

Facebook Friend Recommendation using Graph mining

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 recruiting challenge on kaggle <https://www.kaggle.com/c/FacebookRecruiting>
(<https://www.kaggle.com/c/FacebookRecruiting>)

data contains two columns source and destination eac edge in graph

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

Mapping the problem into supervised learning problem:

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

Business objectives and constraints:

- No low-latency requirement.
- Probability of prediction is useful to recommend highest 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 arithmetic operations on arrays
# matplotlib: used to plot graphs
import matplotlib
import matplotlib.pyplot 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('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('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=False)

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

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

Name:

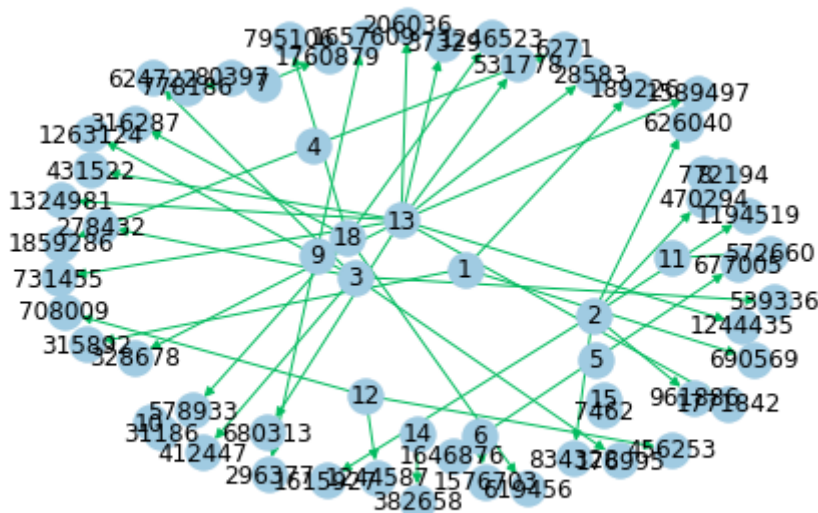
Type: DiGraph

Number of nodes: 66

Number of edges: 50

Average in degree: 0.7576

Average out degree: 0.7576



Exploratory Data Analysis

1. EDA

In [4]:

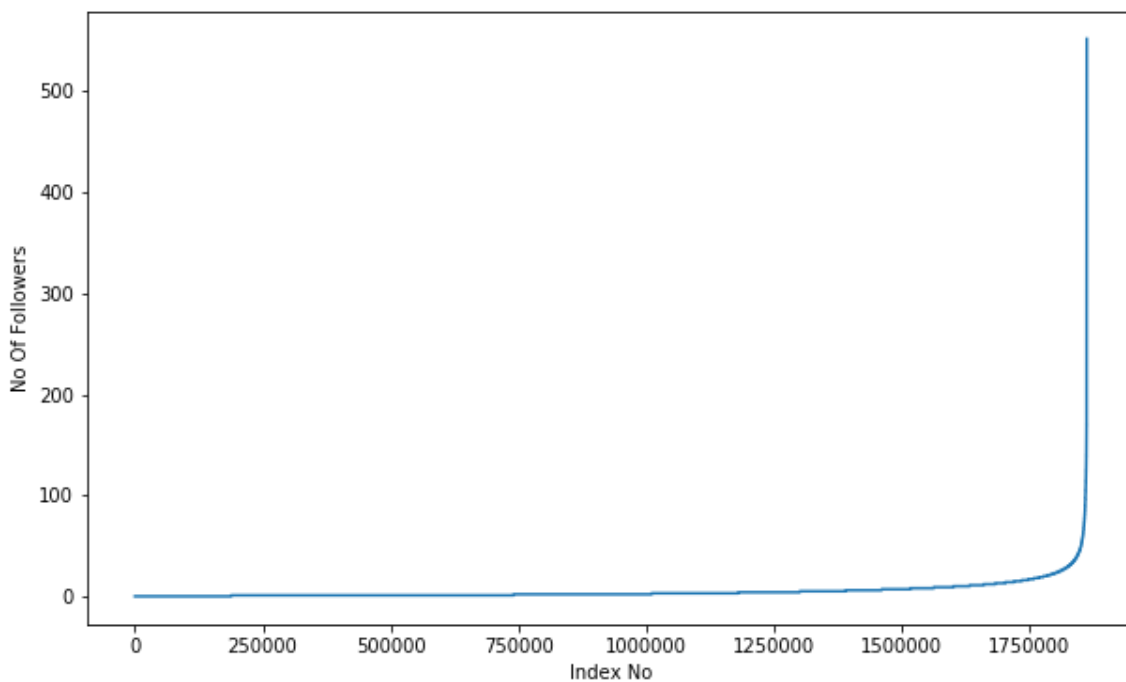
```
# No of Unique persons  
print("The number of unique persons",len(g.nodes()))
```

The number of unique persons 1862220

1.1 No of followers for each person

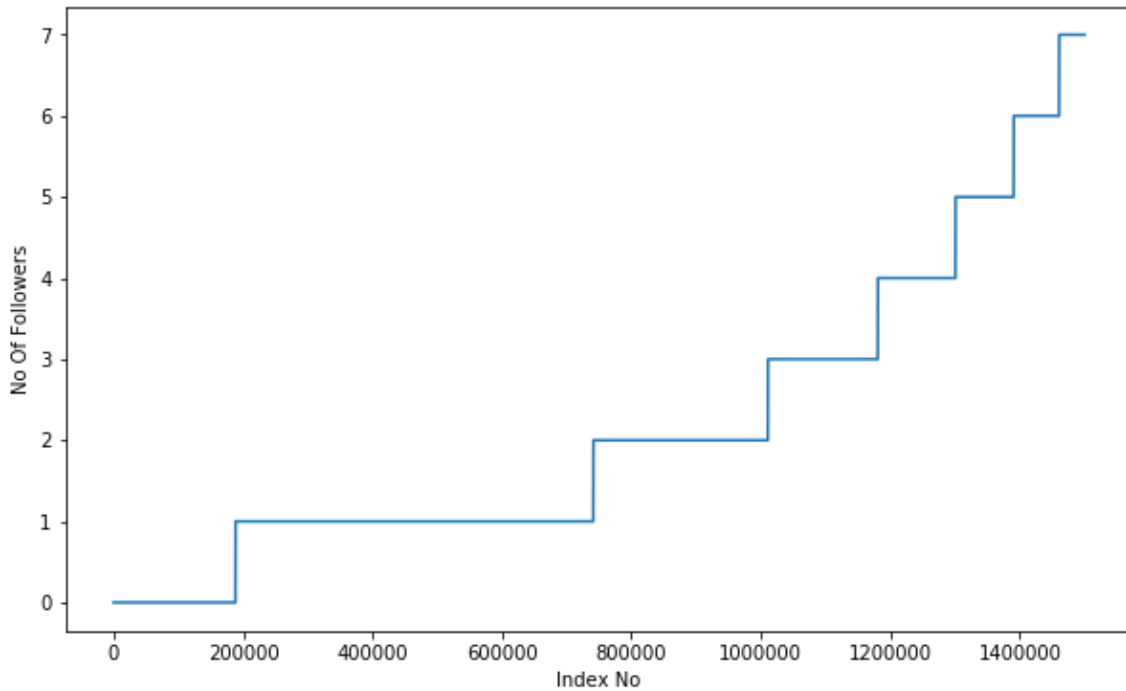
In [5]:

```
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()
```



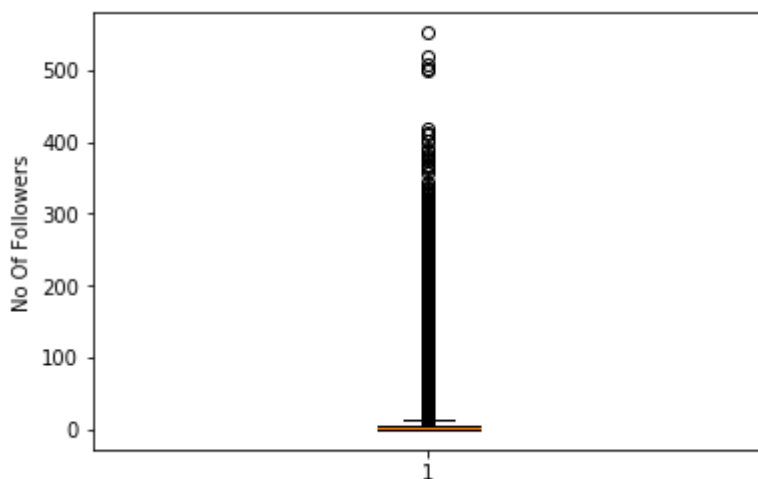
In [6]:

```
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]:

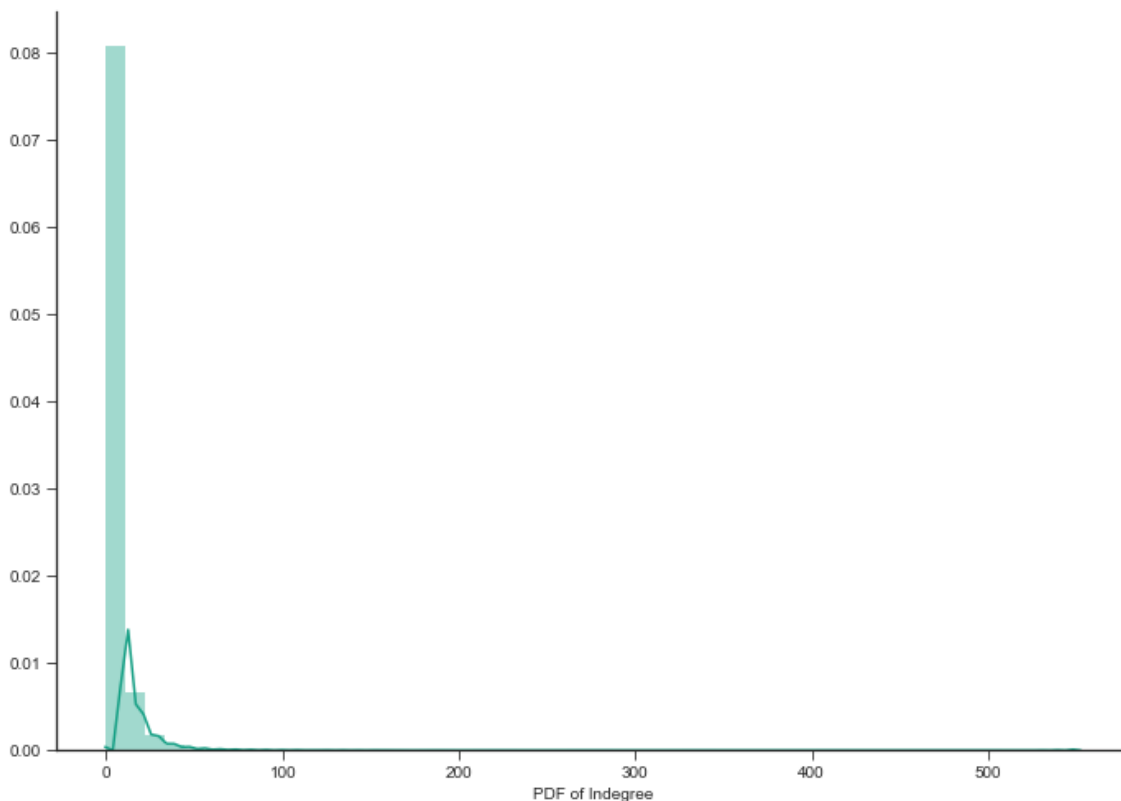
```
### 99-100 percentile
for i in range(10,110,10):
    print(99+(i/100),'percentile value is',np.percentile(indegree_dist,99+(i/100)))
```

```
99.1 percentile value is 42.0
99.2 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()
```

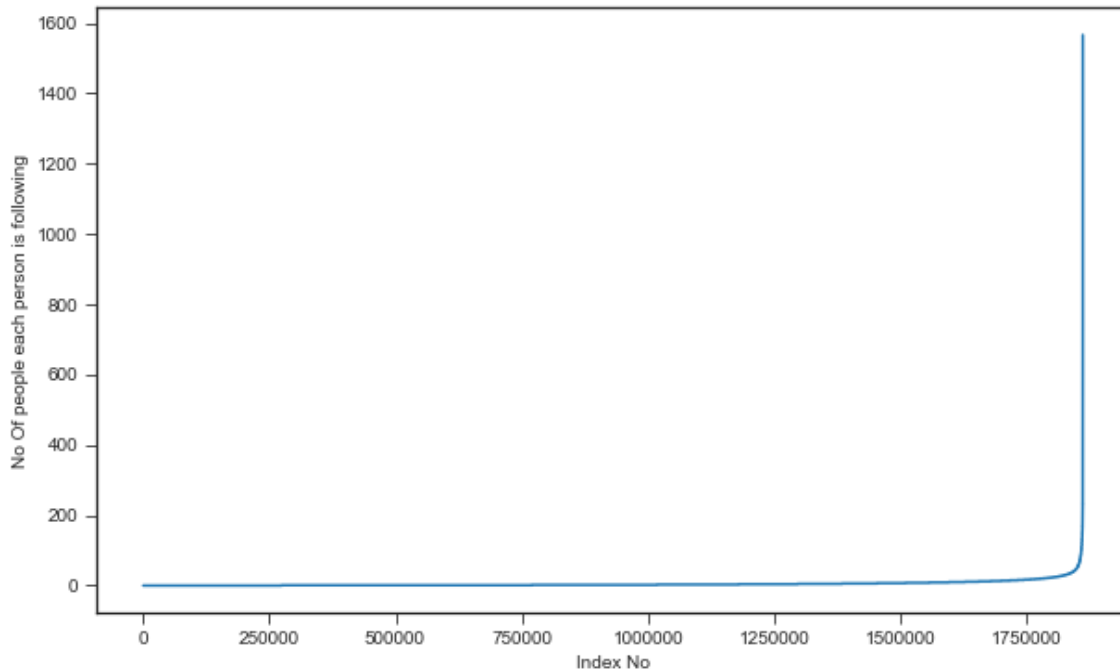
D:\installed\Anaconda3\lib\site-packages\matplotlib\axes_axes.py:6571: UserWarning: The 'normed' kwarg is deprecated, and has been replaced by the 'density' kwarg.
warnings.warn("The 'normed' kwarg is deprecated, and has been "



1.2 No of people each person is following

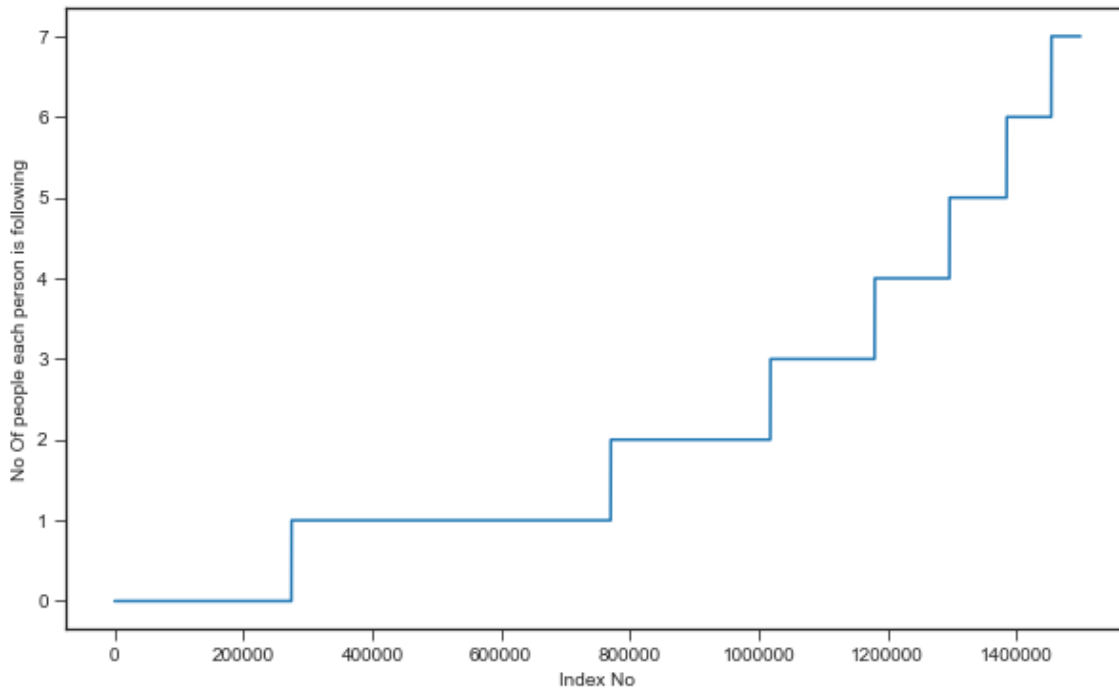
In [11]:

```
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()
```



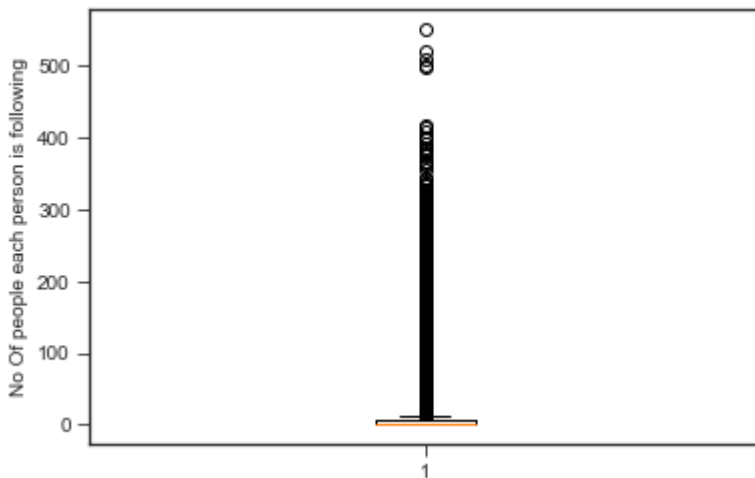
In [12]:

```
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)))
```

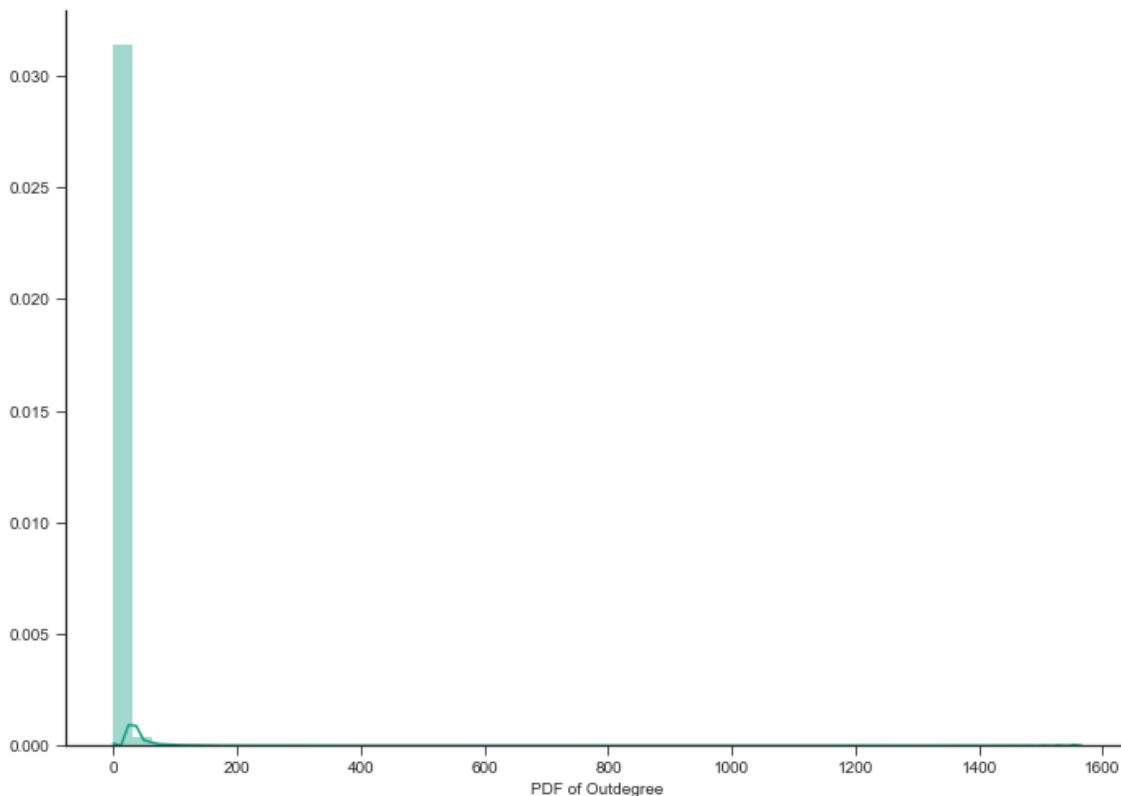
```
99.1 percentile value is 42.0
99.2 percentile value is 45.0
99.3 percentile value is 48.0
99.4 percentile value is 52.0
99.5 percentile value is 56.0
99.6 percentile value is 63.0
99.7 percentile value is 73.0
99.8 percentile value is 90.0
99.9 percentile value is 123.0
100.0 percentile value is 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()
```

D:\installed\Anaconda3\lib\site-packages\matplotlib\axes_axes.py:6571: UserWarning: The 'normed' kwarg is deprecated, and has been replaced by the 'density' kwarg.

warnings.warn("The 'normed' kwarg is deprecated, and has been "



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 [19]:

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

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

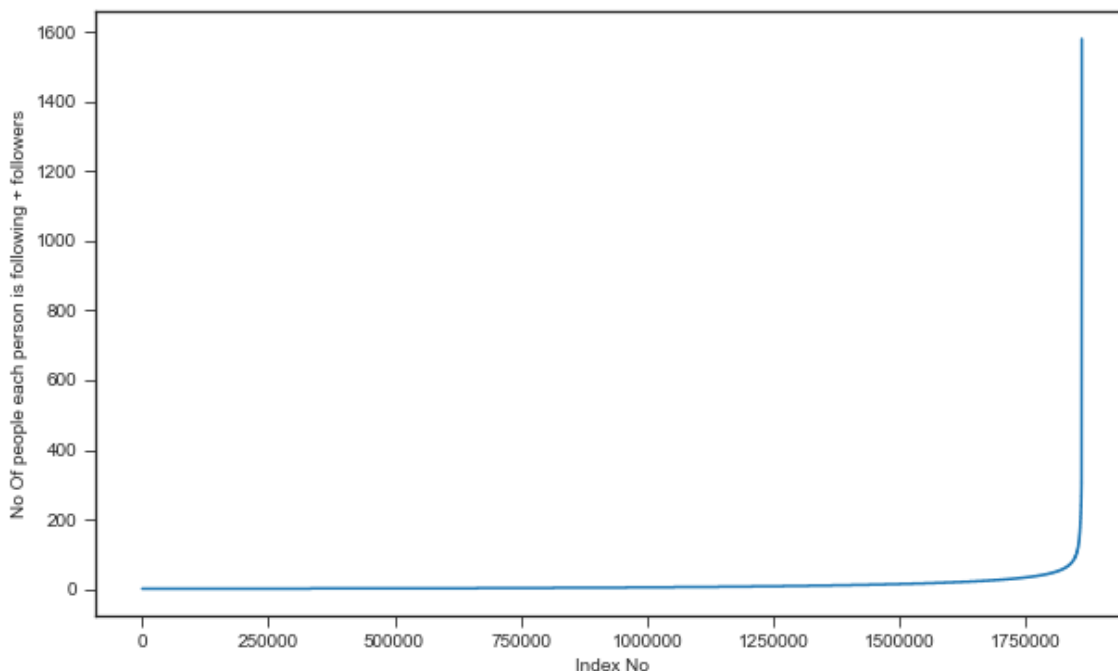
1.3 both followers + following

In [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()))
```

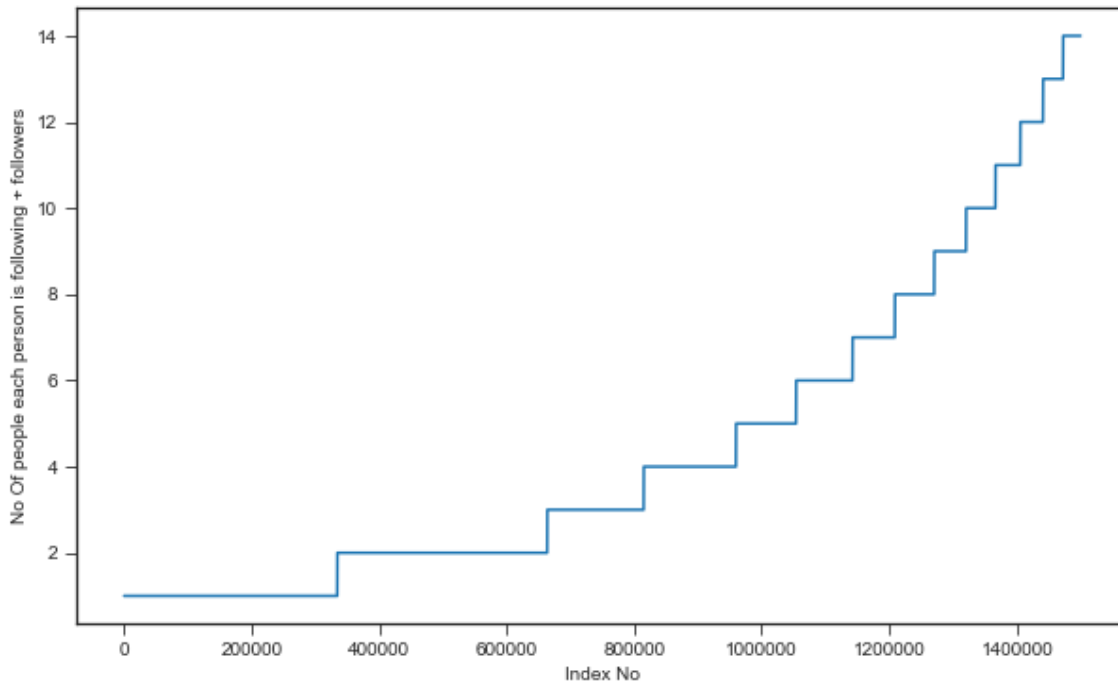
In [21]:

```
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 the 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 [46]:

```
%%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: 5.08 s

In [47]:

```
len(missing_edges)
```

Out[47]:

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 [48]:

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

    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
    #Spiltting data into 80-20
    #positive links and negative links seperatly because we need positive training data
    only for creating graph
    #and for feature generation
    X_train_pos, X_test_pos, y_train_pos, y_test_pos = train_test_split(df_pos,np.ones
(len(df_pos)),test_size=0.2, random_state=9)
    X_train_neg, X_test_neg, y_train_neg, y_test_neg = train_test_split(df_neg,np.zero
s(len(df_neg)),test_size=0.2, random_state=9)

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

    #removing header and saving
    X_train_pos.to_csv('data/after_eda/train_pos_after_eda.csv',header=False, index=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
```

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 [49]:

```
if (os.path.isfile('data/after_eda/train_pos_after_eda.csv')) and (os.path.isfile('data/after_eda/test_pos_after_eda.csv')):
    train_graph=nx.read_edgelist('data/after_eda/train_pos_after_eda.csv',delimiter=',',create_using=nx.DiGraph(),nodetype=int)
    test_graph=nx.read_edgelist('data/after_eda/test_pos_after_eda.csv',delimiter=',',create_using=nx.DiGraph(),nodetype=int)
    print(nx.info(train_graph))
    print(nx.info(test_graph))

    # finding the unique nodes in the both train and test graphs
    train_nodes_pos = set(train_graph.nodes())
    test_nodes_pos = set(test_graph.nodes())

    trY_teY = len(train_nodes_pos.intersection(test_nodes_pos))
    trY_teN = len(train_nodes_pos - test_nodes_pos)
    teY_trN = len(test_nodes_pos - train_nodes_pos)

    print('no of people common in train and test -- ',trY_teY)
    print('no of people present in train but not present in test -- ',trY_teN)

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

Name:

Type: DiGraph

Number of nodes: 1780722

Number of edges: 7550015

Average in degree: 4.2399

Average out degree: 4.2399

Name:

Type: DiGraph

Number of nodes: 1144623

Number of edges: 1887504

Average in degree: 1.6490

Average out degree: 1.6490

no of people common in train and test -- 1063125

no of people present in train but not present in test -- 717597

no of people present in test but not present in train -- 81498

% of people not there in Train but exist in Test in total Test data are

7.1200735962845405 %

we have a cold start problem here

In [50]:

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

    X_train_pos = pd.read_csv('data/after_eda/train_pos_after_eda.csv', names=['source_
node', 'destination_node'])
    X_test_pos = pd.read_csv('data/after_eda/test_pos_after_eda.csv', names=['source_no
de', 'destination_node'])
    X_train_neg = pd.read_csv('data/after_eda/train_neg_after_eda.csv', names=['source_
node', 'destination_node'])
    X_test_neg = pd.read_csv('data/after_eda/test_neg_after_eda.csv', names=['source_no
de', 'destination_node'])

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

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

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

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

In [51]:

```
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,)
```

Feature Engineering

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 arithmetic operations on arrays
# matplotlib: used to plot graphs
import matplotlib
import matplotlib.pyplot 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 [35]:

```
if os.path.isfile('train_pos_after_eda.csv'):
    train_graph=nx.read_edgelist('train_pos_after_eda.csv',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 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/> (<http://www.statisticshowto.com/jaccard-index/>)

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

In [0]:

```
#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 [0]:

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

0.0

In [0]:

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

0.0

In [0]:

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

In [0]:

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

0

In [0]:

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

0

2.2 Cosine distance

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

In [0]:

```
#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 [0]:

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

0.0

In [0]:

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

0

In [0]:

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

In [0]:

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

0.02886751345948129

In [0]:

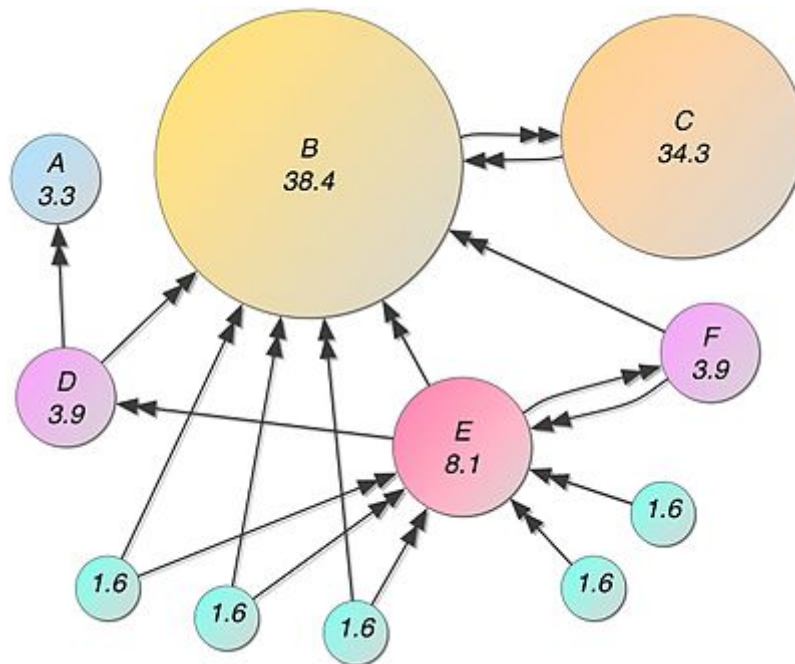
```
print(cosine_for_followers(669354,1635354))
```

0

3. Ranking Measures

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

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



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

3.1 Page Ranking

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

In [0]:

```
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 [0]:

```
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 [0]:

```
#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 two nodes, if nodes have direct path i.e directly connected then we are removing that edge and calculating path.

In [0]:

```
#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 [0]:

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

Out[0]:

```
10
```

In [0]:

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

Out[0]:

-1

4.2 Checking for same community

In [0]:

```
#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 [0]:

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

Out[0]:

0

In [0]:

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

Out[0]:

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 [0]:

```
#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 [0]:

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

Out[0]:

0

In [0]:

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

Out[0]:

0

4.4 Is the person following back:

In [0]:

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

In [0]:

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

Out[0]:

1

In [0]:

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

Out[0]:

0

4.5 Katz Centrality:

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

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

$$x_i = \alpha \sum_j A_{ij} x_j + \beta$$
 where A is the adjacency matrix of the graph G with eigenvalues λ .

The parameter β controls the initial centrality and

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

In [0]:

```
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 [0]:

```
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 [0]:

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

```
0.0007483800935562018
```

4.6 Hits Score

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

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

In [0]:

```
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 [0]:

```
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 [17]:

```
import random
if os.path.isfile('train_after_eda.csv'):
    filename = "train_after_eda.csv"
    # you uncomment this line, if you dont know the lentgh of the file name
    # here we have hardcoded the number of lines as 15100030
    # n_train = sum(1 for line in open(filename)) #number of records in file (excludes
header)
    n_train = 15100028
    s = 100000 #desired sample size
    skip_train = sorted(random.sample(range(1, n_train+1), n_train-s))
#https://stackoverflow.com/a/22259008/4084039
```

In [18]:

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

In [19]:

```
print("Number of rows in the train data file:", n_train)
print("Number of rows we are going to eliminate 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 eliminate in test data are", len(skip_test))
```

Number of rows in the train data file: 15100028

Number of rows we are going to eliminate in train data are 15000028

Number of rows in the test data file: 3775006

Number of rows we are going to eliminate in test data are 3725006

In [0]:

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

Our train matrix size (100002, 3)

Out[0]:

	source_node	destination_node	indicator_link
0	273084	1505602	1
1	832016	1543415	1

In [0]:

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

Our test matrix size (50002, 3)

Out[0]:

	source_node	destination_node	indicator_link
0	848424	784690	1
1	483294	1255532	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 [0]:

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

    #mapping jaccrd followees to train and test data
    df_final_train['jaccard_followees'] = df_final_train.apply(lambda row:
                                                                jaccard_for_followees(row['source_node'],ro
w['destination_node']),axis=1)
    df_final_test['jaccard_followees'] = df_final_test.apply(lambda row:
                                                             jaccard_for_followees(row['source_node'],ro
w['destination_node']),axis=1)

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

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

In [0]:

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

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

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

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

In [0]:

```
if not os.path.isfile('stage1_updated.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=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 [0]:

```
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'],row['destination_node']),axis=1)

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

    #-----
    #mapping shortest path on train
    df_final_train['shortest_path'] = df_final_train.apply(lambda row: compute_shortest_path_length(row['source_node'],row['destination_node']),axis=1)
    #mapping shortest path on test
    df_final_test['shortest_path'] = df_final_test.apply(lambda row: compute_shortest_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=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 [0]:

```

if not os.path.isfile('data/fea_sample/storage_sample_stage3.h5'):

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

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

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

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

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

    df_final_test['hubs_s'] = df_final_test.source_node.apply(lambda x: hits[0].get(x,0
))
    df_final_test['hubs_d'] = df_final_test.destination_node.apply(lambda x: hits[0].ge
t(x,0))
    #=====

    #Hits algorithm score for source and destination in Train and Test
    #if anything not there in train graph then adding 0
    df_final_train['authorities_s'] = df_final_train.source_node.apply(lambda x: hits[1
].get(x,0))
    df_final_train['authorities_d'] = df_final_train.destination_node.apply(lambda x: h
its[1].get(x,0))

    df_final_test['authorities_s'] = df_final_test.source_node.apply(lambda x: hits[1].
get(x,0))
    df_final_test['authorities_d'] = df_final_test.destination_node.apply(lambda x: hit
s[1].get(x,0))
    #=====

    hdf = HDFStore('data/fea_sample/storage_sample_stage3.h5')
    hdf.put('train_df',df_final_train, format='table', data_columns=True)
    hdf.put('test_df',df_final_test, format='table', data_columns=True)
    hdf.close()
else:

```

```
df_final_train = read_hdf('data/fea_sample/storage_sample_stage3.h5', 'train_df', 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 [8]:

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

In [9]:

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

In [11]:

```
len(sadj_col)
```

Out[11]:

```
1780722
```

In [12]:

```
sadj_dict
```

Out[12]:

```
{1: 0,  
 2: 1,  
 3: 2,  
 4: 3,  
 5: 4,  
 6: 5,  
 7: 6,  
 8: 7,  
 9: 8,  
11: 9,  
12: 10,  
13: 11,  
14: 12,  
16: 13,  
18: 14,  
19: 15,  
20: 16,  
21: 17,  
22: 18,  
23: 19,  
24: 20,  
25: 21,  
27: 22,  
28: 23,  
29: 24,  
30: 25,  
31: 26,  
32: 27,  
33: 28,  
34: 29,  
35: 30,  
36: 31,  
37: 32,  
38: 33,  
39: 34,  
40: 35,  
42: 36,  
43: 37,  
44: 38,  
45: 39,  
46: 40,  
47: 41,  
48: 42,  
49: 43,  
50: 44,  
51: 45,  
52: 46,  
53: 47,  
54: 48,  
55: 49,  
56: 50,  
57: 51,  
58: 52,  
59: 53,  
60: 54,  
61: 55,  
62: 56,  
63: 57,  
64: 58,
```

65: 59,
66: 60,
67: 61,
68: 62,
70: 63,
71: 64,
72: 65,
73: 66,
74: 67,
75: 68,
76: 69,
77: 70,
78: 71,
79: 72,
80: 73,
81: 74,
82: 75,
83: 76,
84: 77,
85: 78,
86: 79,
87: 80,
88: 81,
89: 82,
90: 83,
91: 84,
92: 85,
94: 86,
95: 87,
96: 88,
97: 89,
98: 90,
99: 91,
100: 92,
101: 93,
102: 94,
103: 95,
104: 96,
105: 97,
106: 98,
107: 99,
108: 100,
109: 101,
110: 102,
111: 103,
112: 104,
113: 105,
114: 106,
115: 107,
117: 108,
118: 109,
119: 110,
120: 111,
121: 112,
122: 113,
123: 114,
124: 115,
125: 116,
126: 117,
127: 118,
128: 119,

129: 120,
130: 121,
131: 122,
132: 123,
133: 124,
134: 125,
135: 126,
136: 127,
137: 128,
138: 129,
139: 130,
140: 131,
142: 132,
143: 133,
144: 134,
145: 135,
146: 136,
148: 137,
149: 138,
150: 139,
151: 140,
152: 141,
153: 142,
154: 143,
155: 144,
156: 145,
157: 146,
158: 147,
159: 148,
160: 149,
161: 150,
162: 151,
163: 152,
164: 153,
165: 154,
166: 155,
167: 156,
168: 157,
169: 158,
170: 159,
171: 160,
172: 161,
173: 162,
174: 163,
175: 164,
176: 165,
177: 166,
178: 167,
179: 168,
180: 169,
181: 170,
182: 171,
183: 172,
184: 173,
185: 174,
186: 175,
187: 176,
188: 177,
189: 178,
190: 179,
191: 180,

192: 181,
193: 182,
194: 183,
195: 184,
196: 185,
197: 186,
198: 187,
199: 188,
200: 189,
202: 190,
203: 191,
204: 192,
205: 193,
206: 194,
207: 195,
208: 196,
209: 197,
210: 198,
211: 199,
212: 200,
213: 201,
214: 202,
215: 203,
216: 204,
217: 205,
218: 206,
219: 207,
220: 208,
221: 209,
222: 210,
223: 211,
224: 212,
225: 213,
226: 214,
227: 215,
228: 216,
229: 217,
230: 218,
231: 219,
232: 220,
233: 221,
234: 222,
235: 223,
236: 224,
237: 225,
238: 226,
240: 227,
241: 228,
242: 229,
243: 230,
244: 231,
245: 232,
246: 233,
247: 234,
248: 235,
249: 236,
250: 237,
251: 238,
252: 239,
253: 240,
254: 241,

255: 242,
256: 243,
258: 244,
259: 245,
260: 246,
261: 247,
262: 248,
263: 249,
264: 250,
265: 251,
266: 252,
267: 253,
268: 254,
270: 255,
271: 256,
272: 257,
273: 258,
274: 259,
275: 260,
276: 261,
277: 262,
278: 263,
279: 264,
280: 265,
281: 266,
282: 267,
283: 268,
284: 269,
285: 270,
286: 271,
287: 272,
288: 273,
289: 274,
290: 275,
291: 276,
292: 277,
293: 278,
294: 279,
295: 280,
296: 281,
297: 282,
298: 283,
299: 284,
300: 285,
301: 286,
302: 287,
303: 288,
304: 289,
305: 290,
306: 291,
307: 292,
308: 293,
309: 294,
310: 295,
311: 296,
312: 297,
313: 298,
314: 299,
315: 300,
316: 301,
317: 302,

318: 303,
319: 304,
320: 305,
321: 306,
322: 307,
323: 308,
324: 309,
326: 310,
328: 311,
329: 312,
330: 313,
331: 314,
332: 315,
333: 316,
335: 317,
337: 318,
338: 319,
339: 320,
341: 321,
342: 322,
343: 323,
344: 324,
345: 325,
346: 326,
347: 327,
348: 328,
349: 329,
350: 330,
351: 331,
352: 332,
353: 333,
354: 334,
355: 335,
356: 336,
357: 337,
359: 338,
360: 339,
361: 340,
362: 341,
363: 342,
364: 343,
365: 344,
366: 345,
367: 346,
368: 347,
369: 348,
370: 349,
371: 350,
372: 351,
373: 352,
374: 353,
375: 354,
376: 355,
377: 356,
378: 357,
379: 358,
380: 359,
381: 360,
382: 361,
383: 362,
384: 363,

385: 364,
387: 365,
388: 366,
389: 367,
390: 368,
391: 369,
392: 370,
393: 371,
394: 372,
395: 373,
396: 374,
398: 375,
399: 376,
400: 377,
401: 378,
403: 379,
404: 380,
405: 381,
406: 382,
407: 383,
408: 384,
409: 385,
410: 386,
411: 387,
412: 388,
413: 389,
414: 390,
415: 391,
416: 392,
417: 393,
418: 394,
419: 395,
420: 396,
422: 397,
423: 398,
424: 399,
425: 400,
426: 401,
427: 402,
428: 403,
429: 404,
430: 405,
431: 406,
432: 407,
433: 408,
436: 409,
437: 410,
438: 411,
439: 412,
440: 413,
441: 414,
442: 415,
443: 416,
444: 417,
445: 418,
446: 419,
447: 420,
448: 421,
449: 422,
450: 423,
451: 424,

452: 425,
453: 426,
455: 427,
456: 428,
457: 429,
458: 430,
459: 431,
460: 432,
461: 433,
462: 434,
463: 435,
464: 436,
465: 437,
466: 438,
467: 439,
468: 440,
470: 441,
472: 442,
473: 443,
474: 444,
475: 445,
476: 446,
477: 447,
478: 448,
479: 449,
480: 450,
482: 451,
483: 452,
484: 453,
485: 454,
486: 455,
487: 456,
488: 457,
489: 458,
490: 459,
491: 460,
492: 461,
493: 462,
495: 463,
496: 464,
497: 465,
498: 466,
499: 467,
500: 468,
501: 469,
502: 470,
503: 471,
504: 472,
505: 473,
506: 474,
507: 475,
508: 476,
509: 477,
510: 478,
511: 479,
512: 480,
513: 481,
514: 482,
515: 483,
516: 484,
517: 485,

518: 486,
519: 487,
520: 488,
521: 489,
522: 490,
523: 491,
524: 492,
525: 493,
526: 494,
527: 495,
528: 496,
529: 497,
530: 498,
531: 499,
532: 500,
534: 501,
535: 502,
536: 503,
537: 504,
538: 505,
539: 506,
540: 507,
541: 508,
542: 509,
543: 510,
544: 511,
546: 512,
547: 513,
548: 514,
549: 515,
550: 516,
551: 517,
552: 518,
554: 519,
555: 520,
556: 521,
557: 522,
558: 523,
560: 524,
561: 525,
562: 526,
563: 527,
564: 528,
565: 529,
566: 530,
567: 531,
568: 532,
569: 533,
570: 534,
571: 535,
572: 536,
573: 537,
574: 538,
575: 539,
576: 540,
577: 541,
578: 542,
579: 543,
580: 544,
581: 545,
582: 546,

583: 547,
584: 548,
585: 549,
586: 550,
587: 551,
588: 552,
589: 553,
590: 554,
591: 555,
593: 556,
594: 557,
595: 558,
596: 559,
598: 560,
599: 561,
600: 562,
601: 563,
602: 564,
603: 565,
605: 566,
606: 567,
607: 568,
608: 569,
609: 570,
610: 571,
611: 572,
612: 573,
613: 574,
614: 575,
615: 576,
616: 577,
617: 578,
618: 579,
619: 580,
620: 581,
621: 582,
622: 583,
623: 584,
624: 585,
625: 586,
626: 587,
627: 588,
628: 589,
629: 590,
630: 591,
631: 592,
632: 593,
633: 594,
634: 595,
636: 596,
637: 597,
638: 598,
639: 599,
640: 600,
641: 601,
642: 602,
643: 603,
644: 604,
645: 605,
646: 606,
647: 607,

648: 608,
649: 609,
650: 610,
651: 611,
652: 612,
653: 613,
654: 614,
655: 615,
656: 616,
657: 617,
658: 618,
659: 619,
660: 620,
661: 621,
662: 622,
663: 623,
664: 624,
665: 625,
666: 626,
667: 627,
668: 628,
669: 629,
670: 630,
671: 631,
672: 632,
674: 633,
675: 634,
676: 635,
677: 636,
678: 637,
679: 638,
680: 639,
681: 640,
682: 641,
683: 642,
684: 643,
685: 644,
686: 645,
687: 646,
688: 647,
689: 648,
690: 649,
691: 650,
692: 651,
693: 652,
694: 653,
695: 654,
696: 655,
697: 656,
698: 657,
699: 658,
700: 659,
701: 660,
702: 661,
703: 662,
704: 663,
705: 664,
706: 665,
708: 666,
709: 667,
710: 668,

711: 669,
713: 670,
714: 671,
715: 672,
716: 673,
717: 674,
718: 675,
719: 676,
720: 677,
721: 678,
722: 679,
723: 680,
724: 681,
725: 682,
726: 683,
727: 684,
728: 685,
729: 686,
730: 687,
731: 688,
732: 689,
733: 690,
734: 691,
735: 692,
736: 693,
737: 694,
738: 695,
739: 696,
740: 697,
741: 698,
742: 699,
744: 700,
745: 701,
746: 702,
747: 703,
748: 704,
749: 705,
750: 706,
751: 707,
752: 708,
753: 709,
754: 710,
755: 711,
756: 712,
757: 713,
758: 714,
759: 715,
760: 716,
761: 717,
762: 718,
763: 719,
764: 720,
765: 721,
766: 722,
767: 723,
768: 724,
770: 725,
771: 726,
772: 727,
773: 728,
775: 729,

776: 730,
778: 731,
779: 732,
781: 733,
782: 734,
783: 735,
784: 736,
785: 737,
786: 738,
787: 739,
788: 740,
789: 741,
790: 742,
791: 743,
792: 744,
793: 745,
794: 746,
795: 747,
796: 748,
797: 749,
798: 750,
799: 751,
800: 752,
802: 753,
803: 754,
804: 755,
805: 756,
806: 757,
807: 758,
808: 759,
809: 760,
810: 761,
811: 762,
812: 763,
813: 764,
814: 765,
815: 766,
816: 767,
817: 768,
818: 769,
819: 770,
820: 771,
821: 772,
822: 773,
823: 774,
824: 775,
825: 776,
826: 777,
827: 778,
828: 779,
829: 780,
831: 781,
832: 782,
833: 783,
835: 784,
836: 785,
837: 786,
838: 787,
839: 788,
840: 789,
841: 790,

842: 791,
843: 792,
844: 793,
845: 794,
846: 795,
847: 796,
848: 797,
849: 798,
850: 799,
851: 800,
852: 801,
853: 802,
854: 803,
855: 804,
856: 805,
857: 806,
858: 807,
859: 808,
860: 809,
861: 810,
862: 811,
863: 812,
864: 813,
866: 814,
867: 815,
868: 816,
869: 817,
870: 818,
871: 819,
873: 820,
874: 821,
875: 822,
876: 823,
877: 824,
878: 825,
879: 826,
880: 827,
881: 828,
882: 829,
883: 830,
884: 831,
885: 832,
886: 833,
887: 834,
888: 835,
889: 836,
890: 837,
892: 838,
893: 839,
894: 840,
895: 841,
896: 842,
897: 843,
898: 844,
899: 845,
900: 846,
901: 847,
902: 848,
903: 849,
904: 850,
905: 851,

906: 852,
907: 853,
908: 854,
909: 855,
910: 856,
911: 857,
912: 858,
913: 859,
914: 860,
915: 861,
916: 862,
917: 863,
918: 864,
920: 865,
921: 866,
922: 867,
923: 868,
924: 869,
925: 870,
926: 871,
927: 872,
928: 873,
929: 874,
930: 875,
931: 876,
932: 877,
933: 878,
934: 879,
935: 880,
936: 881,
937: 882,
938: 883,
939: 884,
940: 885,
941: 886,
942: 887,
943: 888,
944: 889,
945: 890,
946: 891,
947: 892,
948: 893,
949: 894,
950: 895,
951: 896,
952: 897,
953: 898,
954: 899,
955: 900,
956: 901,
957: 902,
958: 903,
960: 904,
961: 905,
962: 906,
963: 907,
964: 908,
965: 909,
966: 910,
967: 911,
968: 912,

969: 913,
970: 914,
971: 915,
972: 916,
973: 917,
974: 918,
975: 919,
976: 920,
977: 921,
978: 922,
979: 923,
980: 924,
981: 925,
982: 926,
983: 927,
984: 928,
985: 929,
986: 930,
987: 931,
988: 932,
989: 933,
990: 934,
991: 935,
993: 936,
994: 937,
995: 938,
996: 939,
997: 940,
998: 941,
999: 942,
1000: 943,
1001: 944,
1002: 945,
1004: 946,
1005: 947,
1006: 948,
1007: 949,
1008: 950,
1009: 951,
1010: 952,
1011: 953,
1012: 954,
1013: 955,
1014: 956,
1015: 957,
1016: 958,
1017: 959,
1018: 960,
1019: 961,
1020: 962,
1021: 963,
1022: 964,
1023: 965,
1024: 966,
1025: 967,
1026: 968,
1027: 969,
1028: 970,
1029: 971,
1030: 972,
1031: 973,

```
1032: 974,  
1033: 975,  
1034: 976,  
1035: 977,  
1036: 978,  
1037: 979,  
1039: 980,  
1040: 981,  
1041: 982,  
1042: 983,  
1043: 984,  
1044: 985,  
1045: 986,  
1046: 987,  
1047: 988,  
1048: 989,  
1049: 990,  
1050: 991,  
1051: 992,  
1052: 993,  
1053: 994,  
1054: 995,  
1055: 996,  
1056: 997,  
1057: 998,  
1058: 999,  
...}
```

In [0]:

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

In [0]:

```
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 [0]:

```

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 [0]:

```

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

```


In [2]:

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

In [3]:

```
df_final_train.shape
```

Out[3]:

(100002, 54)

In [4]:

```
df_final_test.shape
```

Out[4]:

(50002, 54)

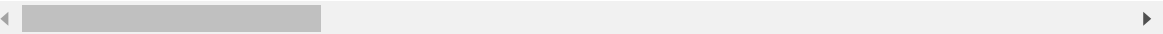
In [5]:

```
df_final_train.head(5)
```

Out[5]:

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

5 rows × 54 columns



In [6]:

```
df_final_train.columns
```

Out[6]:

```
Index(['source_node', 'destination_node', 'indicator_link',
      'jaccard_followers', 'jaccard_followees', 'cosine_followers',
      'cosine_followees', 'num_followers_s', 'num_followees_s',
      'num_followees_d', 'inter_followers', 'inter_followees', 'adar_inde
x',
      'follows_back', 'same_comp', 'shortest_path', 'weight_in', 'weight_
out',
      'weight_f1', 'weight_f2', 'weight_f3', 'weight_f4', 'page_rank_s',
      'page_rank_d', 'katz_s', 'katz_d', 'hubs_s', 'hubs_d', 'authorities
_s',
      'authorities_d', 'svd_u_s_1', 'svd_u_s_2', 'svd_u_s_3', 'svd_u_s_
4',
      'svd_u_s_5', 'svd_u_s_6', 'svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3',
      'svd_u_d_4', 'svd_u_d_5', 'svd_u_d_6', 'svd_v_s_1', 'svd_v_s_2',
      'svd_v_s_3', 'svd_v_s_4', 'svd_v_s_5', 'svd_v_s_6', 'svd_v_d_1',
      'svd_v_d_2', 'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5', 'svd_v_d_6'],
      dtype='object')
```

In [22]:

```
df_train= df_final_train[['source_node','destination_node']]
```

In [52]:

```
df_test= df_final_test[['source_node','destination_node']]
```

In [23]:

```
df_train.head(2)
```

Out[23]:

	source_node	destination_node
0	273084	1505602
1	832016	1543415

In [53]:

```
df_test.head(2)
```

Out[53]:

	source_node	destination_node
0	848424	784690
1	483294	1255532

<http://be.amazd.com/link-prediction/> (<http://be.amazd.com/link-prediction/>)

In [36]:

```
def compute_features_stage1(df_final):
    num_followers_d=[]
    num_followees_d=[]
    for i,row in df_final.iterrows():

        try:
            d1=set(train_graph.predecessors(row['destination_node']))
            d2=set(train_graph.successors(row['destination_node']))
        except:
            d1 = set()
            d2 = set()

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

    return num_followers_d, num_followees_d
```

In [37]:

```
df_train['num_followers_d'],df_train['num_followees_d']= compute_features_stage1(df_train)
df_test['num_followers_d'],df_test['num_followees_d']= compute_features_stage1(df_test)
```

In [38]:

```
df_train.head(3)
```

Out[38]:

	source_node	destination_node	num_followers_d	num_followees_d
0	273084	1505602	6	8
1	832016	1543415	94	142
2	1325247	760242	28	22

In [55]:

```
df_test.head(2)
```

Out[55]:

	source_node	destination_node	num_followers_d	num_followees_d
0	848424	784690	14	9
1	483294	1255532	17	19

In [39]:

```
df_train.num_followers_d.unique()
```

Out[39]:

```
array([ 6, 94, 28, 11, 1, 9, 5, 3, 13, 2, 7, 16, 14,
        60, 12, 4, 18, 24, 152, 70, 29, 215, 17, 23, 10, 40,
        22, 41, 20, 72, 8, 38, 26, 25, 126, 57, 82, 32, 35,
        45, 86, 149, 34, 36, 15, 220, 77, 155, 109, 33, 39, 54,
        21, 46, 31, 47, 104, 19, 27, 216, 44, 49, 43, 114, 121,
        68, 119, 55, 71, 95, 30, 89, 83, 74, 97, 52, 335, 112,
        173, 79, 107, 51, 80, 50, 115, 37, 65, 93, 96, 85, 42,
        98, 127, 122, 61, 73, 102, 106, 53, 91, 296, 108, 56, 140,
        101, 62, 58, 67, 189, 90, 81, 163, 48, 260, 139, 105, 64,
        100, 176, 88, 144, 75, 182, 148, 113, 92, 99, 169, 116, 124,
        193, 218, 181, 78, 165, 66, 103, 59, 179, 158, 123, 233, 191,
        110, 142, 185, 136, 162, 69, 131, 293, 190, 63, 117, 76, 188,
        143, 196, 111, 84, 151, 134, 171, 141, 120, 118, 129, 183, 253,
        239, 130, 232, 247, 135, 203, 164, 210, 159, 133, 150, 221, 303,
        269, 87, 125, 160, 222, 248, 244, 251, 250, 154, 264, 128, 175,
        146, 230, 217, 245, 184, 138, 226, 167, 243, 170, 137, 207, 265,
        132, 211, 333, 314, 204, 157, 195, 178, 172, 212, 153, 200, 240,
        398, 168, 236, 174, 281, 156, 300, 272, 454, 209, 147, 411, 416,
        219, 161, 198, 271, 199, 177, 208, 180, 254, 270, 305, 213, 299,
        235, 406, 228, 234, 186, 0, 318], dtype=int64)
```

In [56]:

```
df_test.num_followers_d.unique()
```

Out[56]:

```
array([ 14, 17, 10, 37, 27, 0, 15, 6, 13, 23, 1, 2, 4,
        28, 8, 7, 3, 16, 5, 12, 11, 29, 31, 19, 100, 41,
        115, 158, 22, 51, 113, 24, 26, 66, 52, 9, 21, 18, 163,
        32, 33, 40, 125, 101, 60, 78, 20, 131, 86, 34, 42, 30,
        25, 55, 71, 56, 70, 58, 36, 47, 64, 43, 39, 82, 69,
        38, 59, 46, 108, 63, 80, 98, 35, 67, 44, 50, 191, 75,
        53, 76, 176, 95, 90, 54, 179, 73, 180, 196, 243, 85, 45,
        183, 102, 112, 48, 230, 89, 104, 61, 105, 141, 118, 236, 164,
        57, 109, 126, 83, 84, 91, 210, 49, 97, 148, 305, 88, 106,
        155, 454, 72, 79, 81, 149, 94, 114, 174, 129, 77, 103, 151,
        237, 68, 134, 175, 65, 111, 245, 74, 144, 62, 239, 96, 99,
        171, 203, 152, 127, 161, 248, 186, 92, 93, 204, 146, 173, 156,
        160, 136, 124, 330, 299, 132, 120, 260, 188, 293, 234, 116, 117,
        211, 142, 139, 184, 288, 221, 233, 219, 240, 168, 333, 145, 119,
        150, 314, 159, 162, 133, 110, 165, 265, 140, 121, 130, 143, 218,
        223, 324, 107, 87, 281, 189, 178, 228, 128, 235, 135, 209, 154,
        270, 181, 215, 212, 335, 268, 264, 200, 244, 247, 137, 253, 138,
        153, 198, 296, 250, 147, 226, 332, 190, 222, 123, 157, 398, 201,
        406, 122, 182, 208, 251, 318], dtype=int64)
```

In [47]:

```
d_followers_tr= df_train.num_followers_d.values  
type(d_followers_tr)  
df_final_train['num_followers_d']= d_followers_tr
```

Out[47]:

numpy.ndarray

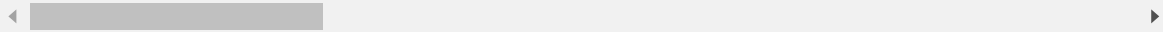
In [51]:

```
df_final_train.head(5)
```

Out[51]:

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

5 rows × 55 columns



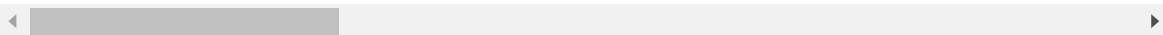
In [58]:

```
d_followers_test= df_test.num_followers_d.values
df_final_test['num_followers_d']= d_followers_test
df_final_test.head(5)
```

Out[58]:

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

5 rows × 55 columns



In [60]:

```
#saving the csv's
df_final_train.to_csv('df_final_train.csv')
df_final_test.to_csv('df_final_test.csv')
```

5.6 Preferential Attachment

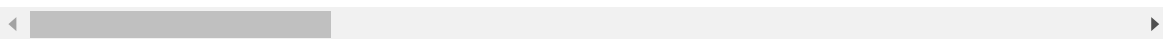
In [61]:

```
# for train data
df_final_train['pa_followers']= (df_final_train.num_followers_s)*(df_final_train.num_followers_d)
df_final_train['pa_followees']= (df_final_train.num_followees_s)*(df_final_train.num_followees_d)
df_final_train.head(3)
```

Out[61]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees
0	273084	1505602	1	0	0.000000
1	832016	1543415	1	0	0.187135
2	1325247	760242	1	0	0.369565

3 rows × 57 columns



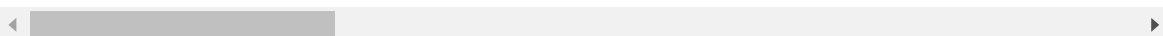
In [62]:

```
# for test data
df_final_test['pa_followers']= (df_final_test.num_followers_s)*(df_final_test.num_followers_d)
df_final_test['pa_followees']= (df_final_test.num_followees_s)*(df_final_test.num_followees_d)
df_final_test.head(3)
```

Out[62]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees
0	848424	784690	1	0	0.0
1	483294	1255532	1	0	0.0
2	626190	1729265	1	0	0.0

3 rows × 57 columns



5.7 svd_dot features (dot product of source & destination node svd features)

In [63]:

```
df_final_train.columns
```

Out[63]:

```
Index(['source_node', 'destination_node', 'indicator_link',
      'jaccard_followers', 'jaccard_followees', 'cosine_followers',
      'cosine_followees', 'num_followers_s', 'num_followees_s',
      'num_followees_d', 'inter_followers', 'inter_followees', 'adar_inde
x',
      'follows_back', 'same_comp', 'shortest_path', 'weight_in', 'weight_
out',
      'weight_f1', 'weight_f2', 'weight_f3', 'weight_f4', 'page_rank_s',
      'page_rank_d', 'katz_s', 'katz_d', 'hubs_s', 'hubs_d', 'authorities
_s',
      'authorities_d', 'svd_u_s_1', 'svd_u_s_2', 'svd_u_s_3', 'svd_u_s_
4',
      'svd_u_s_5', 'svd_u_s_6', 'svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3',
      'svd_u_d_4', 'svd_u_d_5', 'svd_u_d_6', 'svd_v_s_1', 'svd_v_s_2',
      'svd_v_s_3', 'svd_v_s_4', 'svd_v_s_5', 'svd_v_s_6', 'svd_v_d_1',
      'svd_v_d_2', 'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5', 'svd_v_d_6',
      'num_followers_d', 'pa_followers', 'pa_followees'],
      dtype='object')
```

In [80]:

```
#train data

#source nodes
s1= df_final_train.svd_u_s_1.values
s2= df_final_train.svd_u_s_2.values
s3= df_final_train.svd_u_s_3.values
s4= df_final_train.svd_u_s_4.values
s5= df_final_train.svd_u_s_5.values
s6= df_final_train.svd_u_s_6.values
s7= df_final_train.svd_v_s_1.values
s8= df_final_train.svd_v_s_2.values
s9= df_final_train.svd_v_s_3.values
s10= df_final_train.svd_v_s_4.values
s11= df_final_train.svd_v_s_5.values
s12= df_final_train.svd_v_s_6.values

#destination nodes
d1= df_final_train.svd_u_d_1.values
d2= df_final_train.svd_u_d_2.values
d3= df_final_train.svd_u_d_3.values
d4= df_final_train.svd_u_d_4.values
d5= df_final_train.svd_u_d_5.values
d6= df_final_train.svd_u_d_6.values
d7= df_final_train.svd_v_d_1.values
d8= df_final_train.svd_v_d_2.values
d9= df_final_train.svd_v_d_3.values
d10= df_final_train.svd_v_d_4.values
d11= df_final_train.svd_v_d_5.values
d12= df_final_train.svd_v_d_6.values
```


In [81]:

```
# calculating dot product & then assigning the result to the dataframe
svd_dot=[]
for i in range(len(s1)):
    s=[]
    d=[]
    s.append(s1[i])
    s.append(s2[i])
    s.append(s3[i])
    s.append(s4[i])
    s.append(s5[i])
    s.append(s6[i])
    s.append(s7[i])
    s.append(s8[i])
    s.append(s9[i])
    s.append(s10[i])
    s.append(s11[i])
    s.append(s12[i])
    d.append(d1[i])
    d.append(d2[i])
    d.append(d3[i])
    d.append(d4[i])
    d.append(d5[i])
    d.append(d6[i])
    d.append(d7[i])
    d.append(d8[i])
    d.append(d9[i])
    d.append(d10[i])
    d.append(d11[i])
    d.append(d12[i])
    svd_dot.append(np.dot(s,d)) ##https://docs.scipy.org/doc/numpy/reference/generate
d/numpy.dot.html
df_final_train['svd_dot']=svd_dot
df_final_train.head(3)
```

Out[81]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followee
0	273084	1505602	1	0	0.000000
1	832016	1543415	1	0	0.187135
2	1325247	760242	1	0	0.369565

3 rows × 58 columns



In [83]:

```
#test data

#source nodes
s1= df_final_test.svd_u_s_1.values
s2= df_final_test.svd_u_s_2.values
s3= df_final_test.svd_u_s_3.values
s4= df_final_test.svd_u_s_4.values
s5= df_final_test.svd_u_s_5.values
s6= df_final_test.svd_u_s_6.values
s7= df_final_test.svd_v_s_1.values
s8= df_final_test.svd_v_s_2.values
s9= df_final_test.svd_v_s_3.values
s10= df_final_test.svd_v_s_4.values
s11= df_final_test.svd_v_s_5.values
s12= df_final_test.svd_v_s_6.values

#destination nodes
d1= df_final_test.svd_u_d_1.values
d2= df_final_test.svd_u_d_2.values
d3= df_final_test.svd_u_d_3.values
d4= df_final_test.svd_u_d_4.values
d5= df_final_test.svd_u_d_5.values
d6= df_final_test.svd_u_d_6.values
d7= df_final_test.svd_v_d_1.values
d8= df_final_test.svd_v_d_2.values
d9= df_final_test.svd_v_d_3.values
d10= df_final_test.svd_v_d_4.values
d11= df_final_test.svd_v_d_5.values
d12= df_final_test.svd_v_d_6.values
```

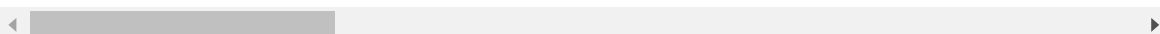
In [84]:

```
# calculating dot product & then assigning the result to the dataframe
svd_dot=[]
for i in range(len(s1)):
    s=[]
    d=[]
    s.append(s1[i])
    s.append(s2[i])
    s.append(s3[i])
    s.append(s4[i])
    s.append(s5[i])
    s.append(s6[i])
    s.append(s7[i])
    s.append(s8[i])
    s.append(s9[i])
    s.append(s10[i])
    s.append(s11[i])
    s.append(s12[i])
    d.append(d1[i])
    d.append(d2[i])
    d.append(d3[i])
    d.append(d4[i])
    d.append(d5[i])
    d.append(d6[i])
    d.append(d7[i])
    d.append(d8[i])
    d.append(d9[i])
    d.append(d10[i])
    d.append(d11[i])
    d.append(d12[i])
    svd_dot.append(np.dot(s,d)) ##https://docs.scipy.org/doc/numpy/reference/generate
d/numpy.dot.html
df_final_test['svd_dot']=svd_dot
df_final_test.head(3)
```

Out[84]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followee
0	848424	784690	1	0	0.0
1	483294	1255532	1	0	0.0
2	626190	1729265	1	0	0.0

3 rows × 58 columns



In [85]:

```
hdf = HDFStore('storage_sample_stage5.h5')
hdf.put('train_df',df_final_train, format='table', data_columns=True)
hdf.put('test_df',df_final_test, format='table', data_columns=True)
hdf.close()
```

In [86]:

```
#saving the csv's
df_final_train.to_csv('df_final_train.csv')
df_final_test.to_csv('df_final_test.csv')
```

ML Models & Results

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 arithmetic operations on arrays
# matplotlib: used to plot graphs
import matplotlib
import matplotlib.pyplot 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
```

1.0 Loading data

In [2]:

```
#reading
from pandas import read_hdf
df_final_train = read_hdf('storage_sample_stage5.h5', 'train_df', mode='r')
df_final_test = read_hdf('storage_sample_stage5.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_followees_s',
      'num_followees_d', 'inter_followers', 'inter_followees', 'adar_inde
x',
      'follows_back', 'same_comp', 'shortest_path', 'weight_in', 'weight_
out',
      'weight_f1', 'weight_f2', 'weight_f3', 'weight_f4', 'page_rank_s',
      'page_rank_d', 'katz_s', 'katz_d', 'hubs_s', 'hubs_d', 'authorities
_s',
      'authorities_d', 'svd_u_s_1', 'svd_u_s_2', 'svd_u_s_3', 'svd_u_s_
4',
      'svd_u_s_5', 'svd_u_s_6', 'svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3',
      'svd_u_d_4', 'svd_u_d_5', 'svd_u_d_6', 'svd_v_s_1', 'svd_v_s_2',
      'svd_v_s_3', 'svd_v_s_4', 'svd_v_s_5', 'svd_v_s_6', 'svd_v_d_1',
      'svd_v_d_2', 'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5', 'svd_v_d_6',
      'num_followers_d', 'pa_followers', 'pa_followees', 'svd_dot'],
      dtype='object')
```

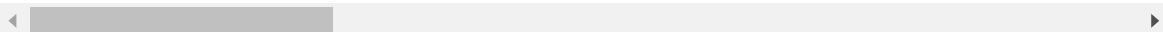
In [4]:

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

Out[4]:

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

2 rows × 58 columns



In [5]:

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

In [6]:

```
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)
```

2.0 Random Forest

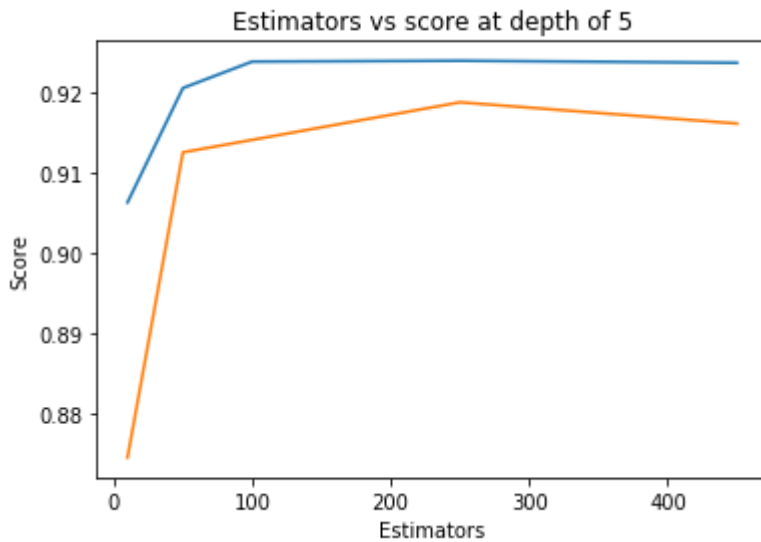
In [0]:

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

Out[0]:

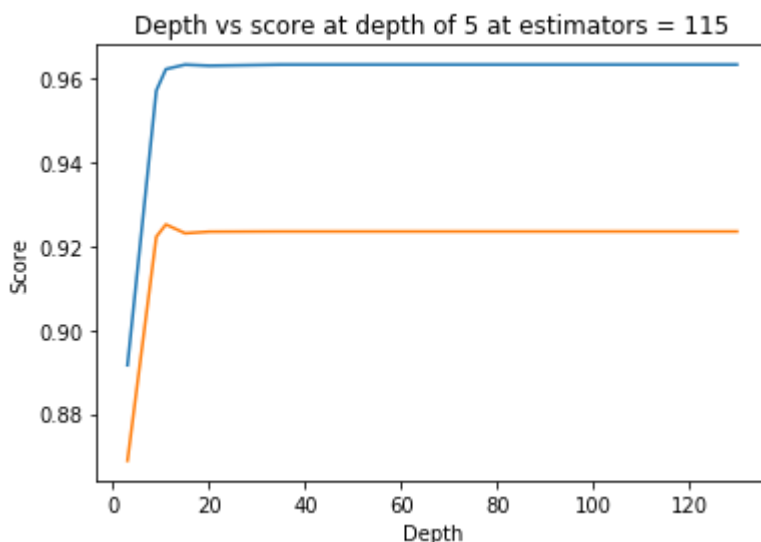
Text(0.5,1,'Estimators vs score at depth of 5')



In [0]:

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

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



In [0]:

```

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)

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 [0.96225043 0.96215493 0.96057081 0.96194015 0.96330005]
mean train scores [0.96294922 0.96266735 0.96115674 0.96263457 0.96430539]

```

In [0]:

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

```

RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                        i',
                        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 [0]:

```

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 [0]:

```

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

```

In [0]:

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

Train f1 score 0.9652533106548414

Test f1 score 0.9241678239279553

In [7]:

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

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

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

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

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

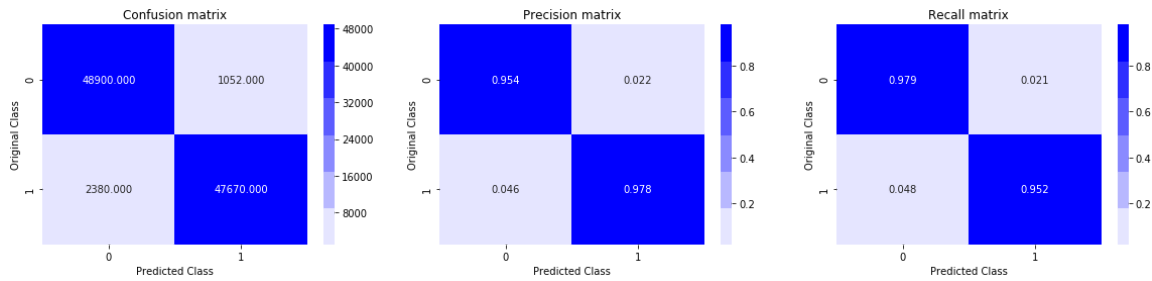
    plt.subplot(1, 3, 3)
    # representing B in heatmap format
    sns.heatmap(A, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=la
bels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()
```

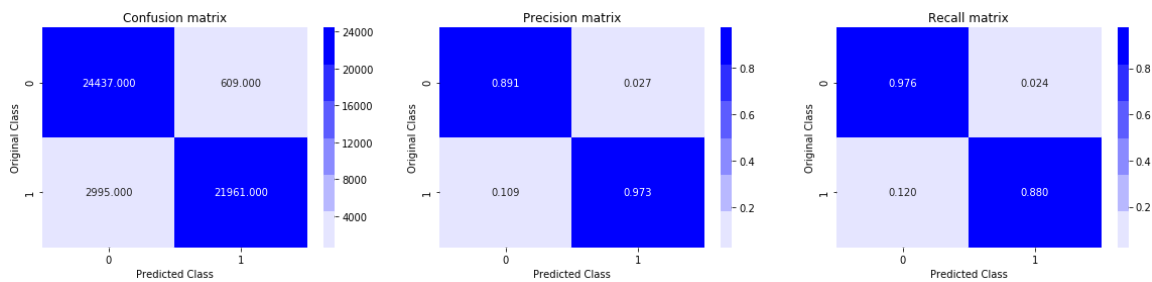
In [0]:

```
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

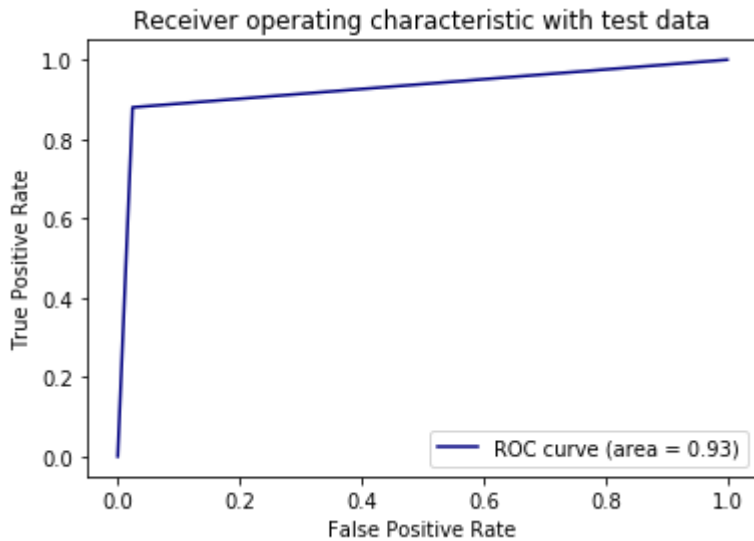


Test confusion_matrix



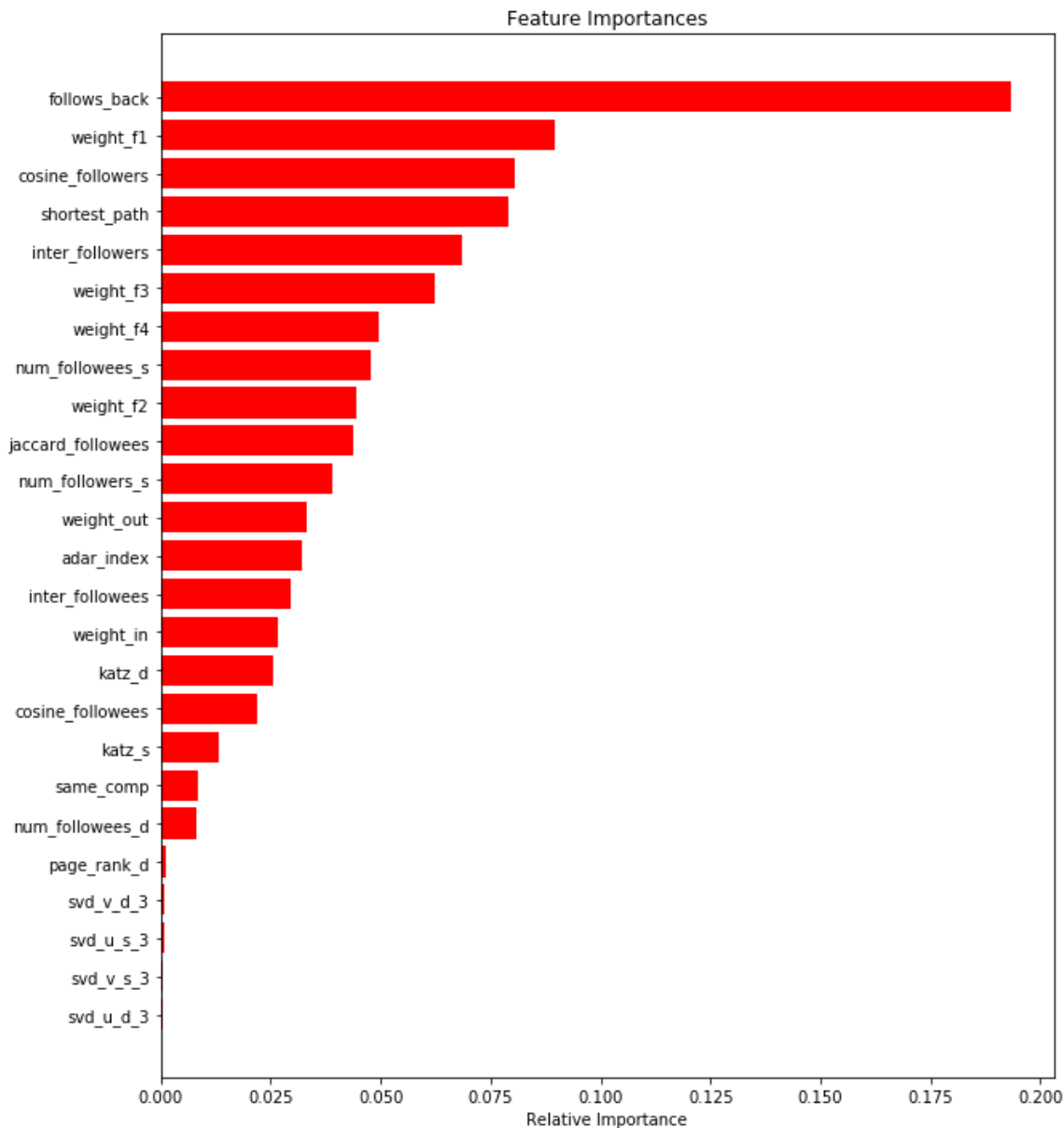
In [0]:

```
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 [0]:

```
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()
```



2.0 XG-Boost with hyperparameter tuning

2.1 Hyperparameter tuning using Randomized search CV

In [9]:

```
#https://dask-ml.readthedocs.io/en/stable/modules/generated/dask_ml.xgboost.XGBClassifier.html
#https://machinelearningmastery.com/develop-first-xgboost-model-python-scikit-learn/

from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import cross_val_score
from xgboost import XGBClassifier

xgb = XGBClassifier()
parameters = {'n_estimators': [10,50,100,250,350,450], 'max_depth': [4,10,12,15,20,35]}
clf1 = RandomizedSearchCV(xgb, parameters, cv=10, scoring='f1', return_train_score=True,
n_jobs=-1)
rs1 = clf1.fit(df_final_train, y_train)
```

2.2 3D-plot

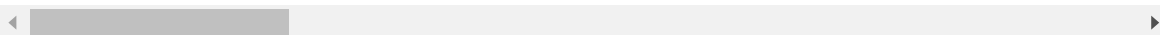
In [10]:

```
df=pd.DataFrame(clf1.cv_results_)
df.head(2)
```

Out[10]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_estimator
0	416.804869	2.574073	0.337476	0.113141	350
1	720.735071	7.928608	0.570546	0.013462	250

2 rows × 32 columns



In [11]:

df.columns

Out[11]:

```
Index(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_time',
      'param_n_estimators', 'param_max_depth', 'params', 'split0_test_score',
      'split1_test_score', 'split2_test_score', 'split3_test_score',
      'split4_test_score', 'split5_test_score', 'split6_test_score',
      'split7_test_score', 'split8_test_score', 'split9_test_score',
      'mean_test_score', 'std_test_score', 'rank_test_score',
      'split0_train_score', 'split1_train_score', 'split2_train_score',
      'split3_train_score', 'split4_train_score', 'split5_train_score',
      'split6_train_score', 'split7_train_score', 'split8_train_score',
      'split9_train_score', 'mean_train_score', 'std_train_score'],
      dtype='object')
```

In [1]:

```
import pandas as pd
df= pd.read_csv('hyp.csv')
```

In [2]:

```
%matplotlib inline
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np

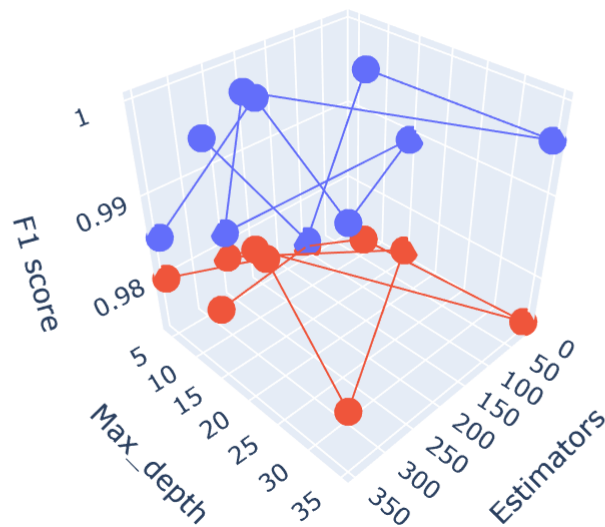
def enable_plotly_in_cell():
    import IPython
    from plotly.offline import init_notebook_mode
    display(IPython.core.display.HTML('''<script src="/static/components/requirejs/require.js"></script>'''))
    init_notebook_mode(connected=False)
```


In [3]:

```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=df['param_n_estimators'],y=df['param_max_depth'],z=df['mean_train_score'], name = 'train')
trace2 = go.Scatter3d(x=df['param_n_estimators'],y=df['param_max_depth'],z=df['mean_test_score'], name = 'Cross validation')
data = [trace1, trace2]
enable_plotly_in_cell()

layout = go.Layout(scene = dict(
    xaxis = dict(title='Estimators'),
    yaxis = dict(title='Max_depth'),
    zaxis = dict(title='F1 score'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```



2.2 Best hyperparameters

In [15]:

```
print(clf1.best_estimator_)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=10,
              min_child_weight=1, missing=None, n_estimators=250, 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)
```

2.3 Applying Best Hyperparameters on train & test data

In [16]:

```
xgb= XGBClassifier(n_estimators= 250 , max_depth= 10)
xgb.fit(df_final_train, y_train)
```

Out[16]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=10,
              min_child_weight=1, missing=None, n_estimators=250, 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 [17]:

```
y_train_pred = xgb.predict(df_final_train)
y_test_pred = xgb.predict(df_final_test)
```

In [18]:

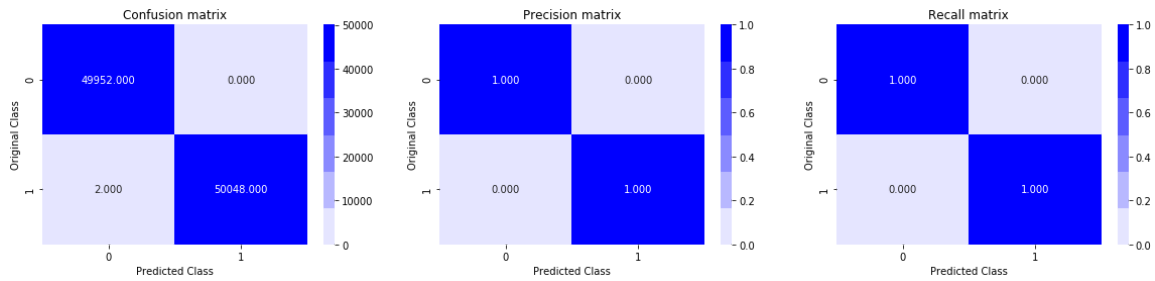
```
from sklearn.metrics import f1_score
print('Train f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))
```

```
Train f1 score 0.9999800195808108
Test f1 score 0.9263565067692503
```

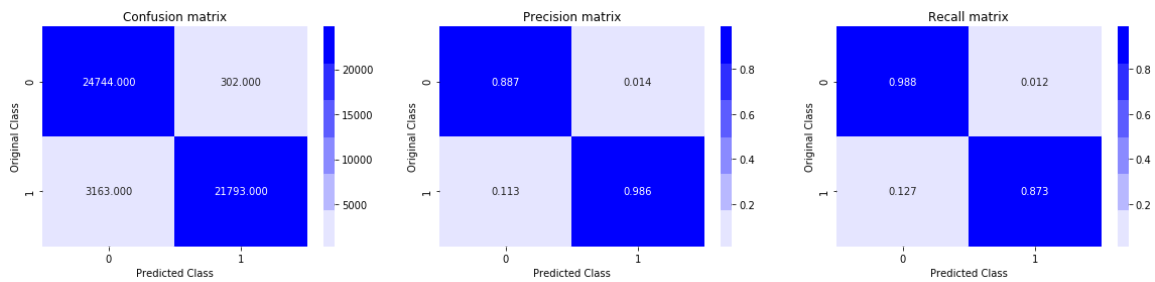
In [22]:

```
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

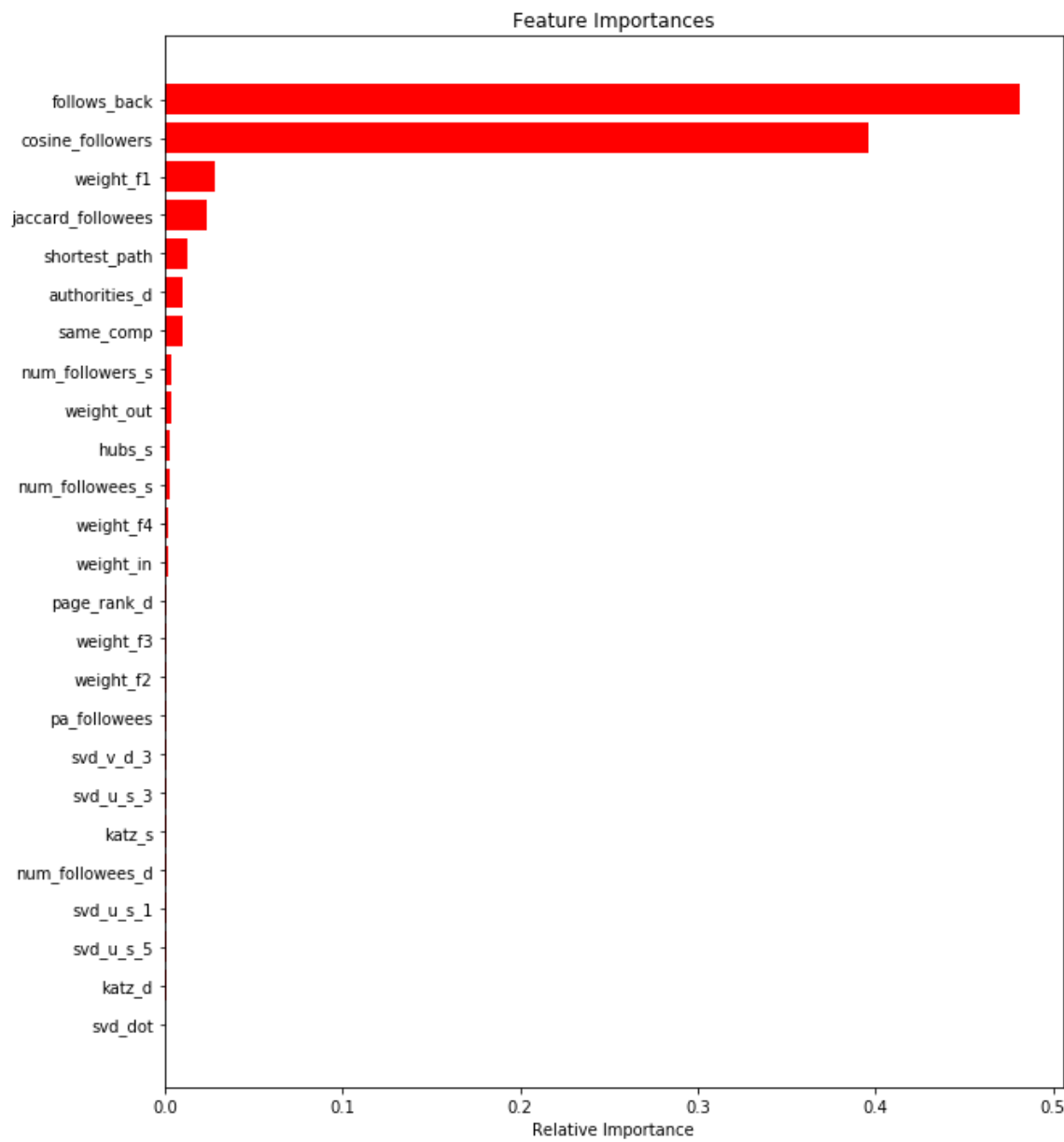


Test confusion_matrix



In [23]:

```
features = df_final_train.columns
importances = xgb.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()
```



3.0 Summary & Conclusion

In [24]:

```
#Ref: http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

x = PrettyTable()
print('Summary')
x.field_names = ["Model", "max_depth", "n_estimators", "Test F1 score"]
x.add_row(["Random Forest", 14, 121, 0.92])
x.add_row(["XGBoost", 10, 250, 0.93])

print(x)
```

Summary

Model	max_depth	n_estimators	Test F1 score
Random Forest	14	121	0.92
XGBoost	10	250	0.93

Conclusion:

- XGBoost model gave a marginally high test F1 score compared to Random forest model (with hyperparameter tuning)
- Regarding feature importance for the XGBoost model, "follows_back" feature was most important & the 2 newly introduced features i.e. "preferential attachment" & "svd_dot" did not turn out to be useful