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 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,in
dex=False)
    print("saved the graph into file")
else:
    g=nx.read_edgelist('data/after_eda/train_woheader.csv',delimiter=
',',create_using=nx.DiGraph(),nodetype=int)
    print(nx.info(g))
```

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

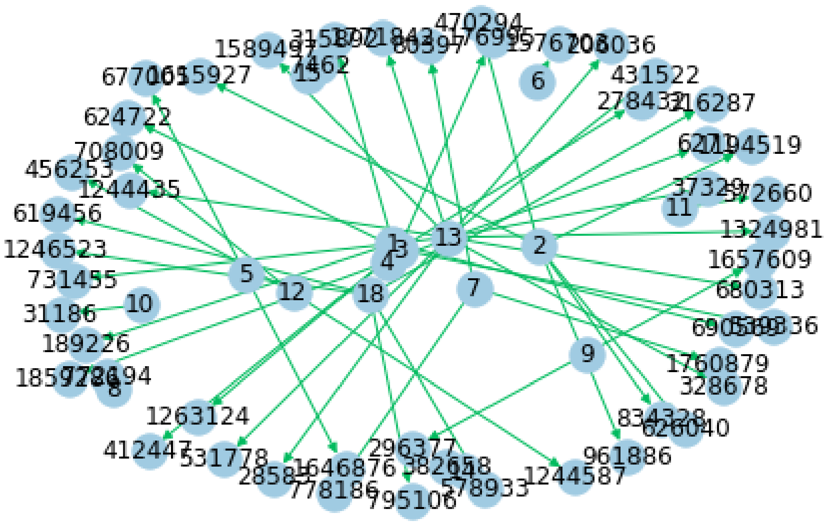
> Displaying a sub graph

In [3]:

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

subgraph=nx.read_edgelist('train_woheader_sample.csv',delimiter=',',cre
```

```
ate_using=nx.DiGraph(),nodetype=int)
# https://stackoverflow.com/questions/9402255/drawing-a-huge-graph-with
-networkx-and-matplotlib

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

```
Name:
Type: DiGraph
Number of nodes: 66
Number of edges: 50
Average in degree:   0.7576
Average out degree:   0.7576
```



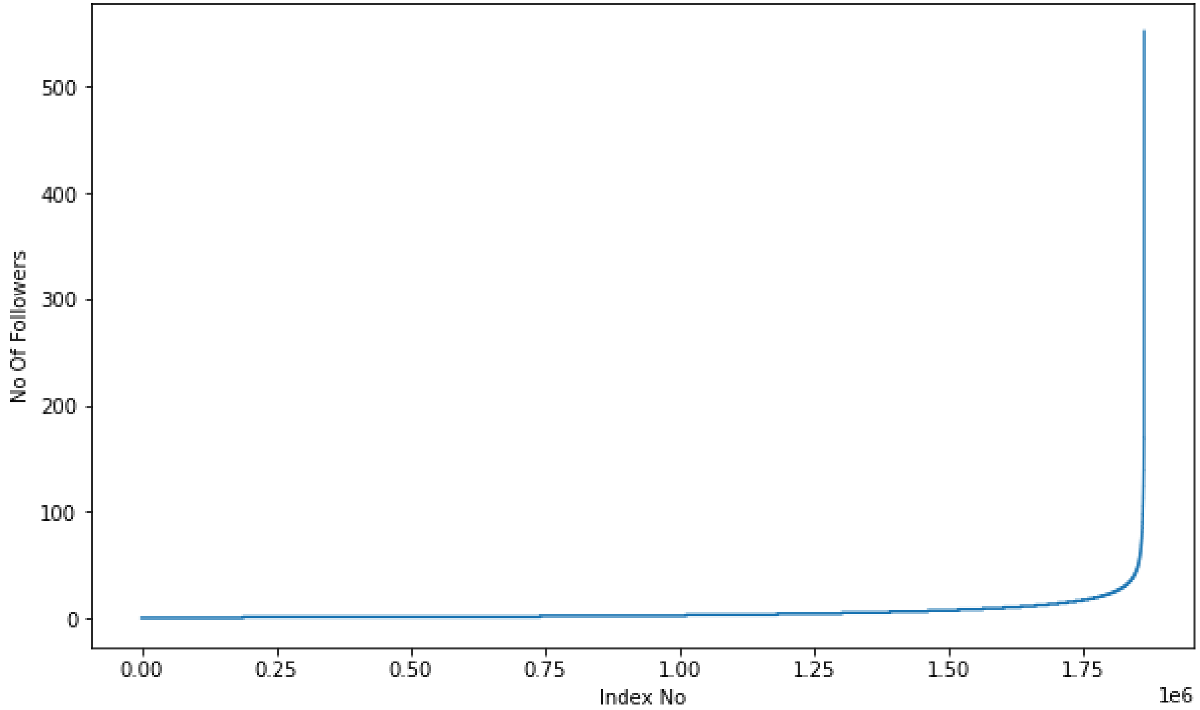
1. Exploratory Data Analysis

In [4]:
```
# No of Unique persons
print("The number of unique persons",len(g.nodes()))
```

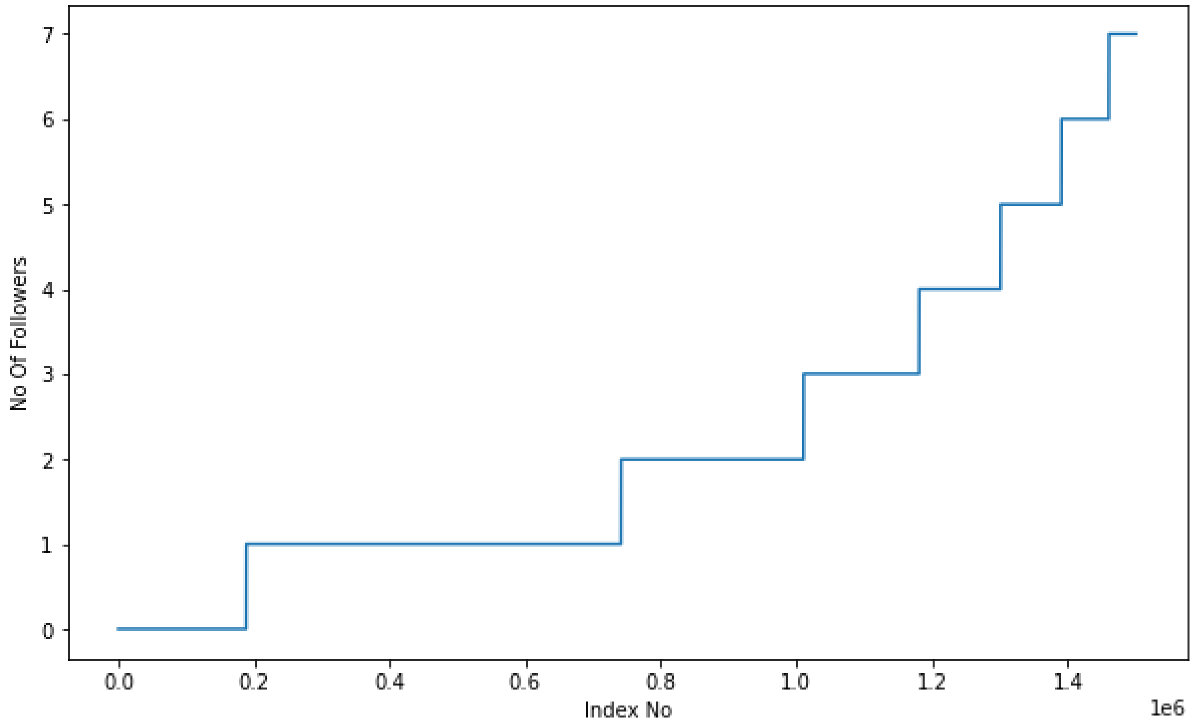
```
The number of unique persons 1862220
```

1.1 No of followers for each person

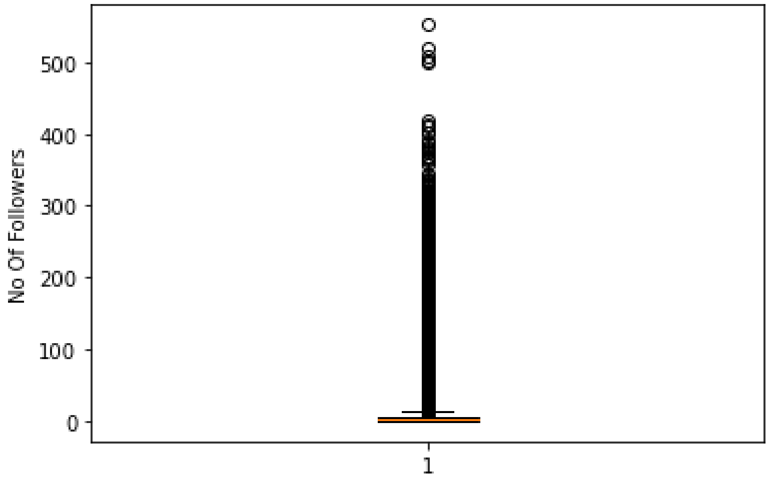
In [5]:
```
indegree_dist = list(dict(g.in_degree()).values())
indegree_dist.sort()
plt.figure(figsize=(10,6))
plt.plot(indegree_dist)
plt.xlabel('Index No')
plt.ylabel('No Of Followers')
plt.show()
```



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
```

99% of data having followers of 40 only.

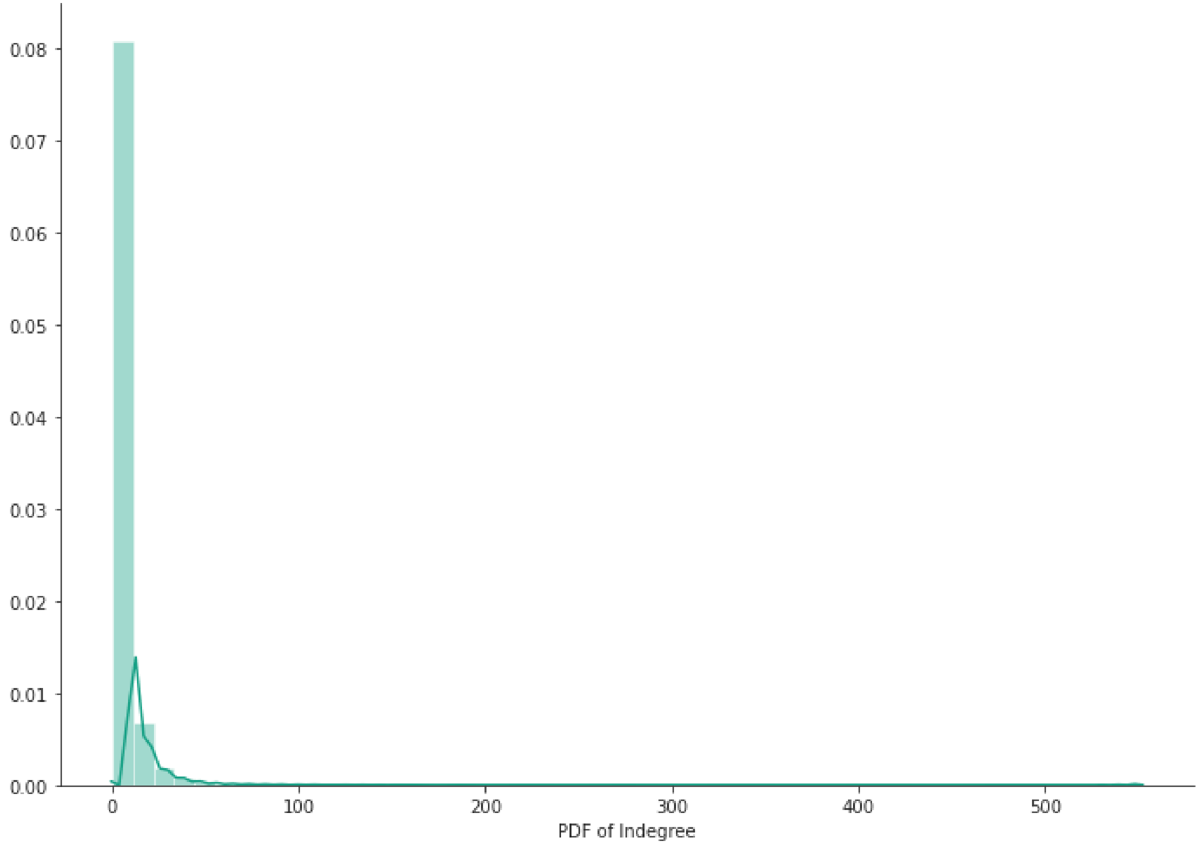
In [9]:
```
### 99-100 percentile
for i in range(10,110,10):
    print(99+(i/100),'percentile value is',np.percentile(indegree_dist,
99+(i/100)))
```

```
99.1 percentile value is 42.0
```

```
99.1 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
```

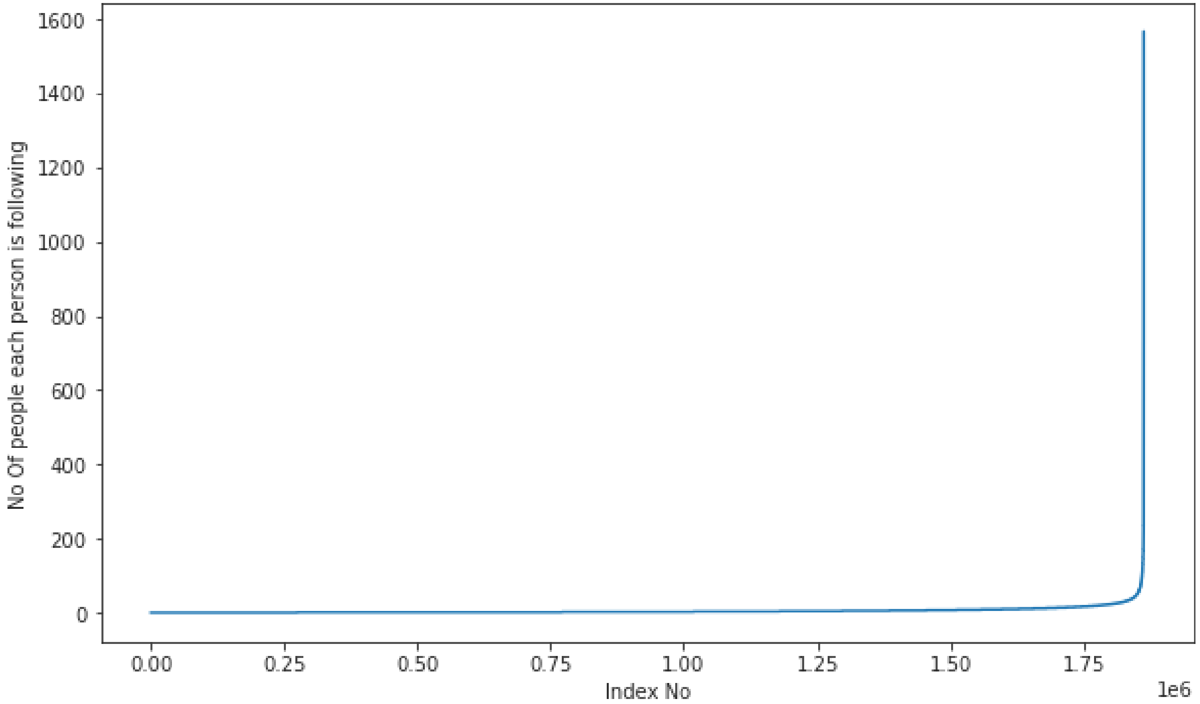
In [10]:

```
%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()
```



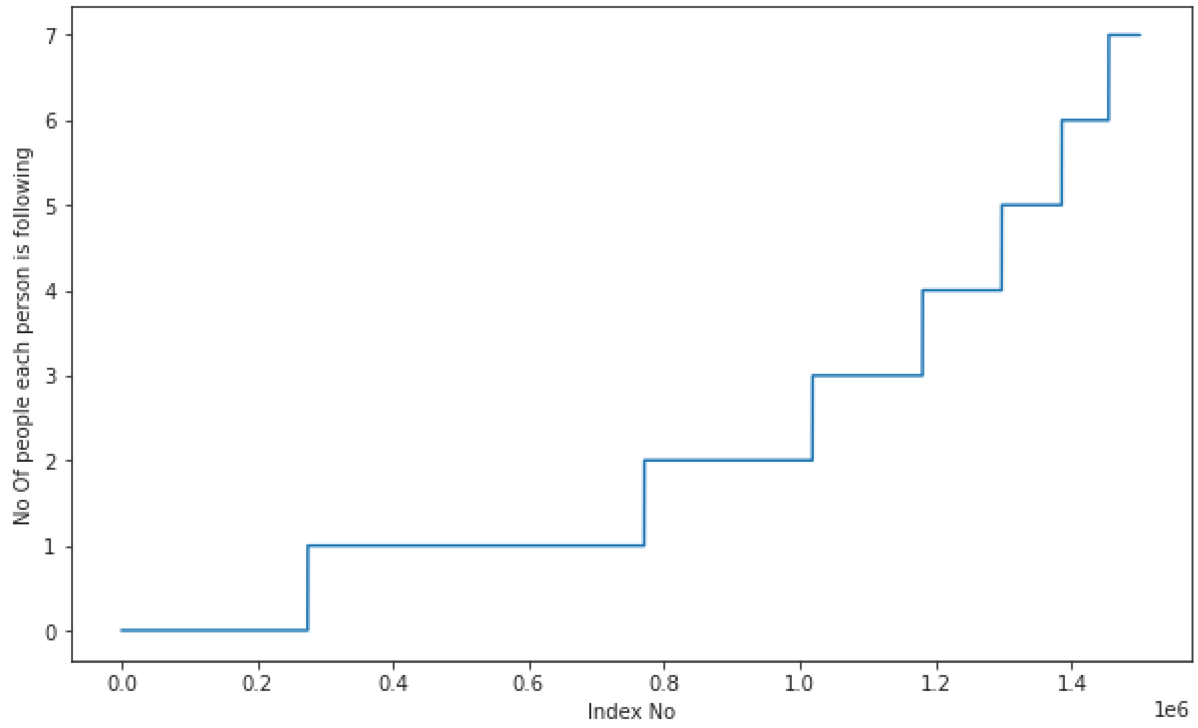
1.2 No of people each person is following

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()
```



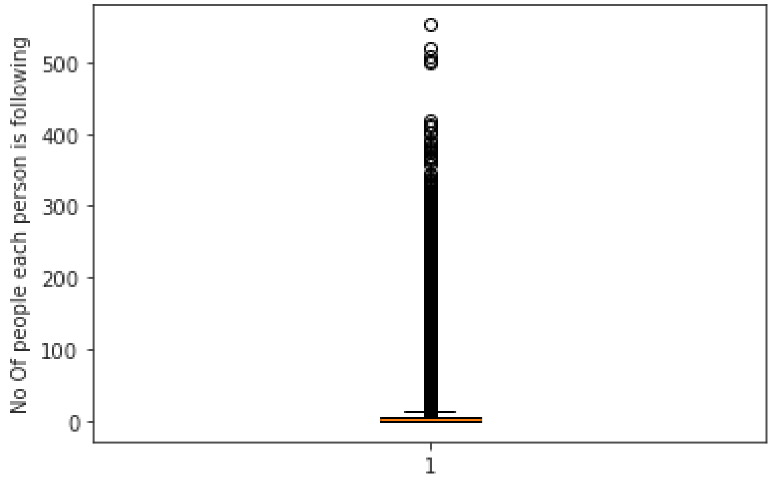
In [12]:

```
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.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
```

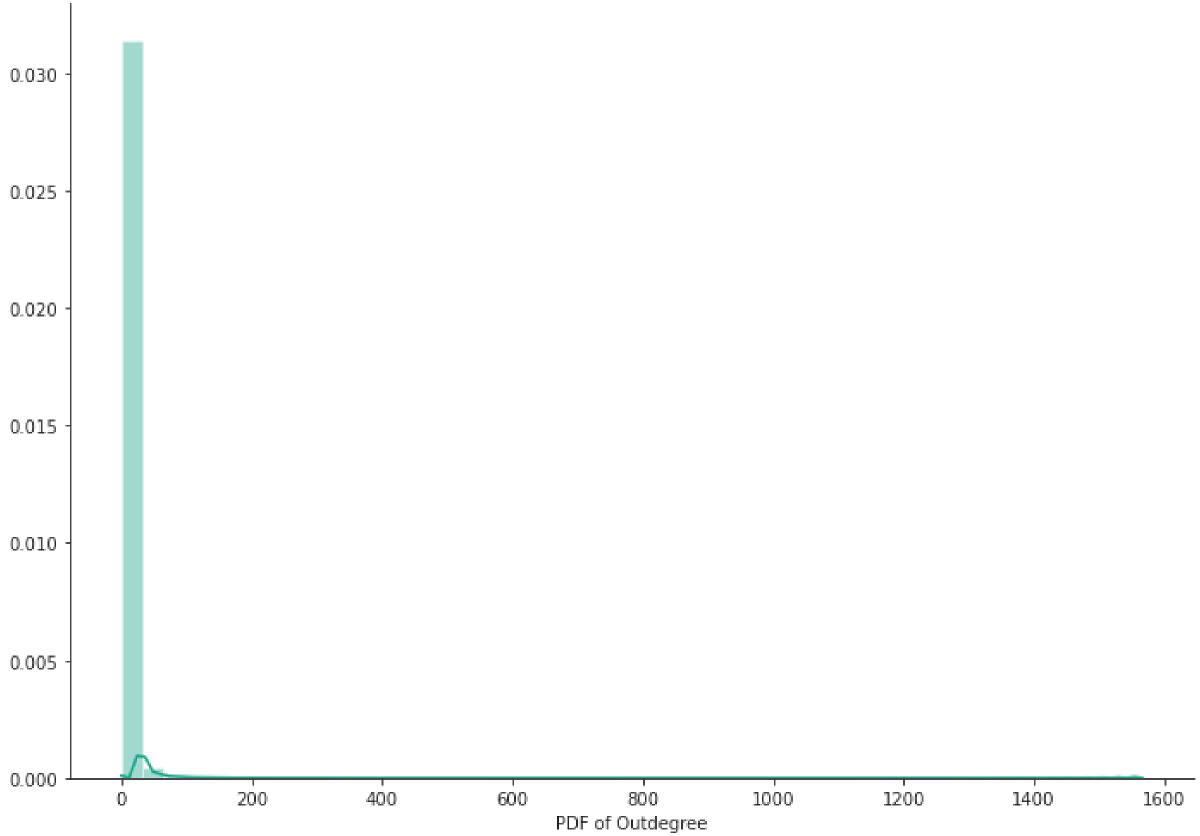
In [15]:
```
### 99-100 percentile
for i in range(10,110,10):
    print(99+(i/100),'percentile value is',np.percentile(outdegree_dist
,99+(i/100)))
```

```
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/le
n(outdegree_dist) )
```

```
No of persons those are not following anyone are 274512 and % is 14.741
115442858524
```

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 10.097786512871
```

```
734
```

In [19]:

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

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

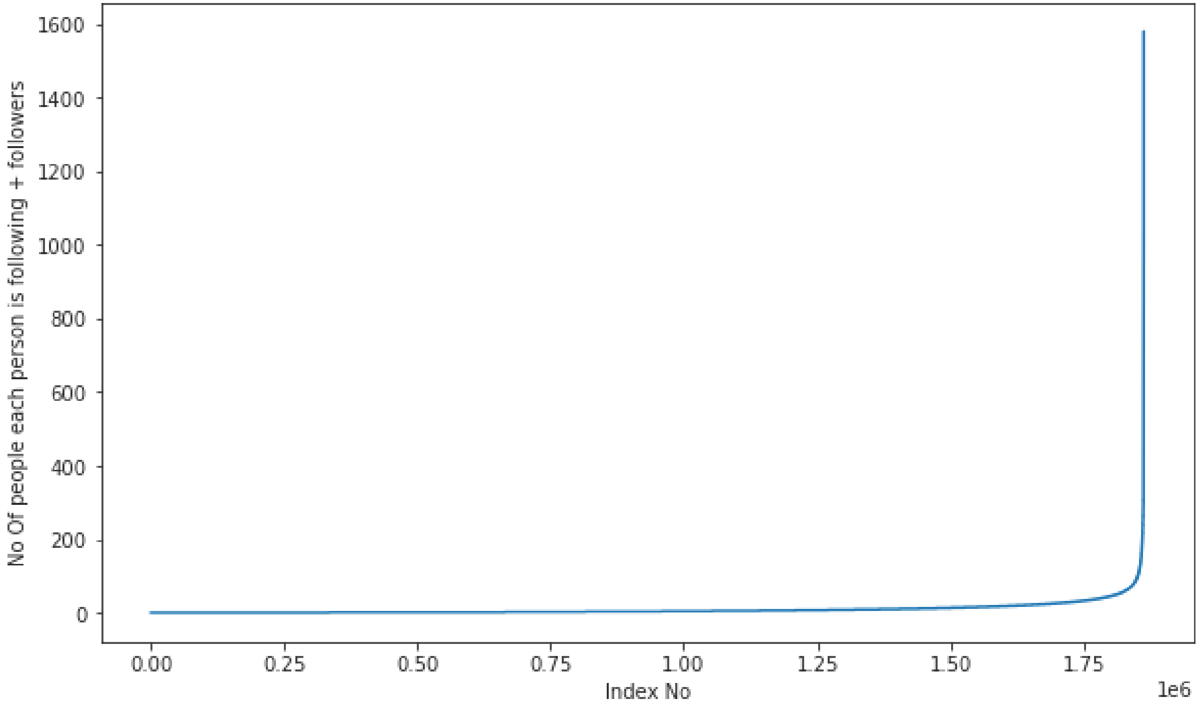
1.3 both followers + following

In [20]:

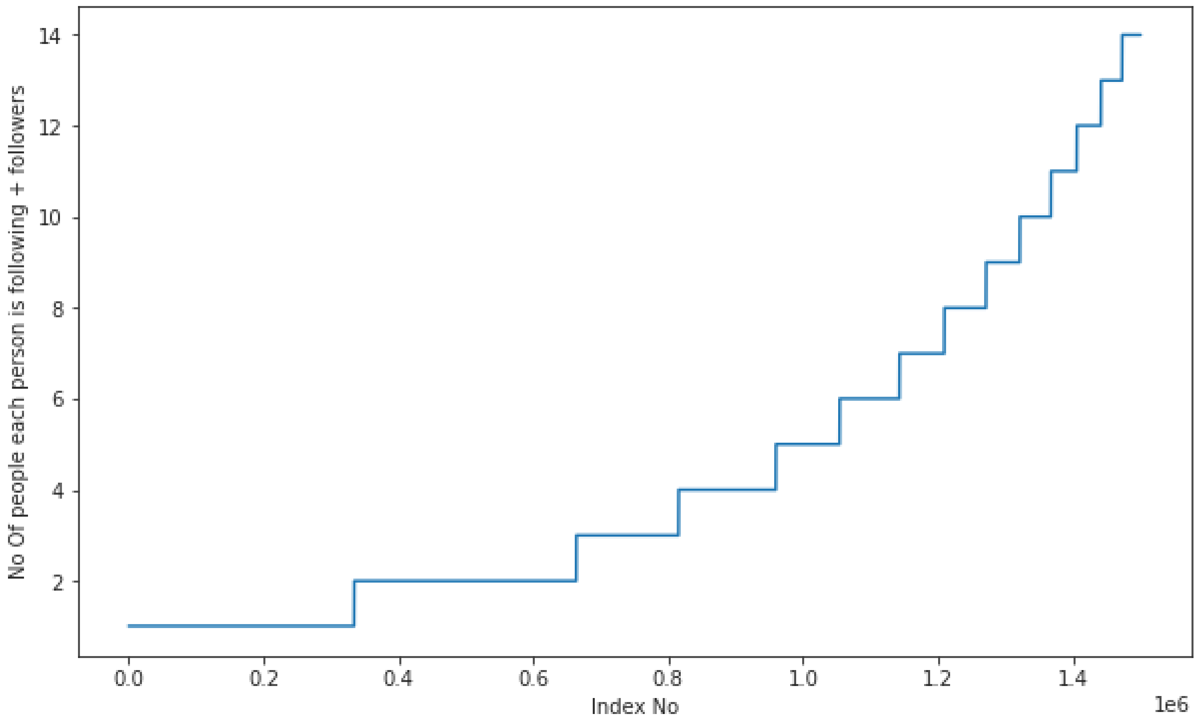
```
from collections import Counter
dict_in = dict(g.in_degree())
dict_out = dict(g.out_degree())
d = Counter(dict_in) + Counter(dict_out)
in_out_degree = np.array(list(d.values()))
```

In [21]:

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



In [22]:
```
in_out_degree_sort = sorted(in_out_degree)
plt.figure(figsize=(10,6))
plt.plot(in_out_degree_sort[0:1500000])
plt.xlabel('Index No')
plt.ylabel('No Of people each person is following + followers')
plt.show()
```



In [23]:
```
### 90-100 percentile
for i in range(0,11):
    print(90+i,'percentile value is',np.percentile(in_out_degree_sort,90+i))
```

```
90 percentile value is 24.0
91 percentile value is 26.0
92 percentile value is 28.0
93 percentile value is 31.0
94 percentile value is 33.0
95 percentile value is 37.0
96 percentile value is 41.0
97 percentile value is 48.0
98 percentile value is 58.0
99 percentile value is 79.0
100 percentile value is 1579.0
```

In [24]:
```
### 99-100 percentile
for i in range(10,110,10):
    print(99+(i/100),'percentile value is',np.percentile(in_out_degree_sort,99+(i/100)))
```

```
99.1 percentile value is 83.0
99.2 percentile value is 87.0
99.3 percentile value is 93.0
99.4 percentile value is 99.0
99.5 percentile value is 108.0
99.6 percentile value is 120.0
99.7 percentile value is 138.0
99.8 percentile value is 168.0
99.9 percentile value is 221.0
100.0 percentile value is 1579.0
```

In [25]:
```
print('Min of no of followers + following is',in_out_degree.min())
print(np.sum(in_out_degree==in_out_degree.min()),' persons having minimum no of followers + following')
```

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

In [26]:
```
print('Max of no of followers + following is',in_out_degree.max())
print(np.sum(in_out_degree==in_out_degree.max()),' persons having maximum no of followers + following')
```

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

In [27]:
```
print('No of persons having followers + following less than 10 are',np.sum(in_out_degree<10))
```

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

In [28]:
```
print('No of weakly connected components',len(list(nx.weakly_connected_components(g))))
count=0
for i in list(nx.weakly_connected_components(g)):
```

```
    if len(i)==2:
        count+=1
print('weakly connected components wit 2 nodes',count)
```

```
No of weakly connected components 45558
weakly connected components wit 2 nodes 32195
```

2. Posing a problem as classification problem

2.1 Generating some edges which are not present in graph for supervised learning

Generated Bad links from graph which are not in graph and whose shortest path is greater than 2.

In [29]:

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


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

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

```
CPU times: user 2.04 s, sys: 2.31 s, total: 4.35 s
Wall time: 4.34 s
```

In [30]:
```
len(missing_edges)
```

Out[30]:
```
9437519
```

2.2 Training and Test data split:

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

In [ ]:
```
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 positi
ve training data only for creating graph
    #and for feature generation
    X_train_pos, X_test_pos, y_train_pos, y_test_pos  = train_test_spli
t(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_spli
t(df_neg,np.zeros(len(df_neg)),test_size=0.2, random_state=9)

    print('='*60)
    print("Number of nodes in the train data graph with edges", X_train
_pos.shape[0],"=",y_train_pos.shape[0])
    print("Number of nodes in the train data graph without edges", X_tr
ain_neg.shape[0],"=", y_train_neg.shape[0])
    print('='*60)
    print("Number of nodes in the test data graph with edges", X_test_p
os.shape[0],"=",y_test_pos.shape[0])
    print("Number of nodes in the test data graph without edges", X_tes
t_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=Fa
lse, 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=Fa
lse, index=False)
else:
    #Graph from Traing data only
    del missing_edges
```

```
Number of nodes in the graph with edges 9437519
Number of nodes in the graph without edges 9437519
============================================================
Number of nodes in the train data graph with edges 7550015 = 7550015
Number of nodes in the train data graph without edges 7550015 = 7550015
============================================================
```

```
Number of nodes in the test data graph with edges 1887504 = 1887504
Number of nodes in the test data graph without edges 1887504 = 1887504
```

In [ ]:

```
if (os.path.isfile('data/after_eda/train_pos_after_eda.csv')) and (os.p
ath.isfile('data/after_eda/test_pos_after_eda.csv')):
    train_graph=nx.read_edgelist('data/after_eda/train_pos_after_eda.cs
v',delimiter=',',create_using=nx.DiGraph(),nodetype=int)
    test_graph=nx.read_edgelist('data/after_eda/test_pos_after_eda.csv'
,delimiter=',',create_using=nx.DiGraph(),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 -- ',t
rY_teN)

    print('no of people present in test but not present in train -- ',t
eY_trN)
    print(' % of people not there in Train but exist in Test in total T
est data are {} %'.format(teY_trN/len(test_nodes_pos)*100))
```

```
Name:
Type: DiGraph
Number of nodes: 1780722
Number of edges: 7550015
Average in degree:   4.2399
Average out degree:   4.2399
Name:
Type: DiGraph
Number of nodes: 1144623
Number of edges: 1887504
Average in degree:   1.6490
```

```
Average out degree:   1.6490
no of people common in train and test --  1063125
no of people present in train but not present in test --  717597
no of people present in test but not present in train --  81498
 % of people not there in Train but exist in Test in total Test data ar
e 7.1200735962845405 %
```

> we have a cold start problem here

In [ ]:

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

    X_train_pos = pd.read_csv('data/after_eda/train_pos_after_eda.csv',
 names=['source_node', 'destination_node'])
    X_test_pos = pd.read_csv('data/after_eda/test_pos_after_eda.csv', n
ames=['source_node', 'destination_node'])
    X_train_neg = pd.read_csv('data/after_eda/train_neg_after_eda.csv',
 names=['source_node', 'destination_node'])
    X_test_neg = pd.read_csv('data/after_eda/test_neg_after_eda.csv', n
ames=['source_node', 'destination_node'])

    print('='*60)
    print("Number of nodes in the train data graph with edges", X_train
_pos.shape[0])
    print("Number of nodes in the train data graph without edges", X_tr
ain_neg.shape[0])
    print('='*60)
    print("Number of nodes in the test data graph with edges", X_test_p
os.shape[0])
```

```
    print("Number of nodes in the test data graph without edges", X_tes
t_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,in
dex=False)
    X_test.to_csv('data/after_eda/test_after_eda.csv',header=False,inde
x=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=Fa
lse,index=False)
```

```
============================================================
Number of nodes in the train data graph with edges 7550015
Number of nodes in the train data graph without edges 7550015
============================================================
Number of nodes in the test data graph with edges 1887504
Number of nodes in the test data graph without edges 1887504
```

In [ ]:

```
print("Data points in train data",X_train.shape)
print("Data points in test data",X_test.shape)
print("Shape of traget variable in train",y_train.shape)
print("Shape of traget variable in test", y_test.shape)
```

```
Data points in train data (15100030, 2)
Data points in test data (3775008, 2)
Shape of traget variable in train (15100030,)
Shape of traget variable in test (3775008,)
```

Social network Graph Link Prediction - Facebook Challenge

In [1]:

```
#Importing Libraries
```

```python
# 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 [2]:
```python
if os.path.isfile('data/after_eda/train_pos_after_eda.csv'):
    train_graph=nx.read_edgelist('data/after_eda/train_pos_after_eda.cs
v',delimiter=',',create_using=nx.DiGraph(),nodetype=int)
```

```
    print(nx.info(train_graph))
else:
    print("please run the FB_EDA.ipynb or download the files from driv
e")
```

```
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/

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

In [3]:

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

In [4]:
```
#one test case
print(jaccard_for_followees(273084,1505602))
```

```
0.0
```

In [5]:
```
#node 1635354 not in graph
print(jaccard_for_followees(273084,1505602))
```

```
0.0
```

In [6]:
```
#for followers
def jaccard_for_followers(a,b):
    try:
        if len(set(train_graph.predecessors(a))) == 0  | len(set(g.pred
ecessors(b))) == 0:
            return 0
        sim = (len(set(train_graph.predecessors(a)).intersection(set(tr
ain_graph.predecessors(b)))))/\
                                 (len(set(train_graph.predecessors(a)).
union(set(train_graph.predecessors(b)))))
        return sim
    except:
        return 0
```

In [7]:
```
print(jaccard_for_followers(273084,470294))
```

```
0
```

In [8]:
```
#node 1635354 not in graph
print(jaccard_for_followees(669354,1635354))
```

```
0
```

2.2 Cosine distance

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

In [9]:
```
#for followees
def cosine_for_followees(a,b):
    try:
        if len(set(train_graph.successors(a))) == 0  | len(set(train_gr
aph.successors(b))) == 0:
            return 0
        sim = (len(set(train_graph.successors(a)).intersection(set(trai
n_graph.successors(b)))))/\
                                    (math.sqrt(len(set(train_graph.succ
essors(a)))*len((set(train_graph.successors(b))))))
        return sim
    except:
        return 0
```

In [10]:
```
print(cosine_for_followees(273084,1505602))
```

```
0.0
```

In [11]:
```
print(cosine_for_followees(273084,1635354))
```

```
0
```

In [12]:
```
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(tr
ain_graph.predecessors(b)))))/\
                                     (math.sqrt(len(set(train_graph.pre
decessors(a))))*(len(set(train_graph.predecessors(b)))))
        return sim
```

```
    except:
        return 0
```

In [13]:
```
print(cosine_for_followers(2,470294))
```

```
0.02886751345948129
```

In [14]:
```
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 [15]:
```
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 [16]:
```
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 [17]:
```
#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 [18]:
```
#if has direct edge then deleting that edge and calculating shortest pa
th
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 [19]:
```
#testing
compute_shortest_path_length(77697, 826021)
```

Out[19]: 10

In [20]:
```
#testing
compute_shortest_path_length(669354,1635354)
```

Out[20]: -1

4.2 Checking for same community

In [21]:
```
#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 [22]:
```
belongs_to_same_wcc(861, 1659750)
```

Out[22]: 0

In [23]:
```
belongs_to_same_wcc(669354,1635354)
```

Out[23]: 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 [24]:
```
#adar index
def calc_adar_in(a,b):
    sum=0
    try:
        n=list(set(train_graph.successors(a)).intersection(set(train_gr
aph.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 [25]:
```
calc_adar_in(1,189226)
```

Out[25]: 0

In [26]:
```
calc_adar_in(669354,1635354)
```

Out[26]: 0

4.4 Is persion was following back:

In [27]:
```
def follows_back(a,b):
    if train_graph.has_edge(b,a):
        return 1
    else:
        return 0
```

In [28]:
```
follows_back(1,189226)
```

Out[28]: 1

In [29]:
```
follows_back(669354,1635354)
```

Out[29]: 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 [30]:
```
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 [31]:
```
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 [32]:
```
mean_katz = float(sum(katz.values())) / len(katz)
print(mean_katz)
```

```
0.00074838009355620188
```

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 [33]:
```
if not os.path.isfile('data/fea_sample/hits.p'):
    hits = nx.hits(train_graph, max_iter=100, tol=1e-08, nstart=None, n
ormalized=True)
    pickle.dump(hits,open('data/fea_sample/hits.p','wb'))
else:
    hits = pickle.load(open('data/fea_sample/hits.p','rb'))
```

In [34]:
```
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 [35]:
```
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 [36]:
```
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 [37]:
```
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(s
kip_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(sk
ip_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 [38]:
```
df_final_train = pd.read_csv('data/after_eda/train_after_eda.csv', skip
rows=skip_train, names=['source_node', 'destination_node'])
df_final_train['indicator_link'] = pd.read_csv('data/train_y.csv', skip
rows=skip_train, names=['indicator_link'])
```

```
print("Our train matrix size ",df_final_train.shape)
df_final_train.head(2)
```

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

Out[38]:

	source_node	destination_node	indicator_link
0	273084	1505602	1
1	365429	1523458	1

In [39]:

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

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

Out[39]:

	source_node	destination_node	indicator_link
0	848424	784690	1
1	1562045	1824397	1

5.2 Adding a set of features

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

1. jaccard_followers
2. jaccard_followees
3. cosine_followers
4. cosine_followees
5. num_followers_s
6. num_followees_s

7. num_followers_d
8. num_followees_d
9. inter_followers
10. inter_followees

In [40]:
```
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 r
ow:
                                            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 r
ow:
                                            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 ro
w:
                                            cosine_for_followers(row['s
ource_node'],row['destination_node']),axis=1)
    df_final_test['cosine_followers'] = df_final_test.apply(lambda row:
                                            cosine_for_followers(row['s
ource_node'],row['destination_node']),axis=1)

    #mapping jaccrd followees to train and test data
    df_final_train['cosine_followees'] = df_final_train.apply(lambda ro
```

```
w:
                                                        cosine_for_followees(row['s
ource_node'],row['destination_node']),axis=1)
    df_final_test['cosine_followees'] = df_final_test.apply(lambda row:
                                                        cosine_for_followees(row['s
ource_node'],row['destination_node']),axis=1)
```

In [41]:
```
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_follo
wees_d, inter_followers, inter_followees
```

In [42]:

```
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(df_final_train)

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

    hdf = HDFStore('data/fea_sample/storage_sample_stage1.h5')
    hdf.put('train_df',df_final_train, format='table', data_columns=Tru
e)
    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.h
5', 'train_df',mode='r')
    df_final_test = read_hdf('data/fea_sample/storage_sample_stage1.h5'
, 'test_df',mode='r')
df_final_train.head()
```

Out[42]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	c
0	273084	1505602	1	0	0.000000	0
1	365429	1523458	1	0	0.023077	0
2	1122952	720171	1	0	0.071429	0
3	201818	992764	1	0	0.100000	0

4	1160801	71690	1	0	0.000000	0

In [43]:
```
df_final_train.head()
```

Out[43]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	c
0	273084	1505602	1	0	0.000000	0
1	365429	1523458	1	0	0.023077	0
2	1122952	720171	1	0	0.071429	0
3	201818	992764	1	0	0.100000	0
4	1160801	71690	1	0	0.000000	0

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 [44]:
```
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: cal
c_adar_in(row['source_node'],row['destination_node']),axis=1)
    #mapping adar index on test
    df_final_test['adar_index'] = df_final_test.apply(lambda row: calc_
adar_in(row['source_node'],row['destination_node']),axis=1)

    #--------------------------------------------------------------------
--------------------------------------
```

```
    #mapping followback or not on train
    df_final_train['follows_back'] = df_final_train.apply(lambda row: f
ollows_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: fol
lows_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: belo
ngs_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: belong
s_to_same_wcc(row['source_node'],row['destination_node']),axis=1)

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

    hdf = HDFStore('data/fea_sample/storage_sample_stage2.h5')
    hdf.put('train_df',df_final_train, format='table', data_columns=Tru
e)
    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.h
5', '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

$$W = \frac{1}{\sqrt{1+|X|}}$$

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

In [45]:
```
#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, 113411.28it/s]
```

In [46]:
```
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,mean_weight_in))
    df_final_train['weight_out'] = df_final_train.source_node.apply(lambda x: Weight_out.get(x,mean_weight_out))

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


    #some features engineerings on the in and out weights
    df_final_train['weight_f1'] = df_final_train.weight_in + df_final_train.weight_out
    df_final_train['weight_f2'] = df_final_train.weight_in * df_final_t
```

```
rain.weight_out
    df_final_train['weight_f3'] = (2*df_final_train.weight_in + 1*df_fi
nal_train.weight_out)
    df_final_train['weight_f4'] = (1*df_final_train.weight_in + 2*df_fi
nal_train.weight_out)

    #some features engineerings on the in and out weights
    df_final_test['weight_f1'] = df_final_test.weight_in + df_final_tes
t.weight_out
    df_final_test['weight_f2'] = df_final_test.weight_in * df_final_tes
t.weight_out
    df_final_test['weight_f3'] = (2*df_final_test.weight_in + 1*df_fina
l_test.weight_out)
    df_final_test['weight_f4'] = (1*df_final_test.weight_in + 2*df_fina
l_test.weight_out)
```

In [47]:
```
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(la
mbda x:pr.get(x,mean_pr))
    df_final_train['page_rank_d'] = df_final_train.destination_node.app
ly(lambda x:pr.get(x,mean_pr))

    df_final_test['page_rank_s'] = df_final_test.source_node.apply(lamb
da 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(la
mbda 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(lamb
da 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(la
mbda 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(lamb
da 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.a
pply(lambda x: hits[1].get(x,0))

    df_final_test['authorities_s'] = df_final_test.source_node.apply(la
mbda x: hits[1].get(x,0))
    df_final_test['authorities_d'] = df_final_test.destination_node.app
ly(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=Tru
e)
    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.h
5', 'train_df',mode='r')
    df_final_test = read_hdf('data/fea_sample/storage_sample_stage3.h5'
, 'test_df',mode='r')
```

In [48]:

```
df_final_train.head()
```

Out[48]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	c
0	273084	1505602	1	0	0.000000	0
1	365429	1523458	1	0	0.023077	0
2	1122952	720171	1	0	0.071429	0
3	201818	992764	1	0	0.100000	0
4	1160801	71690	1	0	0.000000	0

5 rows × 31 columns

Assignments:

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

https://storage.googleapis.com/kaggle-forum-message-attachments/2594/supervised_link_prediction.pdf

3. Tune hyperparameters for XG boost with all these features and check the error metric.

Add new feature Preferential Attachement

Preferential Attachement for Followers

In [49]:
```
#pa= preferential attachment, for train data
pa_src=np.array(df_final_train['num_followers_s'])
pa_des=np.array(df_final_train['num_followers_d'])
pa_followers=[]
for i in range(len(pa_src)):
    pa_followers.append(pa_des[i]*pa_src[i])
df_final_train['Pref_Attach_followers']= pa_followers
df_final_train.head()
```

Out[49]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	c
0	273084	1505602	1	0	0.000000	0
1	365429	1523458	1	0	0.023077	0
2	1122952	720171	1	0	0.071429	0
3	201818	992764	1	0	0.100000	0
4	1160801	71690	1	0	0.000000	0

5 rows × 32 columns

In [50]:
```
df_final_train.shape
```

Out[50]: (100002, 32)

In [51]:
```
df_final_train.columns
```

Out[51]:
```
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',
       'Pref_Attach_followers'],
      dtype='object')
```

In [52]:
```
#pa= preferential attachment ,for test data
pa_src=np.array(df_final_test['num_followers_s'])
pa_des=np.array(df_final_test['num_followers_d'])
pa_followers=[]
for i in range(len(pa_src)):
    pa_followers.append(pa_des[i]*pa_src[i])
df_final_test['Pref_Attach_followees']= pa_followers
df_final_test.head()
```

Out[52]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	c
0	848424	784690	1	0	0.000000	0
1	1562045	1824397	1	0	0.000000	0

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	c
2	1432006	1842942	1	0	0.000000	0
3	1718418	37205	1	0	0.068966	0
4	911169	392490	1	0	0.238095	0

5 rows × 32 columns

In [53]:
```
df_final_test.shape
```

Out[53]: (50002, 32)

Preferential Attachement for Followees

In [54]:
```
#pa= preferential attachment for train data
pa_src=np.array(df_final_train['num_followees_s'])
pa_des=np.array(df_final_train['num_followees_d'])
pa_followees=[]
for i in range(len(pa_src)):
    pa_followees.append(pa_des[i]*pa_src[i])
df_final_train['Pref_Attach_followees']= pa_followees
df_final_train.head()
```

Out[54]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	c
0	273084	1505602	1	0	0.000000	0
1	365429	1523458	1	0	0.023077	0

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	c
2	1122952	720171	1	0	0.071429	0
3	201818	992764	1	0	0.100000	0
4	1160801	71690	1	0	0.000000	0

5 rows × 33 columns

In [55]:
```
df_final_train.shape
```

Out[55]: (100002, 33)

In [56]:
```
#pa= preferential attachment for test data
pa_src=np.array(df_final_test['num_followees_s'])
pa_des=np.array(df_final_test['num_followees_d'])
pa_followees=[]
for i in range(len(pa_src)):
    pa_followees.append(pa_des[i]*pa_src[i])
df_final_test['Pref_Attach_followees']= pa_followees
df_final_test.head()
```

Out[56]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	c
0	848424	784690	1	0	0.000000	0
1	1562045	1824397	1	0	0.000000	0
2	1432006	1842942	1	0	0.000000	0

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	c
3	1718418	37205	1	0	0.068966	0
4	911169	392490	1	0	0.238095	0

5 rows × 33 columns

In [57]:
```
df_final_test.shape
```

Out[57]:
```
(50002, 33)
```

In [58]:
```
df_final_test.columns
```

Out[58]:
```
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',
       'Pref_Attach_followers', 'Pref_Attach_followees'],
      dtype='object')
```

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 in
edx 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 [117]:
```
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.Seri
es)

    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.Se
ries)

    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.Serie
s)

    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.Ser
ies)

    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(p
d.Series)
    #====================================================================
=================================
```

```
    hdf = HDFStore('data/fea_sample/storage_sample_stage4.h5')
    hdf.put('train_df',df_final_train, format='table', data_columns=Tru
e)
    hdf.put('test_df',df_final_test, format='table', data_columns=True)
    hdf.close()
```

Add new feature svd_dot

In [118]:

```
#SVD train data
s1,s2,s3,s4,s5,s6=df_final_train['svd_u_s_1'],df_final_train['svd_u_s_
2'],df_final_train['svd_u_s_3'],df_final_train['svd_u_s_4'],df_final_tr
ain['svd_u_s_5'],df_final_train['svd_u_s_6']
s7,s8,s9,s10,s11,s12=df_final_train['svd_v_s_1'],df_final_train['svd_v_
s_2'],df_final_train['svd_v_s_3'],df_final_train['svd_v_s_4'],df_final_
train['svd_v_s_5'],df_final_train['svd_v_s_6']

d1,d2,d3,d4,d5,d6=df_final_train['svd_u_d_1'],df_final_train['svd_u_d_
2'],df_final_train['svd_u_d_3'],df_final_train['svd_u_d_4'],df_final_tr
ain['svd_u_d_5'],df_final_train['svd_u_d_6']
d7,d8,d9,d10,d11,d12=df_final_train['svd_v_d_1'],df_final_train['svd_v_
d_2'],df_final_train['svd_v_d_3'],df_final_train['svd_v_d_4'],df_final_
train['svd_v_d_5'],df_final_train['svd_v_d_6']
```

In [119]:

```
svd_dot=[]
for i in range(len(np.array(s1))):
    d_1=[]
    d_2=[]
    d_1.append(np.array(s1[i]))
    d_1.append(np.array(s2[i]))
    d_1.append(np.array(s3[i]))
    d_1.append(np.array(s4[i]))
    d_1.append(np.array(s5[i]))
    d_1.append(np.array(s6[i]))
    d_1.append(np.array(s7[i]))
    d_1.append(np.array(s8[i]))
```

```
    d_1.append(np.array(s9[i]))
    d_1.append(np.array(s10[i]))
    d_1.append(np.array(s11[i]))
    d_1.append(np.array(s12[i]))
    d_2.append(np.array(d1[i]))
    d_2.append(np.array(d2[i]))
    d_2.append(np.array(d3[i]))
    d_2.append(np.array(d4[i]))
    d_2.append(np.array(d5[i]))
    d_2.append(np.array(d6[i]))
    d_2.append(np.array(d7[i]))
    d_2.append(np.array(d8[i]))
    d_2.append(np.array(d9[i]))
    d_2.append(np.array(d10[i]))
    d_2.append(np.array(d11[i]))
    d_2.append(np.array(d12[i]))
    svd_dot.append(np.dot(d_1,d_2))
df_final_train['svd_dot']=svd_dot
#https://www.tutorialspoint.com/numpy/numpy_dot.htm
```

In [120]:
```
df_final_train.head()
```

Out[120]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	c
0	273084	1505602	1	0	0.000000	0
1	365429	1523458	1	0	0.023077	0
2	1122952	720171	1	0	0.071429	0
3	201818	992764	1	0	0.100000	0
4	1160801	71690	1	0	0.000000	0

5 rows × 58 columns

In [121]:

```
#SVD test data
s1,s2,s3,s4,s5,s6=df_final_test['svd_u_s_1'],df_final_test['svd_u_s_2'
],df_final_test['svd_u_s_3'],df_final_test['svd_u_s_4'],df_final_test[
'svd_u_s_5'],df_final_test['svd_u_s_6']
s7,s8,s9,s10,s11,s12=df_final_test['svd_v_s_1'],df_final_test['svd_v_s_
2'],df_final_test['svd_v_s_3'],df_final_test['svd_v_s_4'],df_final_test
['svd_v_s_5'],df_final_test['svd_v_s_6']

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

In [122]:

```
svd_dot=[]
for i in range(len(np.array(s1))):
    d_1=[]
    d_2=[]
    d_1.append(np.array(s1[i]))
    d_1.append(np.array(s2[i]))
    d_1.append(np.array(s3[i]))
    d_1.append(np.array(s4[i]))
    d_1.append(np.array(s5[i]))
    d_1.append(np.array(s6[i]))
    d_1.append(np.array(s7[i]))
    d_1.append(np.array(s8[i]))
    d_1.append(np.array(s9[i]))
    d_1.append(np.array(s10[i]))
    d_1.append(np.array(s11[i]))
    d_1.append(np.array(s12[i]))
    d_2.append(np.array(d1[i]))
    d_2.append(np.array(d2[i]))
    d_2.append(np.array(d3[i]))
```

```
    d_2.append(np.array(d4[i]))
    d_2.append(np.array(d5[i]))
    d_2.append(np.array(d6[i]))
    d_2.append(np.array(d7[i]))
    d_2.append(np.array(d8[i]))
    d_2.append(np.array(d9[i]))
    d_2.append(np.array(d10[i]))
    d_2.append(np.array(d11[i]))
    d_2.append(np.array(d12[i]))
    svd_dot.append(np.dot(d_1,d_2))
df_final_test['svd_dot']=svd_dot
#https://www.tutorialspoint.com/numpy/numpy_dot.htm
```

In [123]:

```
df_final_test.head()
```

Out[123]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	c
0	848424	784690	1	0	0.000000	0
1	1562045	1824397	1	0	0.000000	0
2	1432006	1842942	1	0	0.000000	0
3	1718418	37205	1	0	0.068966	0
4	911169	392490	1	0	0.238095	0

5 rows × 58 columns

In [124]:

```
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 [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', 't
est_df',mode='r')
```

In [3]:
```
df_final_train.head()
```

Out[3]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	c
0	273084	1505602	1	0	0.000000	0
1	365429	1523458	1	0	0.023077	0
2	1122952	720171	1	0	0.071429	0
3	201818	992764	1	0	0.100000	0
4	1160801	71690	1	0	0.000000	0

5 rows × 58 columns

In [4]:
```
df_final_test.head()
```

Out[4]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	c
0	848424	784690	1	0	0.000000	0

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	c
1	1562045	1824397	1	0	0.000000	0
2	1432006	1842942	1	0	0.000000	0
3	1718418	37205	1	0	0.068966	0
4	911169	392490	1	0	0.238095	0

5 rows × 58 columns

In [5]:
```
df_final_train.columns
```

Out[5]:
```
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',
       'Pref_Attach_followers', 'Pref_Attach_followees', 'svd_u_s_1',
       'svd_u_s_2', 'svd_u_s_3', 'svd_u_s_4', 'svd_u_s_5', 'svd_u_s_6',
       'svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3', 'svd_u_d_4', 'svd_u_d_5',
       'svd_u_d_6', 'svd_v_s_1', 'svd_v_s_2', 'svd_v_s_3', 'svd_v_s_4',
       'svd_v_s_5', 'svd_v_s_6', 'svd_v_d_1', 'svd_v_d_2', 'svd_v_d_3',
       'svd_v_d_4', 'svd_v_d_5', 'svd_v_d_6', 'svd_dot'],
      dtype='object')
```

In [6]:
```
y_train = df_final_train.indicator_link
y_test = df_final_test.indicator_link
```

In [7]:
```
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)
```

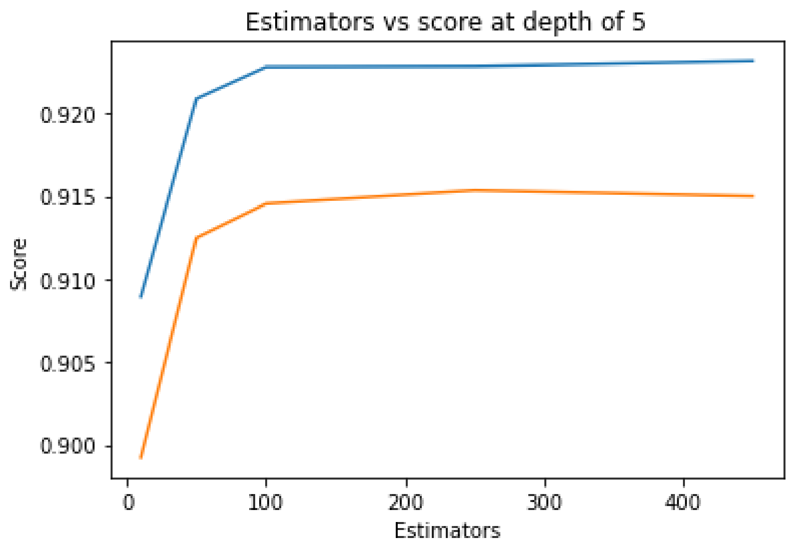
Random Forest

In [132]:
```
estimators = [10,50,100,250,450]
train_scores = []
test_scores = []
for i in estimators:
    clf = RandomForestClassifier(bootstrap=True, class_weight=None, cri
terion='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,ran
dom_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 0.9089390658830807 test Score 0.8992434449
165717
Estimators =  50 Train Score 0.9208917767850635 test Score 0.9125044427
254292
Estimators =  100 Train Score 0.9227863292460214 test Score 0.914545683
2571596
```

```
Estimators =  250 Train Score 0.9228264273040393 test Score 0.915345247
084487
Estimators =  450 Train Score 0.9231363864282132 test Score 0.914993700
1259973
```

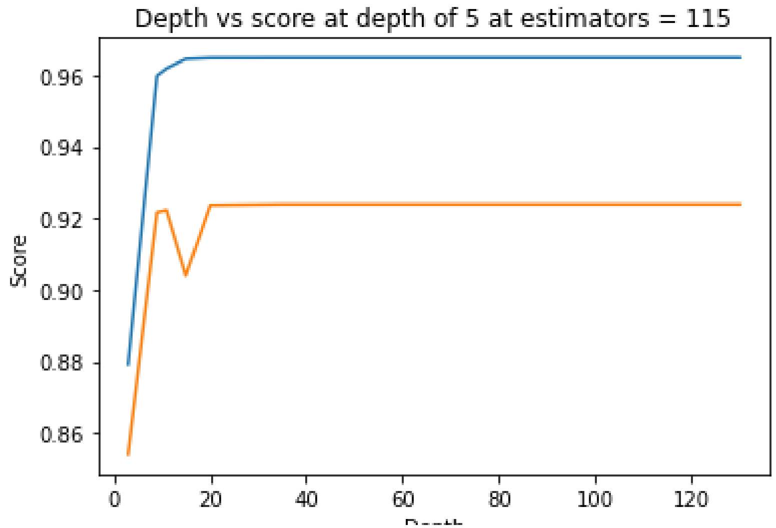
Out[132]:
```
Text(0.5, 1.0, 'Estimators vs score at depth of 5')
```



In [133]:
```
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, cri
terion='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,r
andom_state=25,verbose=0,warm_start=False)
    clf.fit(df_final_train,y_train)
    train_sc = f1_score(y_train,clf.predict(df_final_train))
    test_sc = f1_score(y_test,clf.predict(df_final_test))
    test_scores.append(test_sc)
```

```
    train_scores.append(train_sc)
    print('depth = ',i,'Train Score',train_sc,'test Score',test_sc)
plt.plot(depths,train_scores,label='Train Score')
plt.plot(depths,test_scores,label='Test Score')
plt.xlabel('Depth')
plt.ylabel('Score')
plt.title('Depth vs score at depth of 5 at estimators = 115')
plt.show()
```

```
depth =  3 Train Score 0.8792637051439198 test Score 0.8540307824776666
depth =  9 Train Score 0.9598138186569221 test Score 0.921680050678885
depth =  11 Train Score 0.9618488628026413 test Score 0.922326818354510
6
depth =  15 Train Score 0.964521384928717 test Score 0.9039907212509127
depth =  20 Train Score 0.9648295460332658 test Score 0.923661543658165
9
depth =  35 Train Score 0.9648520649793757 test Score 0.923882298307088
2
depth =  50 Train Score 0.9648520649793757 test Score 0.923882298307088
2
depth =  70 Train Score 0.9648520649793757 test Score 0.923882298307088
2
depth =  130 Train Score 0.9648520649793757 test Score 0.92388229830708
82
```



Depth

In [134]:
```
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_s
tate=25)

rf_random.fit(df_final_train,y_train)
```

```
mean test scores [0.96302753 0.96303605 0.96161726 0.96309172 0.9646340
8]
mean train scores [0.96439527 0.96416869 0.96211981 0.96383528 0.965866
33]
```

In [135]:
```
print(rf_random.best_estimator_)
```

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

In [136]:
```
clf = RandomForestClassifier(bootstrap=True, class_weight=None, criteri
on='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=Fal
se)
```

In [137]:
```
clf.fit(df_final_train,y_train)
y_train_pred = clf.predict(df_final_train)
y_test_pred = clf.predict(df_final_test)
```

In [138]:
```
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))
```

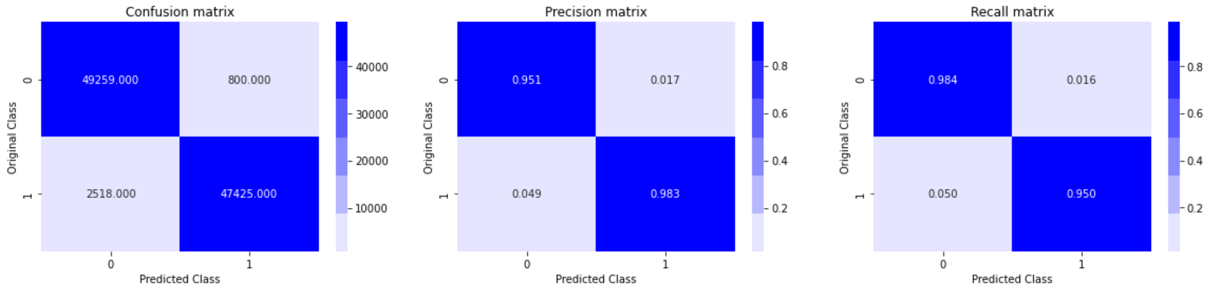
```
Train f1 score 0.9662007986309185
Test f1 score 0.9244187274301523
```

In [8]:
```
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')
```

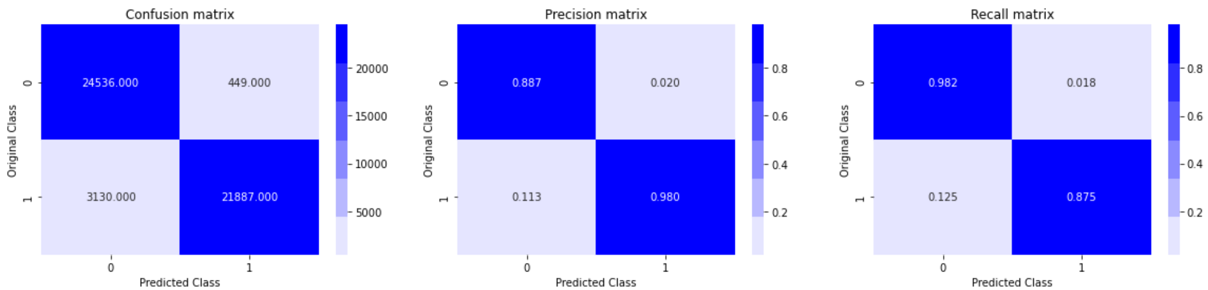
```
    plt.title("Precision matrix")

    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 [140]:
```
print('Train confusion_matrix')
plot_confusion_matrix(y_train,y_train_pred)
print('Test confusion_matrix')
plot_confusion_matrix(y_test,y_test_pred)
```

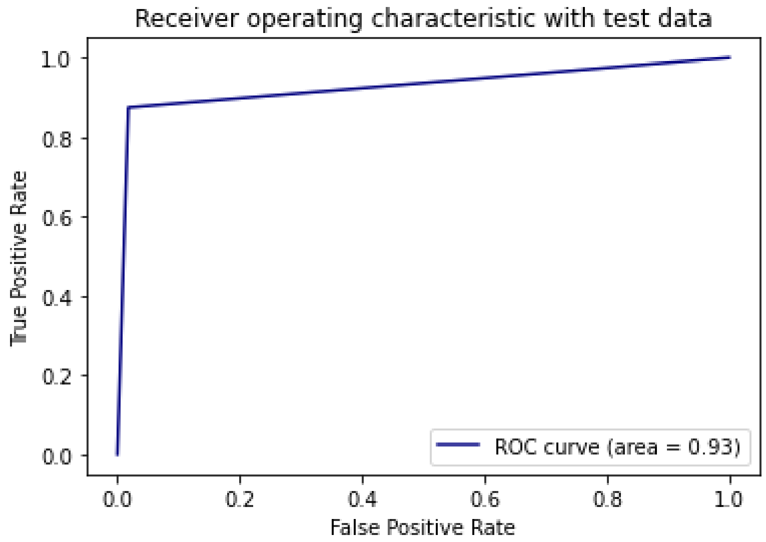
```
Train confusion_matrix
```




```
Test confusion_matrix
```

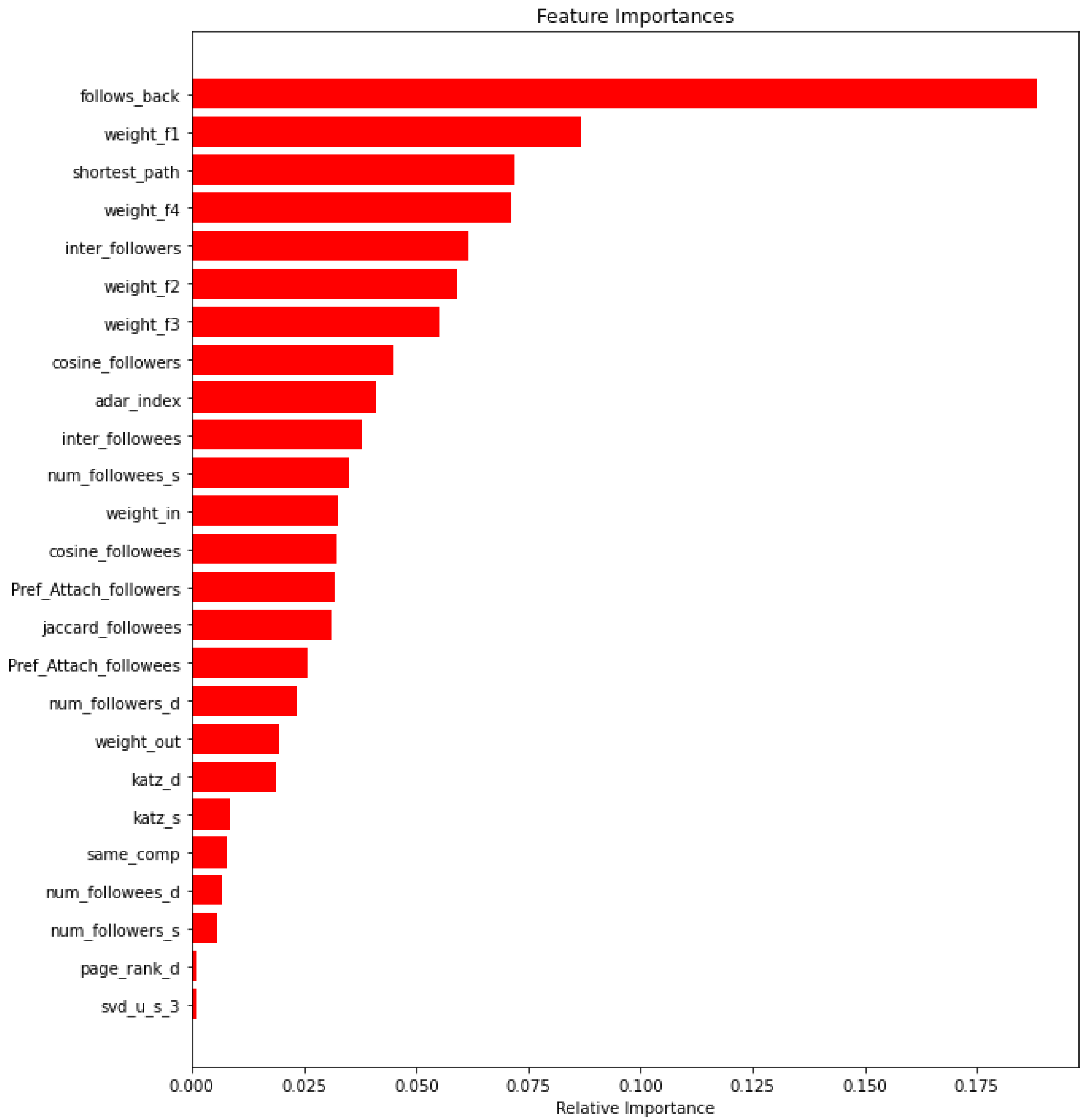


In [141]:
```
from sklearn.metrics import roc_curve, auc
fpr,tpr,ths = roc_curve(y_test,y_test_pred)
auc_sc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='navy',label='ROC curve (area = %0.2f)' % auc_
sc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic with test data')
plt.legend()
plt.show()
```



In [142]:
```
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='c
```

```
enter')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```


Feature Importances
follows_back
weight_f1
shortest_path
weight_f4
inter_followers
weight_f2
weight_f3
cosine_followers
adar_index
inter_followees
num_followees_s
weight_in
cosine_followees
Pref_Attach_followers
jaccard_followees
Pref_Attach_followees
num_followers_d
weight_out
katz_d
katz_s
same_comp
num_followees_d
num_followers_s
page_rank_d
svd_u_s_3
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
Relative Importance

Applying XGBOOST

In [27]:
```
import xgboost as xgb
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
clf = xgb.XGBClassifier()
param_dist = {"n_estimators":sp_randint(105,125),
              "max_depth": sp_randint(10,15)
              }
model = RandomizedSearchCV(clf, param_distributions=param_dist,n_iter=5
,cv=3,scoring='f1',random_state=25, return_train_score=True).fit(df_fin
al_train,y_train)
```

In [28]:
```
print(model.best_estimator_)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-
1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=1
0,
              min_child_weight=1, missing=nan, monotone_constraints
='()',
              n_estimators=109, n_jobs=0, num_parallel_tree=1, random_s
tate=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=
1,
              tree_method='exact', validate_parameters=1, verbosity=Non
e)
```

In [29]:
```
clf=xgb.XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylev
el=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=109
,
        n_jobs=1, nthread=None, objective='binary:logistic', random_stat
e=0,
        reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
        silent=True, subsample=1)
```

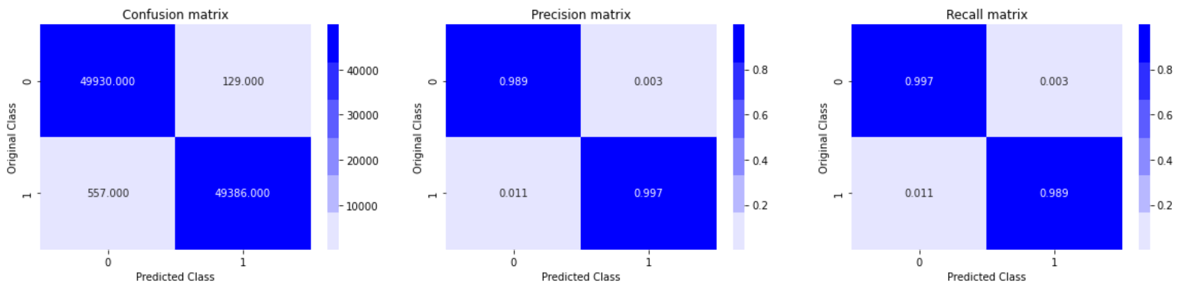
In [22]:
```
clf.fit(df_final_train,y_train)
y_train_pred = clf.predict(df_final_train)
y_test_pred = clf.predict(df_final_test)
```

In [31]:
```
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))
```

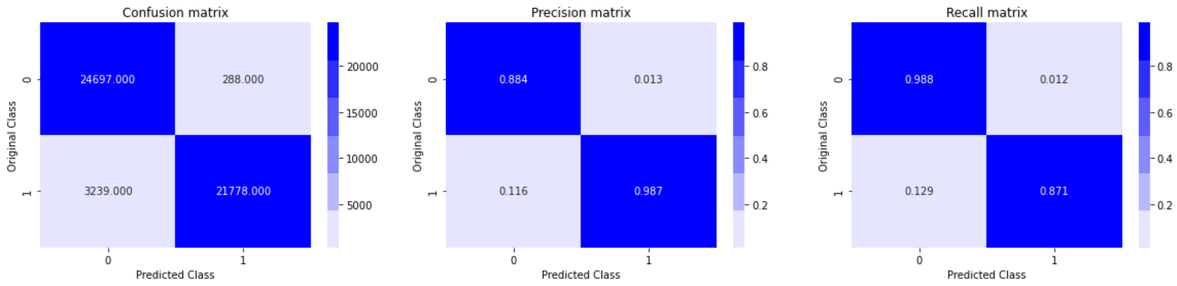
```
Train f1 score 0.9931026161796939
Test f1 score 0.9250897351485675
```

In [32]:
```
print('Train confusion_matrix')
plot_confusion_matrix(y_train,y_train_pred)
print('Test confusion_matrix')
plot_confusion_matrix(y_test,y_test_pred)
```

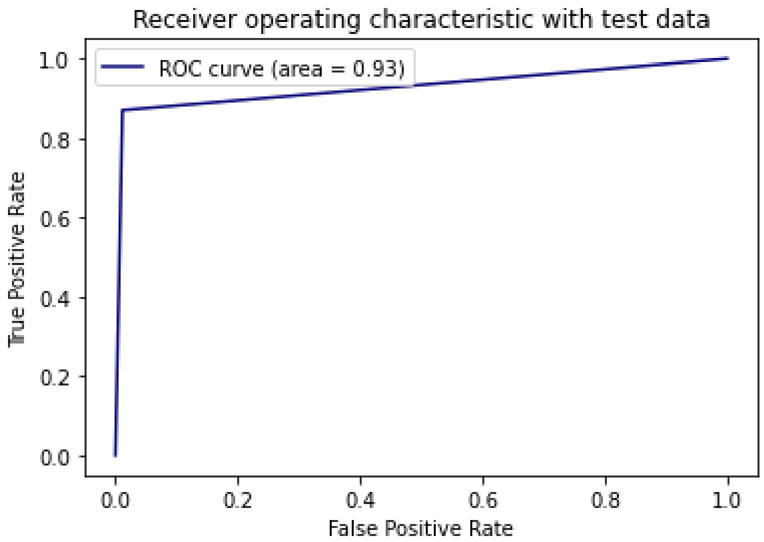
```
Train confusion_matrix
```




```
Test confusion_matrix
```

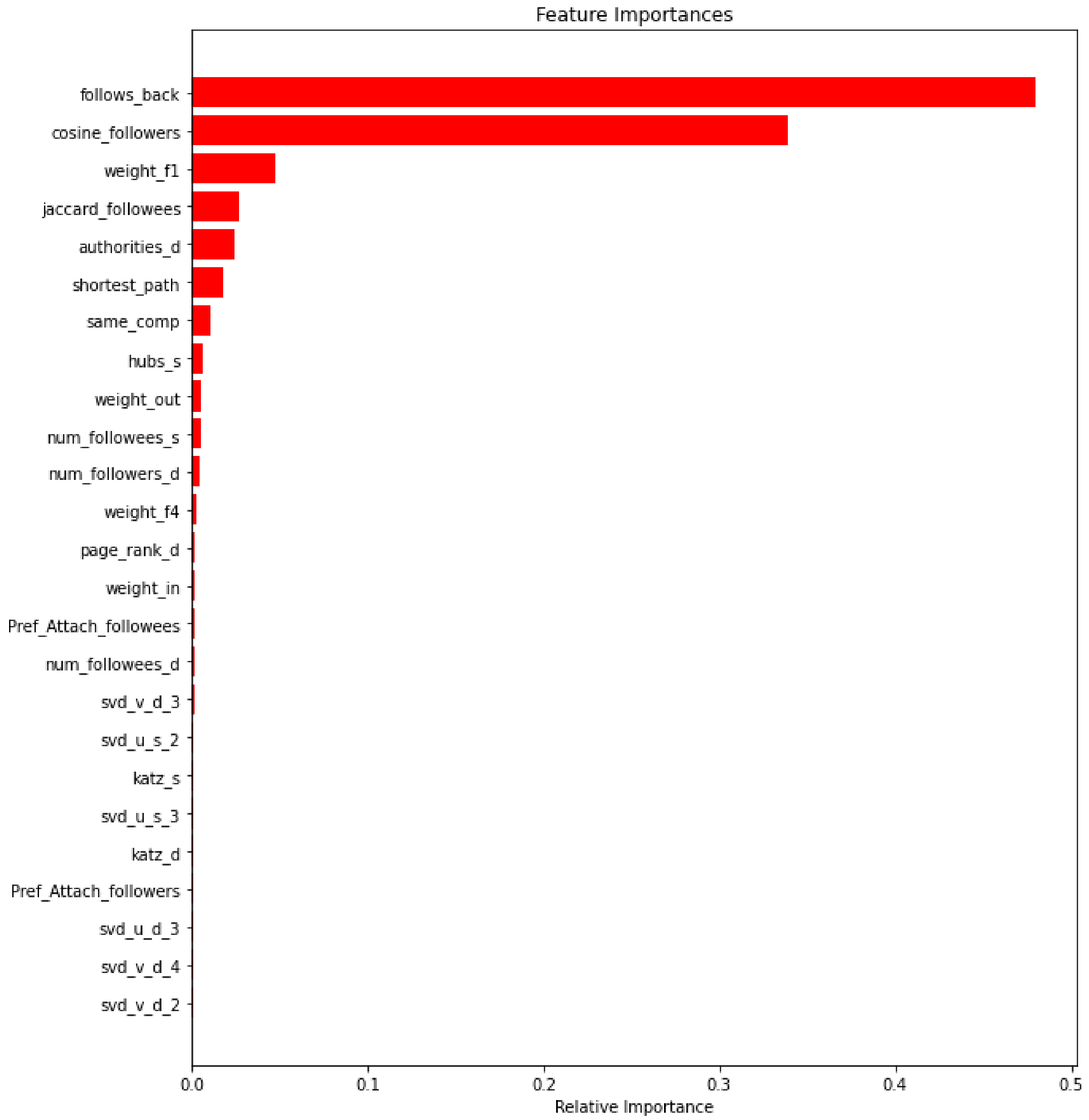


In [33]:
```
from sklearn.metrics import roc_curve, auc
fpr,tpr,ths = roc_curve(y_test,y_test_pred)
auc_sc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='navy',label='ROC curve (area = %0.2f)' % auc_
sc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic with test data')
plt.legend()
plt.show()
```



In [34]:

```
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='c
enter')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```


Feature Importances
follows_back
cosine_followers
weight_f1
jaccard_followees
authorities_d
shortest_path
same_comp
hubs_s
weight_out
num_followees_s
num_followers_d
weight_f4
page_rank_d
weight_in
Pref_Attach_followees
num_followees_d
svd_v_d_3
svd_u_s_2
katz_s
svd_u_s_3
katz_d
Pref_Attach_followers
svd_u_d_3
svd_v_d_4
svd_v_d_2
0.0
0.1
0.2
0.3
0.4
0.5
Relative Importance

Conclusion

In [39]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model","Train f1_score","Test f1_score"]
x.add_row(["RandomForestClassifier",0.9662007986309185 ,0.9244187274301
523])
x.add_row(["XGBClassifier",0.9931026161796939 ,0.9250897351485675])

print(x)
```

```
+------------------------+--------------------+--------------------+
|         Model          |   Train f1_score   |   Test f1_score    |
+------------------------+--------------------+--------------------+
| RandomForestClassifier | 0.9662007986309185 | 0.9244187274301523 |
|     XGBClassifier      | 0.9931026161796939 | 0.9250897351485675 |
+------------------------+--------------------+--------------------+
```