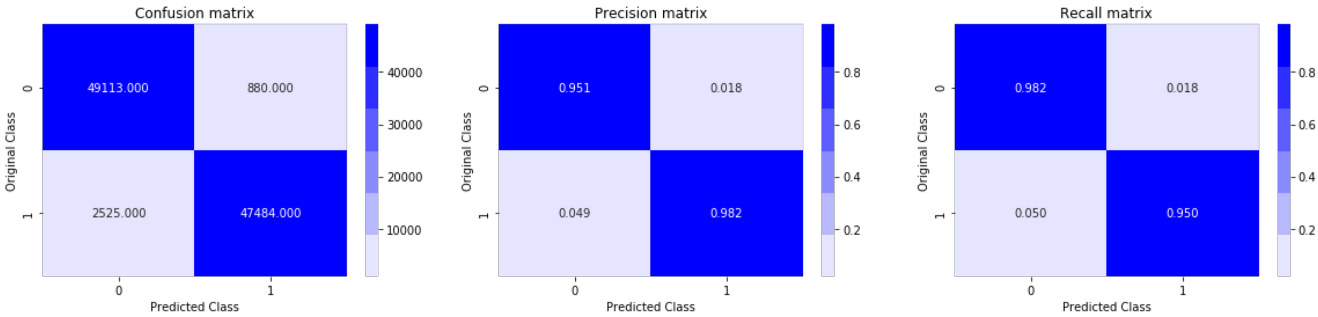
```
from sklearn.metrics import confusion_matrix
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)

    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()
```

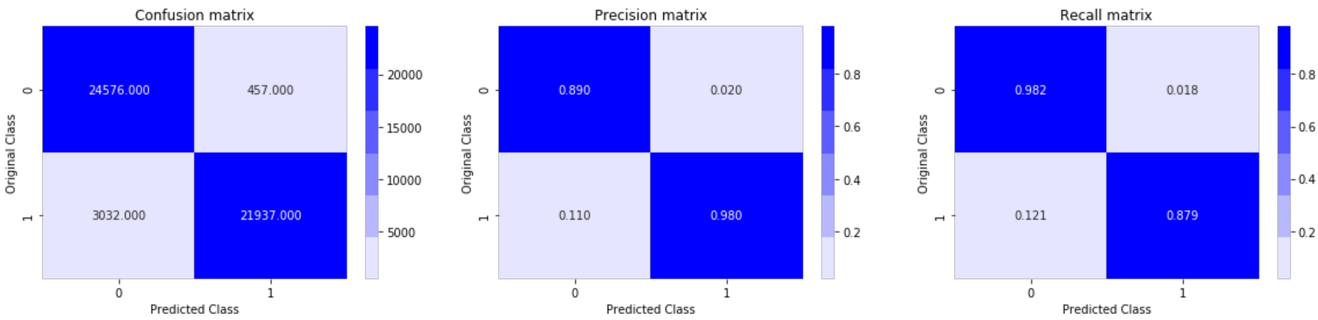
In [15]:

```
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

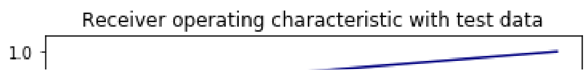


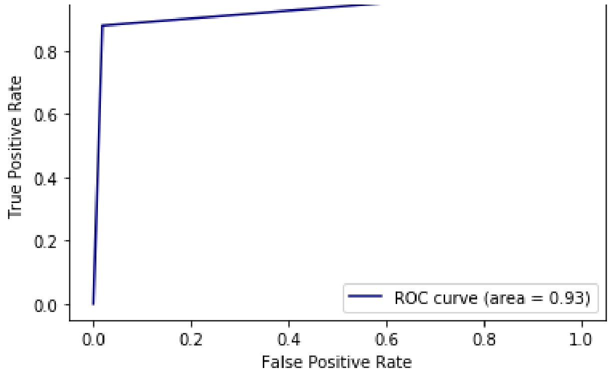
Test confusion_matrix



In [16]:

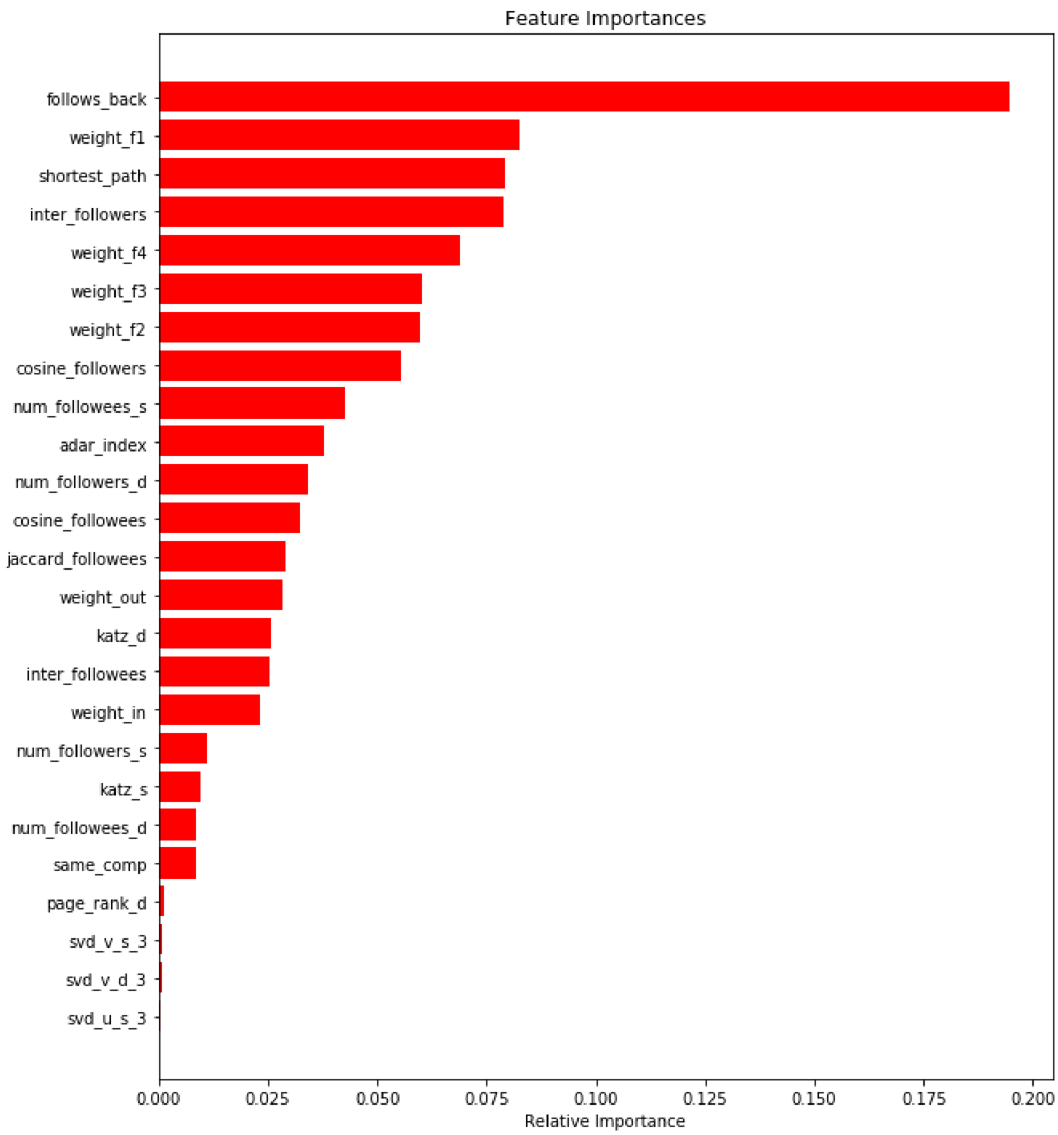
```
from sklearn.metrics import roc_curve, auc
fpr,tpr,ths = roc_curve(y_test,y_test_pred)
auc_sc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='navy',label='ROC curve (area = %0.2f)' % auc_sc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic with test data')
plt.legend()
plt.show()
```





In [17]:

```
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()
```



Assignments:

1. Add another feature called Preferential Attachment with followers and followees data of vertex. you can check about Preferential Attachment in below link http://be.amazd.com/link-prediction/
2. Add feature called svd_dot. you can calculate svd_dot as Dot product between sourse node svd and destination node svd

2. Add feature called svd_dot. you can calculate svd_dot as Dot product between sourse node svd and destination node svd features. you can read about this in below pdf https://storage.googleapis.com/kaggle-forum-message-attachments/2594/supervised_link_prediction.pdf
3. Tune hyperparameters for XG boost with all these features and check the error metric.

In [6]:

```
df_final_train.columns
```

Out[6]:

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

In [7]:

```
num_followers_s = list(df_final_train['num_followers_s'])
num_followers_d = list(df_final_train['num_followers_d'])
num_followees_s = list(df_final_train['num_followees_s'])
num_followees_d = list(df_final_train['num_followees_d'])

preferential_followers_train = []
for i in range(df_final_train.shape[0]):
    res = num_followers_s[i] * num_followers_d[i]
    preferential_followers_train.append(res)

preferential_followees_train = []
for i in range(df_final_train.shape[0]):
    res = num_followees_s[i] * num_followees_d[i]
    preferential_followees_train.append(res)

num_followers_s = list(df_final_test['num_followers_s'])
num_followers_d = list(df_final_test['num_followers_d'])
num_followees_s = list(df_final_test['num_followees_s'])
num_followees_d = list(df_final_test['num_followees_d'])


preferential_followers_test = []
for i in range(df_final_test.shape[0]):
    res = num_followers_s[i] * num_followers_d[i]
    preferential_followers_test.append(res)

preferential_followees_test = []
for i in range(df_final_test.shape[0]):
    res = num_followees_s[i] * num_followees_d[i]
    preferential_followees_test.append(res)

print("preferential_followers_train ",len(preferential_followers_train))
print("preferential_followees_train ",len(preferential_followees_train))
print("preferential_followers_test ",len(preferential_followers_test))
print("preferential_followees_test ",len(preferential_followees_test))
```

```
preferential_followers_train  100002
preferential_followees_train  100002
preferential_followers_test  50002
preferential_followees_test  50002
```

In [46]:

```
ss = df_final_train[['svd_u_s_1','svd_u_s_2','svd_u_s_3','svd_u_s_4','svd_u_s_5']].values
dd = df_final_train[['svd_u_d_1','svd_u_d_2','svd_u_d_3','svd_u_d_4','svd_u_d_5']].values
```

In [50]:

```
np.dot(ss[0],dd[0])
```

Out[50]:

```
1.1149274107932878e-11
```

In [8]:

```
ss = df_final_train[['svd_u_s_1','svd_u_s_2','svd_u_s_3','svd_u_s_4','svd_u_s_5']].values
dd = df_final_train[['svd_u_d_1','svd_u_d_2','svd_u_d_3','svd_u_d_4','svd_u_d_5']].values
svd_u_dot_train = []
for i in range(df_final_train.shape[0]):
    res = np.dot(ss[i],dd[i])
    svd_u_dot_train.append(res)

ss = df_final_test[['svd_u_s_1','svd_u_s_2','svd_u_s_3','svd_u_s_4','svd_u_s_5']].values
dd = df_final_test[['svd_u_d_1','svd_u_d_2','svd_u_d_3','svd_u_d_4','svd_u_d_5']].values
svd_u_dot_test = []
for i in range(df_final_test.shape[0]):
    res = np.dot(ss[i],dd[i])
    svd_u_dot_test.append(res)

print("svd_dot_train ",len(svd_u_dot_train))
print("svd_dot_test ",len(svd_u_dot_test))
```

```
svd_dot_train  100002
svd_dot_test  50002
```

In [9]:

```
ss = df_final_train[['svd_v_s_1','svd_v_s_2','svd_v_s_3','svd_v_s_4','svd_v_s_5']].values
dd = df_final_train[['svd_v_d_1','svd_v_d_2','svd_v_d_3','svd_v_d_4','svd_v_d_5']].values
svd_v_dot_train = []
for i in range(df_final_train.shape[0]):
    res = np.dot(ss[i],dd[i])
    svd_v_dot_train.append(res)

ss = df_final_test[['svd_v_s_1','svd_v_s_2','svd_v_s_3','svd_v_s_4','svd_v_s_5']].values
dd = df_final_test[['svd_v_s_1','svd_v_s_2','svd_v_s_3','svd_v_s_4','svd_v_s_5']].values
svd_v_dot_test = []
for i in range(df_final_test.shape[0]):
    res = np.dot(ss[i],dd[i])
    svd_v_dot_test.append(res)

print("svd_dot_train ",len(svd_v_dot_train))
print("svd_dot_test ",len(svd_v_dot_test))
```

```
svd_dot_train  100002
svd_dot_test  50002
```

In [10]:

```
#https://stackoverflow.com/a/51308247
dataset_train = pd.DataFrame({'preferential_followers_train': preferential_followers_train,
'preferential_followees_train': preferential_followees_train,'svd_u_dot_train':svd_u_dot_train,'sv
d_v_dot_train':svd_v_dot_train})
#https://stackoverflow.com/a/51308247
dataset_test = pd.DataFrame({'preferential_followers_test': preferential_followers_test,
'preferential_followees_test':
preferential_followees_test,'svd_u_dot_test':svd_u_dot_test,'svd_v_dot_test':svd_v_dot_test})
```

In [11]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((df_final_train,dataset_train))
X_te = hstack((df_final_test,dataset_test))

print("Final Data matrix on BOW")
print(X_tr.shape, y_train.shape)
```

```
# print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix on BOW
(100002, 56) (100002,)
(50002, 56) (50002,)
====================================================================================================
```

In [12]:

```
from sklearn.model_selection import GridSearchCV
import xgboost as xgb
import time

start_time = time.time()
gbdt = xgb.XGBClassifier(n_jobs=-1,class_weight='balanced')
parameters = {'n_estimators': [10, 100, 500], 'max_depth':[10, 50, 100, 500]}
clf = GridSearchCV(gbdt, parameters, cv= 3, scoring='f1',return_train_score=True)
clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
print("Execution time: " + str((time.time() - start_time)) + ' ms')
```

```
Execution time: 2740.791193962097 ms
```

In [13]:

```
print(clf.best_estimator_)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', class_weight='balanced',
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              gamma=0, learning_rate=0.1, max_delta_step=0, max_depth=10,
              min_child_weight=1, missing=None, n_estimators=500, n_jobs=-1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

With clf.best*estimator* we are overfitting the model with those params , so we'll manually check the params using heatmaps

In [14]:

```
train_auc = train_auc.reshape(3,4)
cv_auc = cv_auc.reshape(3,4)
train_auc
cv_auc
```

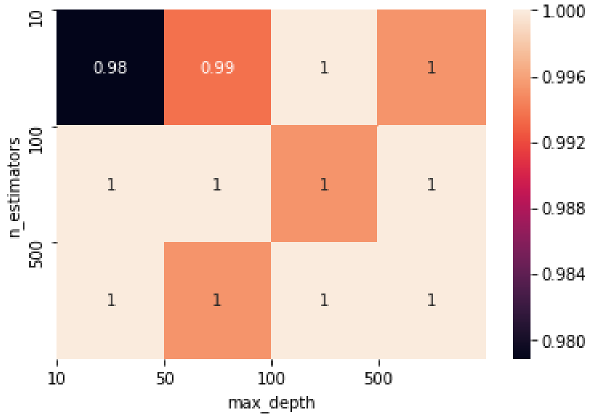
Out[14]:

```
array([[0.97389823, 0.98041685, 0.98268328, 0.97457532],
       [0.97933999, 0.98121594, 0.97457532, 0.97933999],
       [0.98121594, 0.97457532, 0.97933999, 0.98121594]])
```

In [15]:

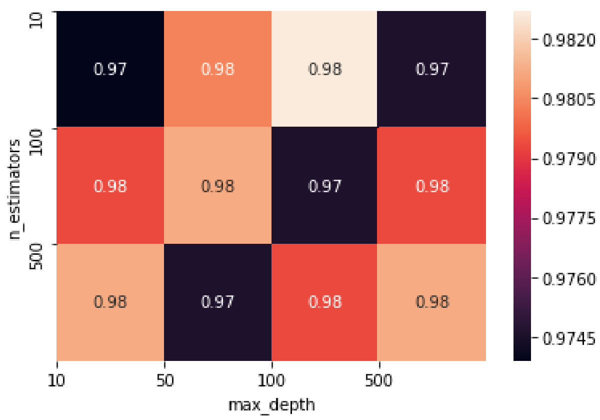
```
import matplotlib.pyplot as plt
# plt.show()
import numpy as np; np.random.seed(0)
import seaborn as sns

sns.heatmap(train_auc,annot=True)
plt.yticks(np.arange(3), [10, 100, 500])
plt.xticks(np.arange(4), [10, 50, 100, 500])
plt.xlabel('max_depth')
plt.ylabel('n_estimators')
```

```
plt.show()
```



In [16]:

```
import matplotlib.pyplot as plt
# plt.show()
import numpy as np; np.random.seed(0)
import seaborn as sns

sns.heatmap(cv_auc,annot=True)
plt.yticks(np.arange(3), [10, 100, 500])
plt.xticks(np.arange(4), [10, 50, 100, 500])
plt.xlabel('max_depth')
plt.ylabel('n_estimators')
plt.show()
```



In [17]:

```
clf = xgb.XGBClassifier(base_score=0.5, booster='gbtree', class_weight='balanced',
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              gamma=0, learning_rate=0.1, max_delta_step=0, max_depth=10,
              min_child_weight=1, missing=None, n_estimators=10, n_jobs=-1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

In [18]:

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

In [19]:

```
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 0.977334523761211
Test f1 score 0.933524912310358
```

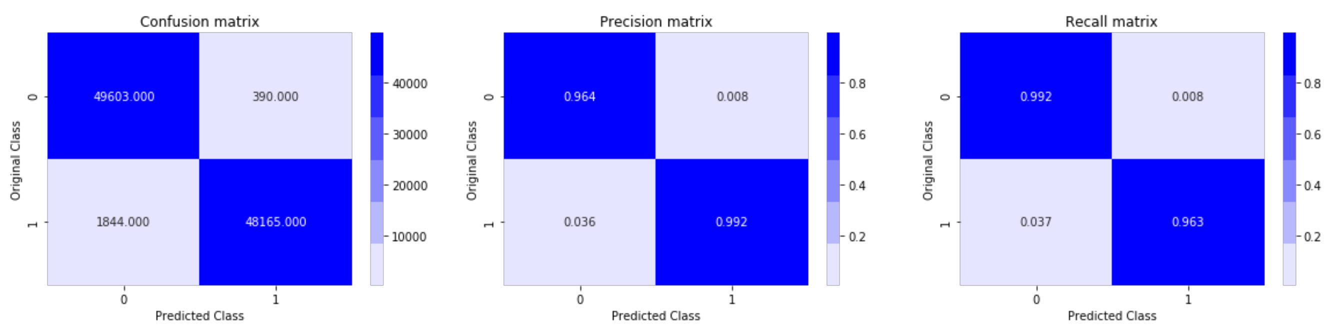
In [20]:

```python
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()
```

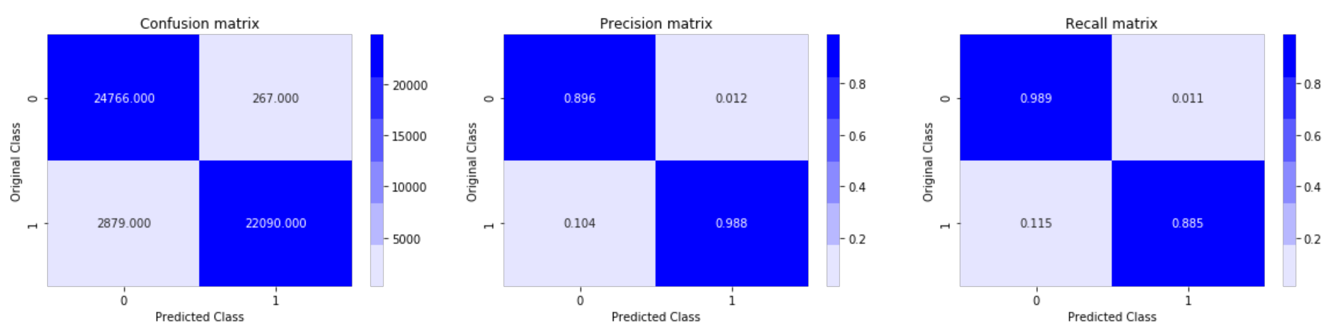
In [21]:

```python
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
```

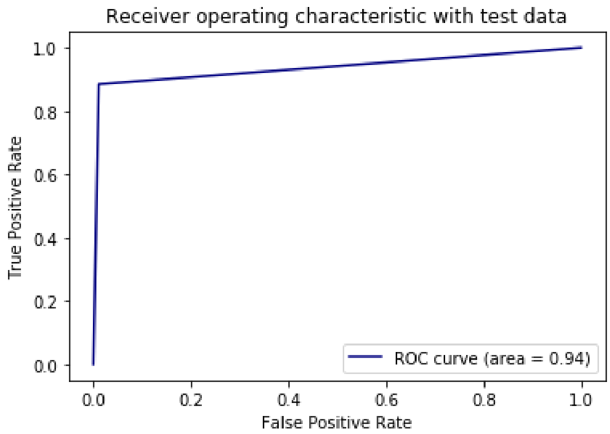



```
Test confusion_matrix
```



In [22]:

```
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 [23]:

```
names = df_final_train.columns
```

In [24]:

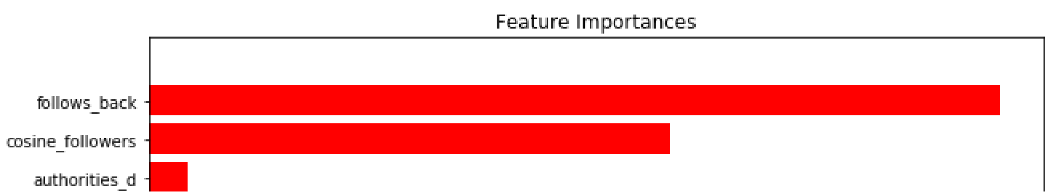
```
names.append(dataset_train.columns)
```

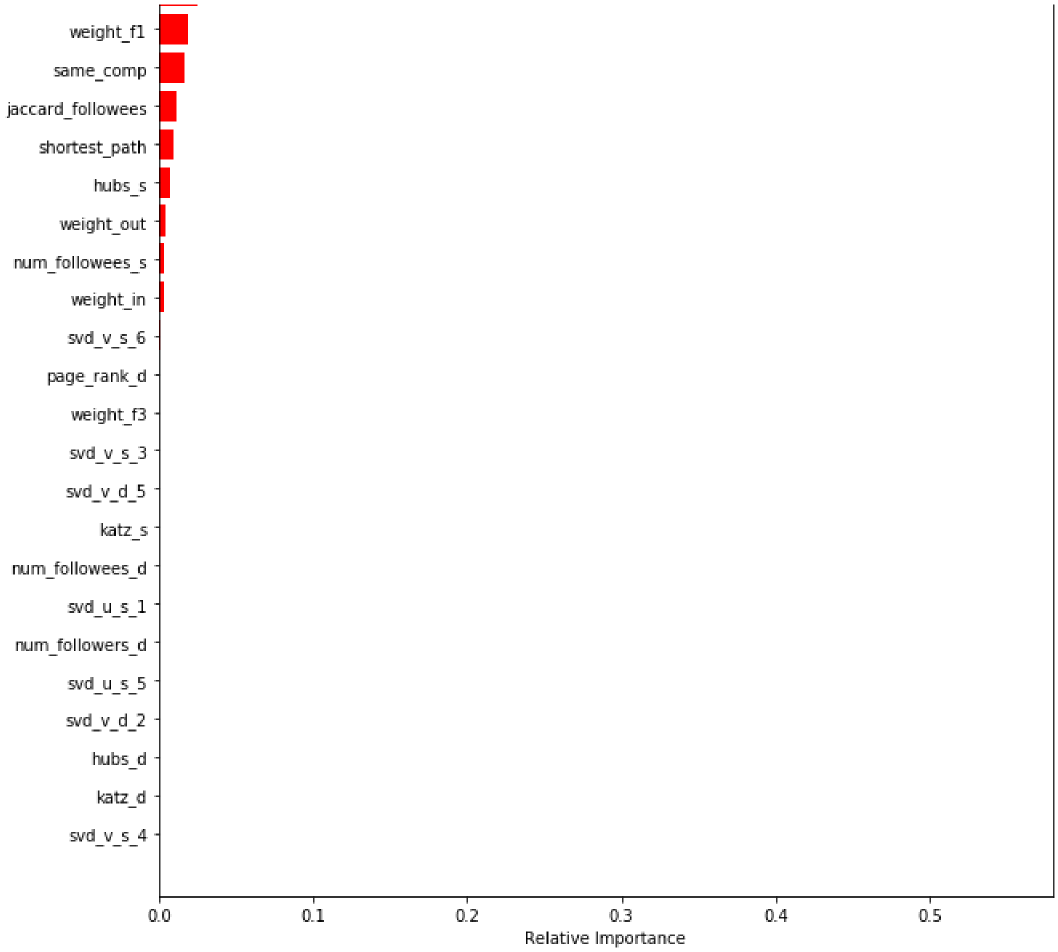
Out[24]:

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

In [25]:

```
# 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)), [names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```





Results(Pretty Table):

In [26]:

```
from prettytable import PrettyTable
x = PrettyTable()


x.field_names = [ "Model", "Hyperparameters(max_depth,n_estimators)" , "Test F1"]
x.add_row([ "RF","(14,121)", 0.92])
x.add_row([ "GBDT After Feature Engineering", "(10,10)", 0.93])
print(x)
```

```
+--------------------------------+-----------------------------------------+---------+
|             Model              | Hyperparameters(max_depth,n_estimators) | Test F1 |
+--------------------------------+-----------------------------------------+---------+
|               RF               |                 (14,121)                |   0.92  |
| GBDT After Feature Engineering |                 (10,10)                 |   0.93  |
+--------------------------------+-----------------------------------------+---------+
```

Step by Step Procedure:

1. For the 1st part of the assignment i.e for preferential attachment of followers and followees , i created two more features i.e preferential_followers_train and prefere ntial_followees_train i.e I multiplied number of followers of source node and destination node
2. Also did the same thing for followees as in the 1st point
3. For the 2nd part of the assignment I have created a new features called svd_u_dot and svd_v_dot where I took dot product of source node and destination node of reduced dimensions from matrix factorization
4. For these new features I created a new dataframe and then hstacked both train and test features finally
5. Then I hyperparameter tuned XGBoost with n_estimators and max_depth as hyper params
6. With clf.bestestimator we are overfitting the model with those params , so we'll manually check the params using heatmaps
7. The best params I got is 10 for max_depth and 10 for n_estimators
8. Now I applied XGBoost using these parameters

9. Got test F1 score of 0.93 and train F1 score of 0.97
10. Got AUC of 0.94
11. Found out that most important features are follows_back and cosine_followers

<img src= tqfb.png>