Social network Graph Link Prediction - Facebook Challenge

Problem statement:

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

Data Overview

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

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

Mapping the problem into supervised learning problem:

- Generated training samples of good and bad links from given directed graph and for each link got some features like no of followers, is he followed back, page rank, katz score, adar index, some svd fetures of adj matrix, some weight features etc. and trained ml model based on these features to predict link.
- Some reference papers and videos :
  - https://www.cs.cornell.edu/home/kleinber/link-pred.pdf (https://www.cs.cornell.edu/home/kleinber/link-pred.pdf)
  - https://www3.nd.edu/~dial/publications/lichtenwalter2010new.pdf (https://www3.nd.edu/~dial/publications/lichtenwalter2010new.pdf)
  - https://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)

Performance metric for supervised learning:

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

Importing Libraries and Pre Processing

In [0]:
```
#Importing Libraries
# please do go through this python notebook:
import warnings
warnings.filterwarnings("ignore")

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

import warnings
import networkx as nx
import pdb
import pickle
from pandas import read_hdf
from pandas import HDFStore,DataFrame
from tqdm import tqdm
from scipy.sparse.linalg import svds, eigs
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
import seaborn as sns#Plots
import xgboost as xgb
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
```

In [0]:
```
# Load the Drive helper and mount
from google.colab import drive
# This will prompt for authorization.
drive.mount('/content/drive')
```

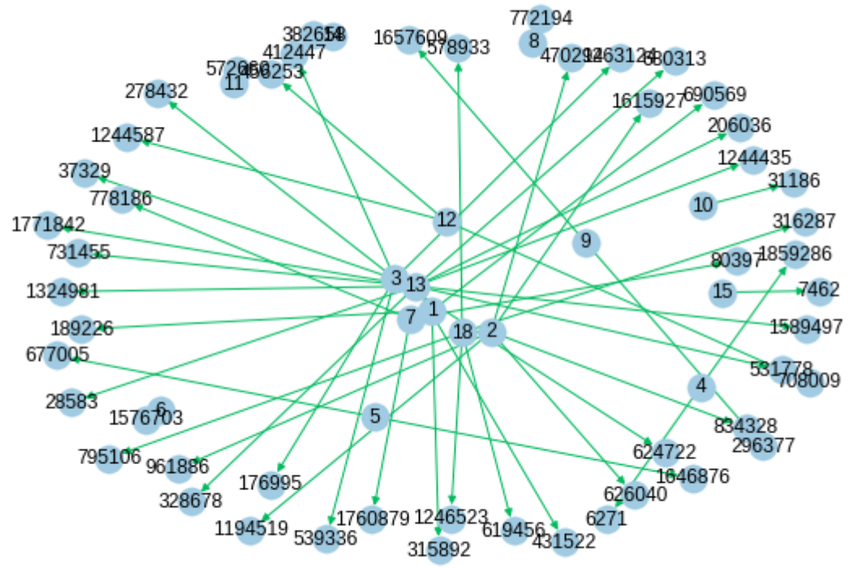
```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code

Enter your authorization code:
··········
Mounted at /content/drive
```

In [0]:
```
%%time
#reading graph with filtered file
g=nx.read_edgelist('/content/drive/My Drive/Colab Notebooks/data_facebook/facebook_train_woheader.csv',delimiter=',',create_using=nx.DiGraph(),nodetype=int)
print(nx.info(g))
```

```
Name:
Type: DiGraph
Number of nodes: 1862220
Number of edges: 9437519
Average in degree:   5.0679
Average out degree:   5.0679
CPU times: user 1min 13s, sys: 4.49 s, total: 1min 17s
Wall time: 1min 18s
```

In [0]:
```
%%time
#Displaying a sub graph with 50 rows

pd.read_csv('/content/drive/My Drive/Colab Notebooks/data_facebook/train.csv', nrows=50).to_csv('/content/drive/My Drive/Colab Notebooks/data_facebook/train_woheader_sample.csv',
header=False,index=False)

subgraph=nx.read_edgelist('/content/drive/My Drive/Colab Notebooks/data_facebook/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
CPU times: user 366 ms, sys: 113 ms, total: 479 ms
Wall time: 946 ms
```



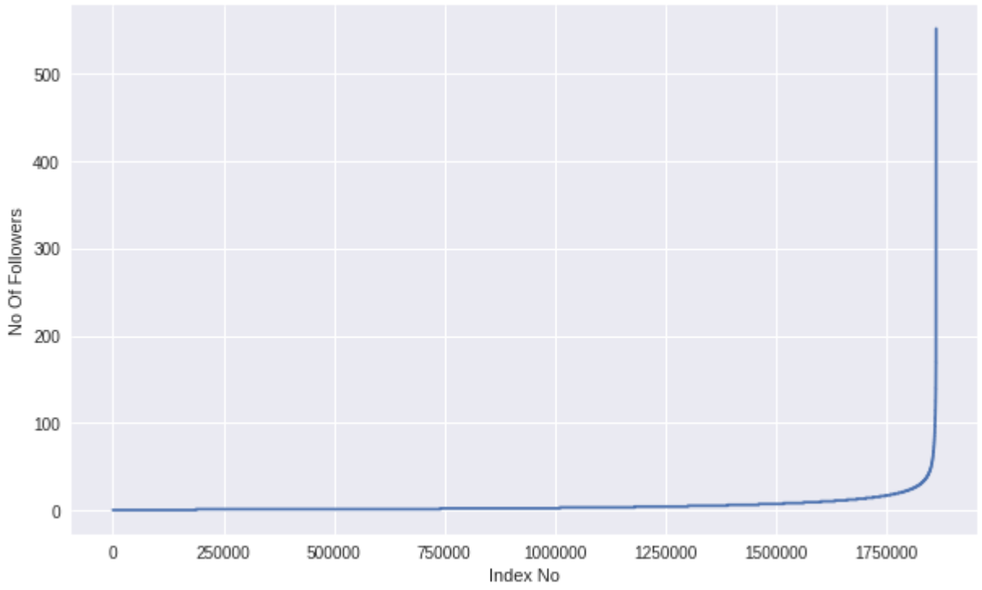
1. Exploratory Data Analysis

In [0]:
```
# No of Unique persons
print("The number of unique persons",len(g.nodes()))
```

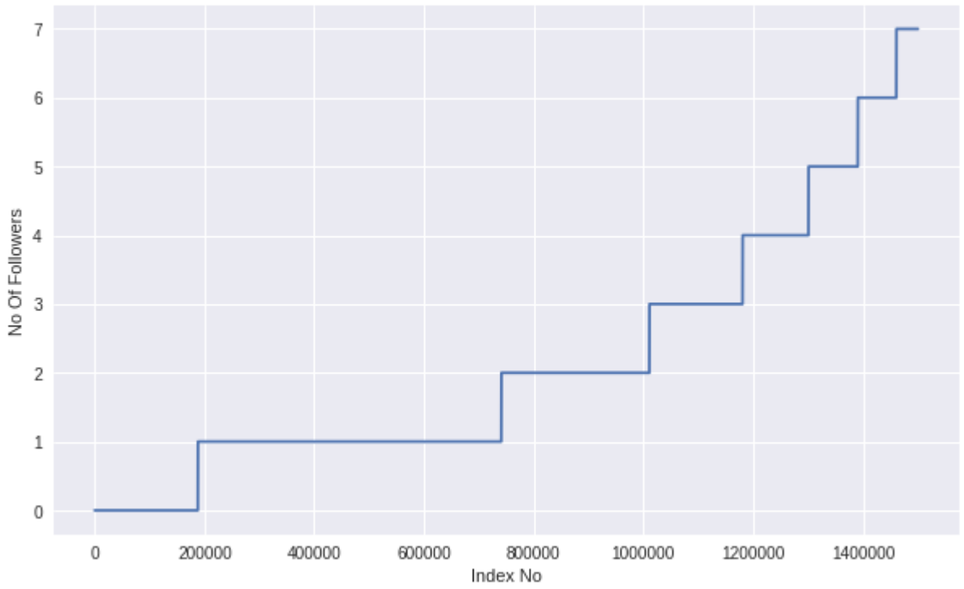
```
The number of unique persons 1862220
```

1.1 No of followers for each person

In [0]:
```
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 [0]:
```
#ploting at some index
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()
#indegree_dist
```



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

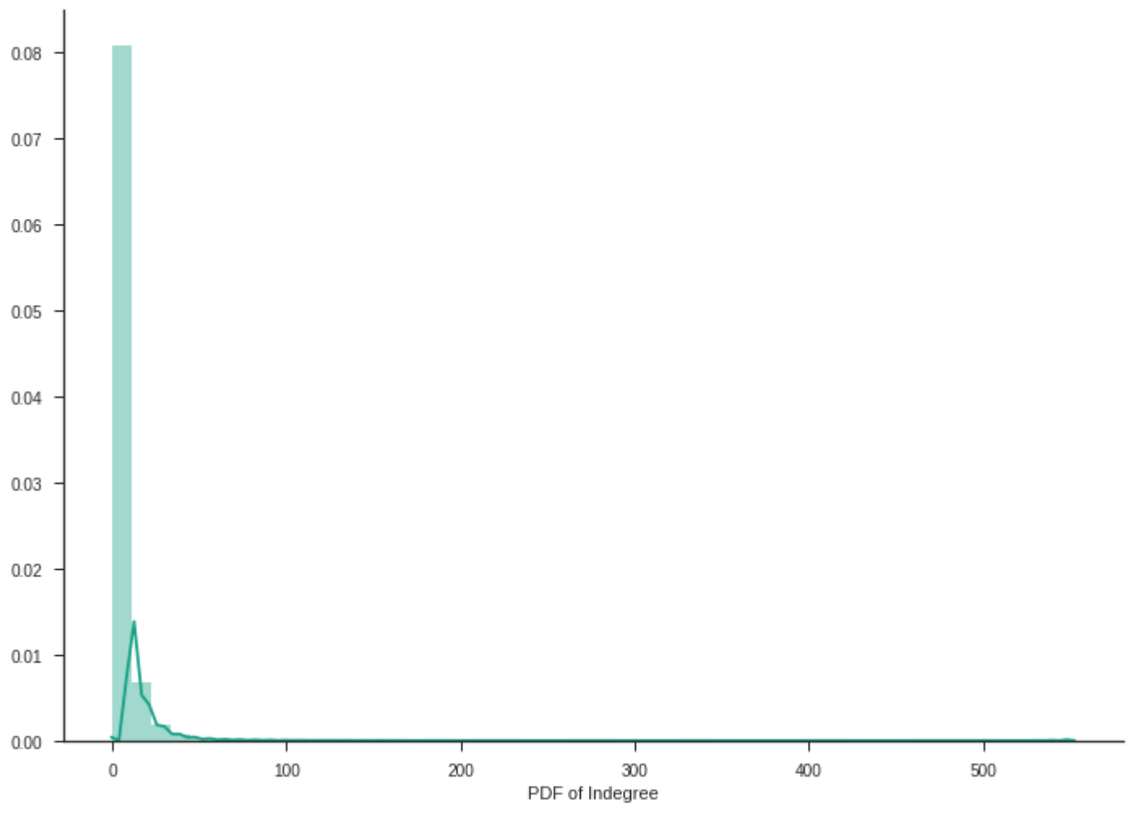
```
90 percentile value is 12.0
91 percentile value is 13.0
92 percentile value is 14.0
93 percentile value is 15.0
94 percentile value is 17.0
95 percentile value is 19.0
96 percentile value is 21.0
97 percentile value is 24.0
98 percentile value is 29.0
99 percentile value is 40.0
100 percentile value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

99% of data having followers of 40 only.

In [0]:
```
### 99-100 percentile
for i in range(10,110,10):
    print(99+(i/100),'percentile value is',np.percentile(indegree_dist,99+(i/100)))
```

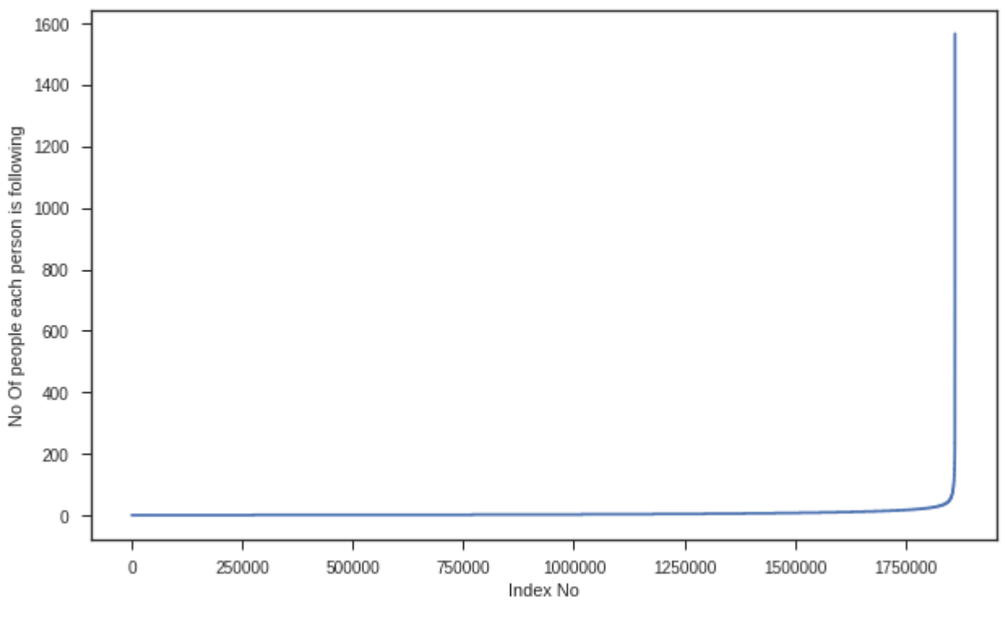
```
99.1 percentile value is 42.0
99.2 percentile value is 44.0
99.3 percentile value is 47.0
99.4 percentile value is 50.0
99.5 percentile value is 55.0
99.6 percentile value is 61.0
99.7 percentile value is 70.0
99.8 percentile value is 84.0
99.9 percentile value is 112.0
100.0 percentile value is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

In [0]:
```
%matplotlib inline
sns.set_style('ticks')
fig, ax = plt.subplots()
fig.set_size_inches(11.7, 8.27)
sns.distplot(indegree_dist, color='#16A085')
plt.xlabel('PDF of Indegree')
sns.despine()
#plt.show()
```

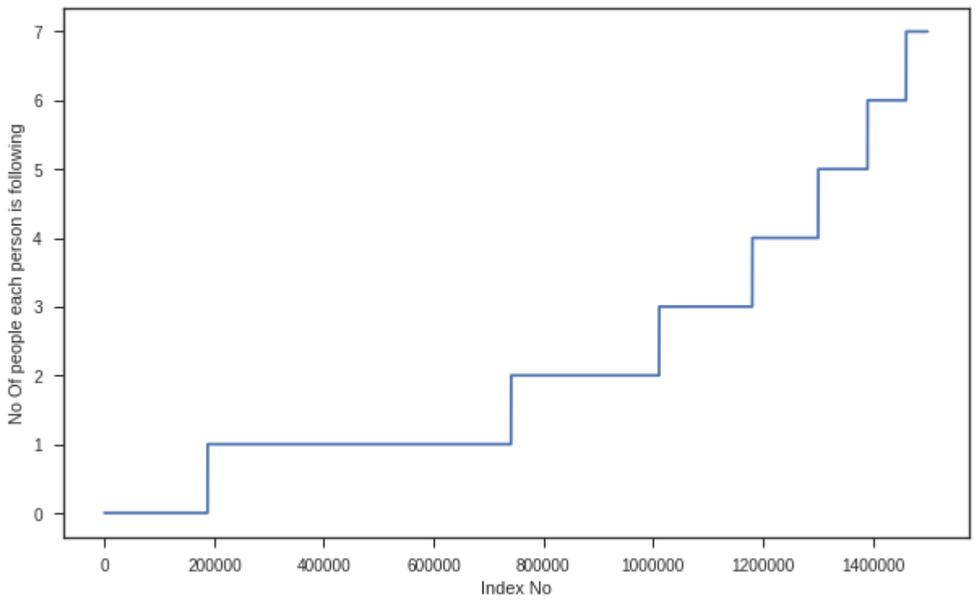


1.2 No of people each person is following

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



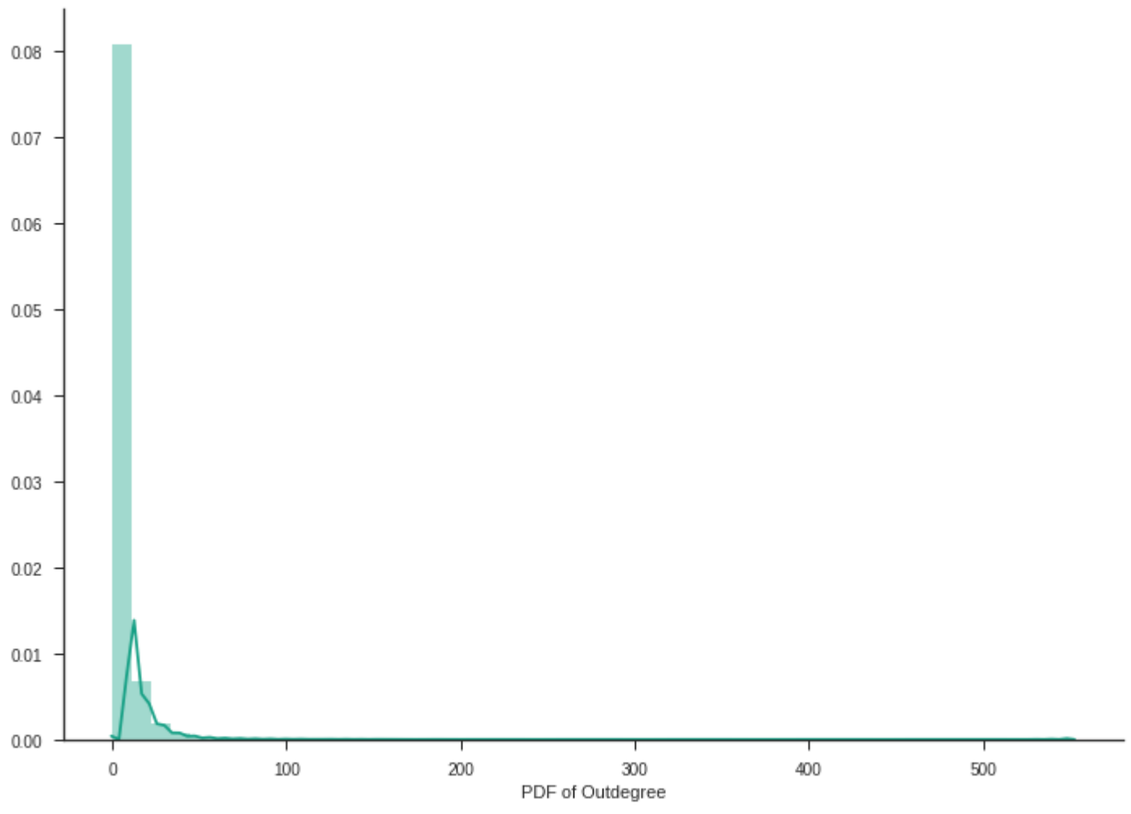
In [0]:
```
### 90-100 percentile
for i in range(0,11):
    print(90+i,'percentile value is',np.percentile(outdegree_dist,90+i))
```

```
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 552.0
```

In [0]:
```
### 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 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 [0]:
```
sns.set_style('ticks')
fig, ax = plt.subplots()
fig.set_size_inches(11.7, 8.27)
sns.distplot(outdegree_dist, color='#16A085')
plt.xlabel('PDF of Outdegree')
sns.despine()
```



In [0]:
```
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 :  188043 and % is : 10.097786512871734
```

In [0]:
```
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
```

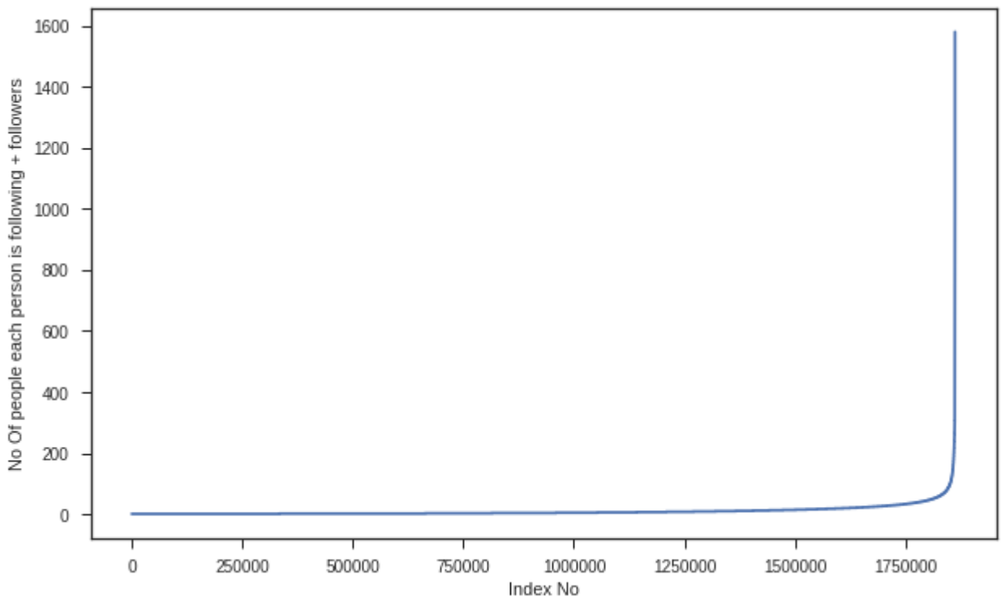
In [0]:
```
count=0
for i in g.nodes():
    if len(list(g.predecessors(i)))==0 :
        if len(list(g.successors(i)))==0:
            count+=1
print('No of persons those are not following anyone and also not having any followers are : ',count)
```

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

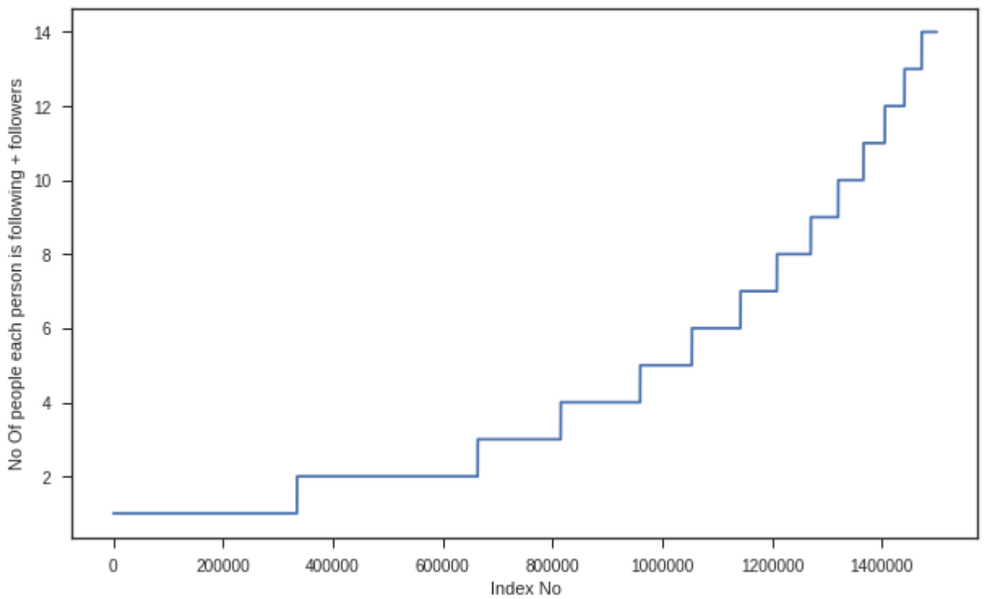
1.3 both followers + following

In [0]:
```
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 [0]:
```
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 [0]:
```
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 [0]:
```
### 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 [0]:
```
### 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 [0]:
```
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 [0]:
```
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 [0]:
```
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 [0]:
```
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
```

1.4 Posing a problem as classification problem

In [0]:
```
%%time
###generating bad edges from given graph
import random
if not os.path.isfile('/content/drive/My Drive/Colab Notebooks/data_facebook/missing_edges_final.p'):
    #getting all set of edges
  r = csv.reader(open('/content/drive/My Drive/Colab Notebooks/data_facebook/train_woheader_sample.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:
  print("b")
  missing_edges = pickle.load(open('/content/drive/My Drive/Colab Notebooks/data_facebook/missing_edges_final.p','rb'))
```

```
b
CPU times: user 2.41 s, sys: 1.61 s, total: 4.02 s
Wall time: 7.24 s
```

In [0]:
```
len(missing_edges)
```

Out[0]:
```
9437519
```

1.5 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 [0]:
```
from sklearn.model_selection import train_test_split
if (not os.path.isfile('/content/drive/My Drive/Colab Notebooks/data_facebook/train_pos_after_eda.csv')) and (not os.path.isfile('/content/drive/My Drive/Colab Notebooks/data_fac
ebook/test_pos_after_eda.csv')):
    #reading total data df
    print("New file")
    df_pos = pd.read_csv('/content/drive/My Drive/Colab Notebooks/data_facebook/facebook_train.csv')
    df_neg = pd.DataFrame(list(missing_edges), columns=['source_node', 'destination_node'])

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

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

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

    #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
    print("Already Existed")
    del missing_edges
```

```
Already Existed
```

In [0]:
```
if (os.path.isfile('/content/drive/My Drive/Colab Notebooks/data_facebook/train_pos_after_eda.csv')) and (os.path.isfile('/content/drive/My Drive/Colab Notebooks/data_facebook/test_pos_after_eda.csv')):
    train_graph=nx.read_edgelist('/content/drive/My Drive/Colab Notebooks/data_facebook/train_pos_after_eda.csv',delimiter=',',create_using=nx.DiGraph(),nodetype=int)
    test_graph=nx.read_edgelist('/content/drive/My Drive/Colab Notebooks/data_facebook/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 [0]:
```
#final train and test data sets
if (not os.path.isfile('/content/drive/My Drive/Colab Notebooks/data_facebook/train_after_eda.csv')) and \
(not os.path.isfile('/content/drive/My Drive/Colab Notebooks/data_facebook/test_after_eda.csv')) and \
(not os.path.isfile('/content/drive/My Drive/Colab Notebooks/data_facebook/train_y.csv')) and \
(not os.path.isfile('/content/drive/My Drive/Colab Notebooks/data_facebook/test_y.csv')) and \
(os.path.isfile('/content/drive/My Drive/Colab Notebooks/data_facebook/train_pos_after_eda.csv')) and \
(os.path.isfile('/content/drive/My Drive/Colab Notebooks/data_facebook/test_pos_after_eda.csv')) and \
(os.path.isfile('/content/drive/My Drive/Colab Notebooks/data_facebook/train_neg_after_eda.csv')) and \
(os.path.isfile('/content/drive/My Drive/Colab Notebooks/data_facebook/test_neg_after_eda.csv')):

    X_train_pos = pd.read_csv('/content/drive/My Drive/Colab Notebooks/data_facebook/train_pos_after_eda.csv', names=['source_node', 'destination_node'])
    X_test_pos = pd.read_csv('/content/drive/My Drive/Colab Notebooks/data_facebook/test_pos_after_eda.csv', names=['source_node', 'destination_node'])
    X_train_neg = pd.read_csv('/content/drive/My Drive/Colab Notebooks/data_facebook/train_neg_after_eda.csv', names=['source_node', 'destination_node'])
    X_test_neg = pd.read_csv('/content/drive/My Drive/Colab Notebooks/data_facebook/test_neg_after_eda.csv', names=['source_node', 'destination_node'])

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

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

    X_train.to_csv('/content/drive/My Drive/Colab Notebooks/data_facebook/train_after_eda.csv',header=False,index=False)
    X_test.to_csv('/content/drive/My Drive/Colab Notebooks/data_facebook/test_after_eda.csv',header=False,index=False)
    pd.DataFrame(y_train.astype(int)).to_csv('/content/drive/My Drive/Colab Notebooks/data_facebook/train_y.csv',header=False,index=False)
    pd.DataFrame(y_test.astype(int)).to_csv('/content/drive/My Drive/Colab Notebooks/data_facebook/test_y.csv',header=False,index=False)

else:

  print("testing")
```

```
============================================================
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
```

Featurization

1. Reading Data

In [0]:
```
%%time
#Reading Data
if os.path.isfile('/content/drive/My Drive/Colab Notebooks/data_facebook/train_pos_after_eda.csv'):
    train_graph=nx.read_edgelist('/content/drive/My Drive/Colab Notebooks/data_facebook/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
CPU times: user 1min 2s, sys: 3.43 s, total: 1min 6s
Wall time: 1min 7s
```

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

In [0]:
```
print(jaccard_for_followers(273084,470294))
```

```
0
```

2.2 Cosine distance

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

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]:
```
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
```

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('/content/drive/My Drive/Colab Notebooks/data_facebook/fea_sample/page_rank.p'):
    pr = nx.pagerank(train_graph, alpha=0.85)
    pickle.dump(pr,open('/content/drive/My Drive/Colab Notebooks/data_facebook/fea_sample/page_rank.p','wb'))
else:
    print("b")
    pr = pickle.load(open('/content/drive/My Drive/Colab Notebooks/data_facebook/fea_sample/page_rank.p','rb'))
```

```
b
```

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

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

4.4 Is persion was 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

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

$$\lambda$$

.

The parameter

$$\beta$$

controls the initial centrality and

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

In [0]:
```
if not os.path.isfile('/content/drive/My Drive/Colab Notebooks/data_facebook/fea_sample/katz.p'):
    katz = nx.katz.katz_centrality(train_graph,alpha=0.005,beta=1)
    pickle.dump(katz,open('/content/drive/My Drive/Colab Notebooks/data_facebook/fea_sample/katz.p','wb'))
else:
    katz = pickle.load(open('/content/drive/My Drive/Colab Notebooks/data_facebook/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))
```

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('/content/drive/My Drive/Colab Notebooks/data_facebook/fea_sample/hits.p'):
    hits = nx.hits(train_graph, max_iter=100, tol=1e-08, nstart=None, normalized=True)
    pickle.dump(hits,open('/content/drive/My Drive/Colab Notebooks/data_facebook/fea_sample/hits.p','wb'))
else:
    hits = pickle.load(open('/content/drive/My Drive/Colab Notebooks/data_facebook/fea_sample/hits.p','rb'))
```

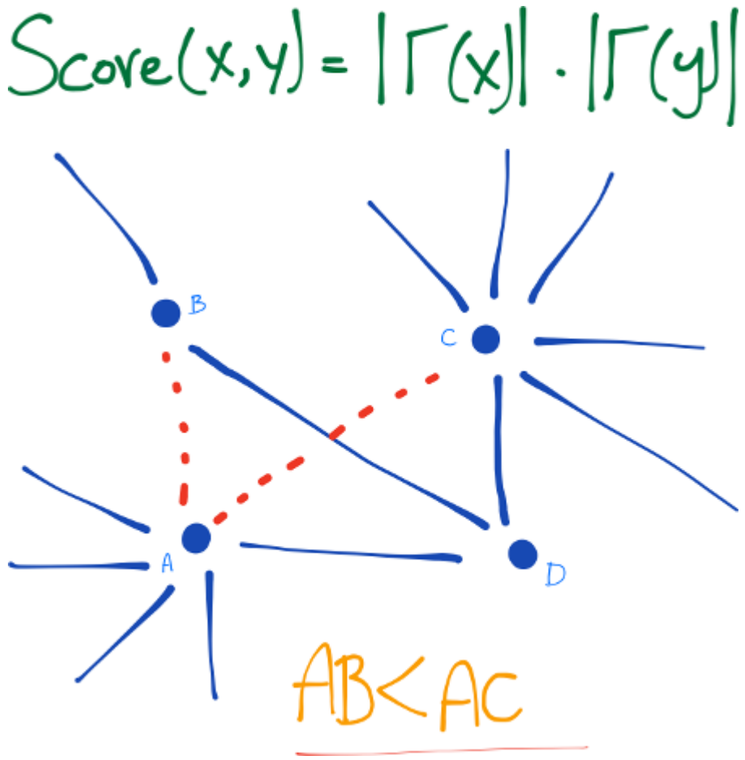
In [0]:
```
print('min',hits[0][min(hits[0], key=hits[0].get)])
print('max',hits[0][max(hits[0], key=hits[0].get)])
print('mean',float(sum(hits[0].values())) / len(hits[0]))
```

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

5. Preferential Attachment (Similarity measures)

Preferential Attachment One well-known concept in social networks is that users with many friends tend to create more connections in the future. This is due to the fact that in some social networks, like in finance, the rich get richer. We estimate how ”rich” our two vertices are by calculating the multiplication between the number of friends (|Γ(x)|) or followers each vertex has. It may be noted that the similarity index does not require any node neighbor information; therefore, this similarity index has the lowest computational complexity.

$$j = |X|.|Y|$$



The link between A and C is more probable than the link between A and B as C have many more neighbors than B

In [0]:
```
#working code
#G= nx.barabasi_albert_graph(20,10)
#nx.draw(G, with_labels=True)
#plt.show()
```

In [0]:
```
#preferential attachment

def preferential_attachment(a,b):
    try:
        if len(set(train_graph.successors(a))) == 0  | len(set(train_graph.successors(b))) == 0:
            #print("a")
            return 0
        #print("d")
        sim = ((math.sqrt(len(set(train_graph.successors(a)))*len(set(train_graph.successors(b))))))
        #print("c")
    except:
        #print("b")
        return 0

    return sim
```

In [0]:
```
preferential_attachment(200,320)
```

Out[0]:
```
3.7416573867739413
```

Dot product of columns Va and Vb in low-rank approximation

5. Featurization

5.1 Reading a sample of Data from both train and test

In [0]:
```
import random
if os.path.isfile('/content/drive/My Drive/Colab Notebooks/data_facebook/train_after_eda.csv'):
    filename = "/content/drive/My Drive/Colab Notebooks/data_facebook/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 [0]:
```
if os.path.isfile('/content/drive/My Drive/Colab Notebooks/data_facebook/test_after_eda.csv'):
    filename = "/content/drive/My Drive/Colab Notebooks/data_facebook/test_after_eda.csv"
    # you uncomment this line, if you dont know the lentgh of the file name
    # here we have hardcoded the number of lines as 3775008
    # n_test = sum(1 for line in open(filename)) #number of records in file (excludes header)
    n_test = 3775006
    s = 50000 #desired sample size
    skip_test = sorted(random.sample(range(1,n_test+1),n_test-s))
    #https://stackoverflow.com/a/22259008/4084039
```

In [0]:
```
print("Number of rows in the train data file:", n_train)
print("Number of rows we are going to elimiate in train data are",len(skip_train))
print("Number of rows in the test data file:", n_test)
print("Number of rows we are going to elimiate in test data are",len(skip_test))
```

```
Number of rows in the train data file: 15100028
Number of rows we are going to elimiate in train data are 15000028
Number of rows in the test data file: 3775006
Number of rows we are going to elimiate in test data are 3725006
```

In [0]:
```
df_final_train = pd.read_csv('/content/drive/My Drive/Colab Notebooks/data_facebook/train_after_eda.csv', skiprows=skip_train, names=['source_node', 'destination_node'])
df_final_train['indicator_link'] = pd.read_csv('/content/drive/My Drive/Colab Notebooks/data_facebook/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	707628	681447	1

In [0]:
```
df_final_test = pd.read_csv('/content/drive/My Drive/Colab Notebooks/data_facebook/test_after_eda.csv', skiprows=skip_test, names=['source_node', 'destination_node'])
df_final_test['indicator_link'] = pd.read_csv('/content/drive/My Drive/Colab Notebooks/data_facebook/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	1446582	1393749	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('/content/drive/My Drive/Colab Notebooks/data_facebook/fea_sample/storage_sample_stage1.h5'):

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

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


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

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

In [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_followees_s, num_followers_d, num_followees_d, inter_followers, inter_followees
```

In [0]:
```
%%time
if not os.path.isfile('/content/drive/My Drive/Colab Notebooks/data_facebook/fea_sample/storage_sample_stage1.h5'):
    print("a")
    df_final_train['num_followers_s'], df_final_train['num_followees_s'], \
    df_final_train['num_followers_d'], 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_followees_s'], \
    df_final_test['num_followers_d'], df_final_test['num_followees_d'], \
    df_final_test['inter_followers'], df_final_test['inter_followees']= compute_features_stage1(df_final_test)

    hdf = HDFStore('/content/drive/My Drive/Colab Notebooks/data_facebook/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:
    print("b")
    df_final_train = read_hdf('/content/drive/My Drive/Colab Notebooks/data_facebook/fea_sample/storage_sample_stage1.h5', 'train_df',mode='r')
    df_final_test = read_hdf('/content/drive/My Drive/Colab Notebooks/data_facebook/fea_sample/storage_sample_stage1.h5', 'test_df',mode='r')
```

```
b
CPU times: user 158 ms, sys: 26 ms, total: 184 ms
Wall time: 221 ms
```

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('/content/drive/My Drive/Colab Notebooks/data_facebook/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 followback or not on train
    df_final_train['follows_back'] = df_final_train.apply(lambda row: follows_back(row['source_node'],row['destination_node']),axis=1)

    #mapping followback or not on test
    df_final_test['follows_back'] = df_final_test.apply(lambda row: follows_back(row['source_node'],row['destination_node']),axis=1)

    #----------------------------------------------------------------------------------------------------
    #mapping same component of wcc or not on train
    df_final_train['same_comp'] = df_final_train.apply(lambda row: belongs_to_same_wcc(row['source_node'],row['destination_node']),axis=1)

    ##mapping same component of wcc or not on train
    df_final_test['same_comp'] = df_final_test.apply(lambda row: belongs_to_same_wcc(row['source_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('/content/drive/My Drive/Colab Notebooks/data_facebook/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:
    print("Already Existed")
    df_final_train = read_hdf('/content/drive/My Drive/Colab Notebooks/data_facebook/fea_sample/storage_sample_stage2.h5', 'train_df',mode='r')
    df_final_test = read_hdf('/content/drive/My Drive/Colab Notebooks/data_facebook/fea_sample/storage_sample_stage2.h5', 'test_df',mode='r')
```

```
Already Existed
```

5.4 Adding new set of features

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

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

Weight Features

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

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

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

In [0]:
```
%%time
#weight for source and destination of each link
Weight_in = {}
Weight_out = {}
for i in  tqdm(train_graph.nodes()):
    s1=set(train_graph.predecessors(i))
    w_in = 1.0/(np.sqrt(1+len(s1)))
    Weight_in[i]=w_in

    s2=set(train_graph.successors(i))
    w_out = 1.0/(np.sqrt(1+len(s2)))
    Weight_out[i]=w_out

#for imputing with mean
mean_weight_in = np.mean(list(Weight_in.values()))
mean_weight_out = np.mean(list(Weight_out.values()))
```

```
100%|██████████| 1780722/1780722 [00:17<00:00, 99947.76it/s]

CPU times: user 17.8 s, sys: 362 ms, total: 18.2 s
Wall time: 18.2 s
```

In [0]:
```
if not os.path.isfile('/content/drive/My Drive/Colab Notebooks/data_facebook/fea_sample/storage_sample_stage3.h5'):
    #mapping to pandas train
    df_final_train['weight_in'] = df_final_train.destination_node.apply(lambda x: Weight_in.get(x,mean_weight_in))
    df_final_train['weight_out'] = df_final_train.source_node.apply(lambda x: Weight_out.get(x,mean_weight_out))

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


    #some features engineerings on the in and out weights
    df_final_train['weight_f1'] = df_final_train.weight_in + df_final_train.weight_out
    df_final_train['weight_f2'] = df_final_train.weight_in * df_final_train.weight_out
    df_final_train['weight_f3'] = (2*df_final_train.weight_in + 1*df_final_train.weight_out)
    df_final_train['weight_f4'] = (1*df_final_train.weight_in + 2*df_final_train.weight_out)

    #some features engineerings on the in and out weights
    df_final_test['weight_f1'] = df_final_test.weight_in + df_final_test.weight_out
    df_final_test['weight_f2'] = df_final_test.weight_in * df_final_test.weight_out
    df_final_test['weight_f3'] = (2*df_final_test.weight_in + 1*df_final_test.weight_out)
    df_final_test['weight_f4'] = (1*df_final_test.weight_in + 2*df_final_test.weight_out)

else:

  print("Already Existed")
```

```
Already Existed
```

In [0]:
```
if not os.path.isfile('/content/drive/My Drive/Colab Notebooks/data_facebook/fea_sample/storage_sample_stage3.h5'):

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

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

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

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

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

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

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

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

    hdf = HDFStore('data/fea_sample/storage_sample_stage3.h5')
    hdf.put('train_df',df_final_train, format='table', data_columns=True)
    hdf.put('test_df',df_final_test, format='table', data_columns=True)
    hdf.close()
else:
    hdf.close()
    df_final_train = read_hdf('/content/drive/My Drive/Colab Notebooks/data_facebook/fea_sample/storage_sample_stage3.h5', 'train_df',mode='r')
    df_final_test = read_hdf('/content/drive/My Drive/Colab Notebooks/data_facebook/fea_sample/storage_sample_stage3.h5', 'test_df',mode='r')
    hdf.close()
```

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

In [0]:
```
#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 [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('/content/drive/My Drive/Colab Notebooks/data_facebook/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('/content/drive/My Drive/Colab Notebooks/data_facebook/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()

else:

  print("Already Existed")
```

```
Already Existed
```

5.6 Adding New Feature preferential_attachment

In [0]:
```
#reading from file
from pandas import read_hdf
df_final_train = read_hdf('/content/drive/My Drive/Colab Notebooks/data_facebook/fea_sample/storage_sample_stage4.h5', 'train_df',mode='r')
df_final_test = read_hdf('/content/drive/My Drive/Colab Notebooks/data_facebook/fea_sample/storage_sample_stage4.h5', 'test_df',mode='r')
```

In [0]:
```
#creating new file
with open('/content/drive/My Drive/Colab Notebooks/data_facebook/storage_sample_stage5.h5', 'w') as f:
  f.write('content')
```

In [0]:
```
#adding new features
df_final_train['preferential_attachment'] = df_final_train.apply(lambda row:preferential_attachment(row['source_node'],row['destination_node']),axis=1)
df_final_test['preferential_attachment'] = df_final_test.apply(lambda row:preferential_attachment(row['source_node'],row['destination_node']),axis=1)


hdf = HDFStore('/content/drive/My Drive/Colab Notebooks/data_facebook/storage_sample_stage5.h5',mode='w')
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]:
```
df_final_train.head(2)
```

Out[0]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	num_followers_s	num_followees_s	num_followees_d	...	svd_v_s_4	svd_v_
0	273084	1505602	1	0	0.000000	0.000000	0.000000	6	15	8	...	1.545075e-13	8.10843 13
1	832016	1543415	1	0	0.187135	0.028382	0.343828	94	61	142	...	1.345726e-02	3.70347 12

2 rows × 55 columns

Adding SVD DOT features (source node * destination node)

In [0]:
```
#Appling SVD DOT in source and destination nodes

df_final_train['svd_u_1_dot'] = df_final_train['svd_u_s_1']*(df_final_train['svd_u_d_1'])
df_final_test['svd_u_1_dot'] = df_final_test['svd_u_s_1']*(df_final_test['svd_u_d_1'])
df_final_train['svd_v_1_dot'] = df_final_train['svd_v_s_1']*(df_final_train['svd_v_d_1'])
df_final_test['svd_v_1_dot'] = df_final_test['svd_v_s_1']*(df_final_test['svd_v_d_1'])

df_final_train['svd_u_2_dot'] = df_final_train['svd_u_s_2']*(df_final_train['svd_u_d_2'])
df_final_test['svd_u_2_dot'] = df_final_test['svd_u_s_2']*(df_final_test['svd_u_d_2'])
df_final_train['svd_v_2_dot'] = df_final_train['svd_v_s_2']*(df_final_train['svd_v_d_2'])
df_final_test['svd_v_2_dot'] = df_final_test['svd_v_s_2']*(df_final_test['svd_v_d_2'])

df_final_train['svd_u_3_dot'] = df_final_train['svd_u_s_3']*(df_final_train['svd_u_d_3'])
df_final_test['svd_u_3_dot'] = df_final_test['svd_u_s_3']*(df_final_test['svd_u_d_3'])
df_final_train['svd_v_3_dot'] = df_final_train['svd_v_s_3']*(df_final_train['svd_v_d_3'])
df_final_test['svd_v_3_dot'] = df_final_test['svd_v_s_3']*(df_final_test['svd_v_d_3'])

df_final_train['svd_u_4_dot'] = df_final_train['svd_u_s_4']*(df_final_train['svd_u_d_4'])
df_final_test['svd_u_4_dot'] = df_final_test['svd_u_s_4']*(df_final_test['svd_u_d_4'])
df_final_train['svd_v_4_dot'] = df_final_train['svd_v_s_4']*(df_final_train['svd_v_d_5'])
df_final_test['svd_v_4_dot'] = df_final_test['svd_v_s_4']*(df_final_test['svd_v_d_5'])

df_final_train['svd_u_5_dot'] = df_final_train['svd_u_s_5']*(df_final_train['svd_u_d_5'])
df_final_test['svd_u_5_dot'] = df_final_test['svd_u_s_5']*(df_final_test['svd_u_d_5'])
df_final_train['svd_v_5_dot'] = df_final_train['svd_v_s_5']*(df_final_train['svd_v_d_5'])
df_final_test['svd_v_5_dot'] = df_final_test['svd_v_s_5']*(df_final_test['svd_v_d_5'])

df_final_train['svd_u_6_dot'] = df_final_train['svd_u_s_6']*(df_final_train['svd_u_d_6'])
df_final_test['svd_u_6_dot'] = df_final_test['svd_u_s_6']*(df_final_test['svd_u_d_6'])
df_final_train['svd_v_6_dot'] = df_final_train['svd_v_s_6']*(df_final_train['svd_v_d_6'])
df_final_test['svd_v_6_dot'] = df_final_test['svd_v_s_6']*(df_final_test['svd_v_d_6'])


#creating new file
hdf = HDFStore('/content/drive/My Drive/Colab Notebooks/data_facebook/storage_sample_stage6.h5',mode='w')
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]:
```
df_final_train.head()
```

Out[0]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	num_followers_s	num_followees_s	num_followees_d	...	svd_u_2_dot	svd_
0	273084	1505602	1	0	0.000000	0.000000	0.000000	6	15	8	...	2.490079e-25	1.835
1	832016	1543415	1	0	0.187135	0.028382	0.343828	94	61	142	...	1.689511e-20	5.429
2	1325247	760242	1	0	0.369565	0.156957	0.566038	28	41	22	...	7.011467e-37	2.092
3	1368400	1006992	1	0	0.000000	0.000000	0.000000	11	5	7	...	5.825985e-24	1.798
4	140165	1708748	1	0	0.000000	0.000000	0.000000	1	11	3	...	2.728218e-26	0.000

5 rows × 67 columns

Implementing Models

RandomForestClassifier

In [0]:
```
#reading
from pandas import read_hdf
df_final_train = read_hdf('/content/drive/My Drive/Colab Notebooks/data_facebook/storage_sample_stage6.h5', 'train_df',mode='r')
df_final_test = read_hdf('/content/drive/My Drive/Colab Notebooks/data_facebook/storage_sample_stage6.h5', 'test_df',mode='r')
```

In [0]:
```
df_final_train.columns
```

Out[0]:
```
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_index',
       'follows_back', 'same_comp', 'shortest_path', 'weight_in', 'weight_out',
       'weight_f1', 'weight_f2', 'weight_f3', 'weight_f4', 'page_rank_s',
       'page_rank_d', 'katz_s', 'katz_d', 'hubs_s', 'hubs_d', 'authorities_s',
       'authorities_d', 'svd_u_s_1', 'svd_u_s_2', 'svd_u_s_3', 'svd_u_s_4',
       'svd_u_s_5', 'svd_u_s_6', 'svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3',
       'svd_u_d_4', 'svd_u_d_5', 'svd_u_d_6', 'svd_v_s_1', 'svd_v_s_2',
       'svd_v_s_3', 'svd_v_s_4', 'svd_v_s_5', 'svd_v_s_6', 'svd_v_d_1',
       'svd_v_d_2', 'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5', 'svd_v_d_6',
       'preferential_attachment', 'svd_u_1_dot', 'svd_v_1_dot', 'svd_u_2_dot',
       'svd_v_2_dot', 'svd_u_3_dot', 'svd_v_3_dot', 'svd_u_4_dot',
       'svd_v_4_dot', 'svd_u_5_dot', 'svd_v_5_dot', 'svd_u_6_dot',
       'svd_v_6_dot'],
      dtype='object')
```

In [0]:
```
y_train = df_final_train.indicator_link
y_test = df_final_test.indicator_link
```

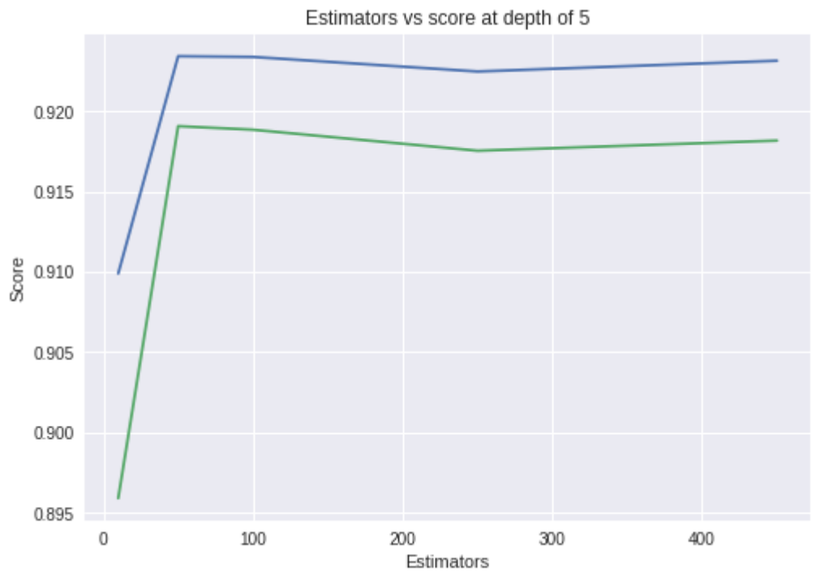
In [0]:
```
#droping source code, destination code and indicator link

df_final_train.drop(['source_node', 'destination_node','indicator_link'],axis=1,inplace=True)
df_final_test.drop(['source_node', 'destination_node','indicator_link'],axis=1,inplace=True)
```

In [0]:
```
df_final_train.columns
```

Out[0]:
```
Index(['jaccard_followers', 'jaccard_followees', 'cosine_followers',
       'cosine_followees', 'num_followers_s', 'num_followees_s',
       'num_followees_d', 'inter_followers', 'inter_followees', 'adar_index',
       'follows_back', 'same_comp', 'shortest_path', 'weight_in', 'weight_out',
       'weight_f1', 'weight_f2', 'weight_f3', 'weight_f4', 'page_rank_s',
       'page_rank_d', 'katz_s', 'katz_d', 'hubs_s', 'hubs_d', 'authorities_s',
       'authorities_d', 'svd_u_s_1', 'svd_u_s_2', 'svd_u_s_3', 'svd_u_s_4',
       'svd_u_s_5', 'svd_u_s_6', 'svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3',
       'svd_u_d_4', 'svd_u_d_5', 'svd_u_d_6', 'svd_v_s_1', 'svd_v_s_2',
       'svd_v_s_3', 'svd_v_s_4', 'svd_v_s_5', 'svd_v_s_6', 'svd_v_d_1',
       'svd_v_d_2', 'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5', 'svd_v_d_6',
       'preferential_attachment', 'svd_u_1_dot', 'svd_v_1_dot', 'svd_u_2_dot',
       'svd_v_2_dot', 'svd_u_3_dot', 'svd_v_3_dot', 'svd_u_4_dot',
       'svd_v_4_dot', 'svd_u_5_dot', 'svd_v_5_dot', 'svd_u_6_dot',
       'svd_v_6_dot'],
      dtype='object')
```

In [0]:
```
%%time
#estimator wise

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)


#ploting graph
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 0.9099031396125584 test Score 0.8959235857230405
Estimators =  50 Train Score 0.9234163581989668 test Score 0.9190653146060338
Estimators =  100 Train Score 0.9233729295633262 test Score 0.9188439889103587
Estimators =  250 Train Score 0.9224650422841834 test Score 0.917535305985205
Estimators =  450 Train Score 0.9231348198325336 test Score 0.918167695343662
CPU times: user 4min 11s, sys: 859 ms, total: 4min 12s
Wall time: 2min 8s
```



In [0]:
```
%%time
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 [0.96141322 0.96087341 0.9596804  0.960717   0.96220727]
mean train scores [0.96214874 0.96141099 0.95976391 0.96118062 0.96303008]
CPU times: user 2min 45s, sys: 5.78 s, total: 2min 51s
Wall time: 20min 12s
```

In [0]:
```
print(rf_random.best_estimator_)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=14, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=28, min_samples_split=111,
            min_weight_fraction_leaf=0.0, n_estimators=121, n_jobs=-1,
            oob_score=False, random_state=25, verbose=0, warm_start=False)
```

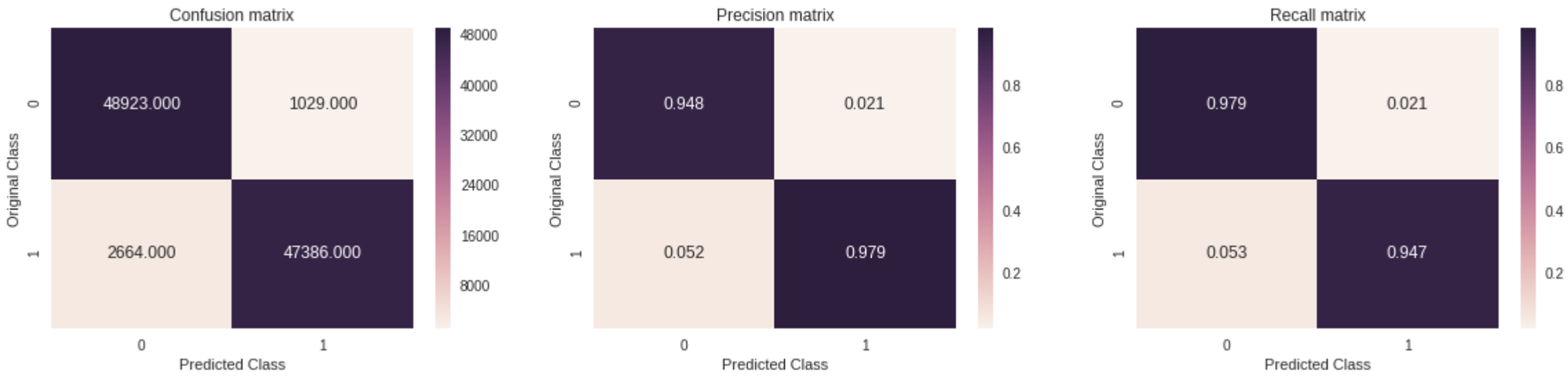
In [0]:
```
%%time
#fitting and predicting through model

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)


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

```
CPU times: user 54.8 s, sys: 193 ms, total: 55 s
Wall time: 28 s
```

In [0]:
```
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

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

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

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

    plt.show()
```

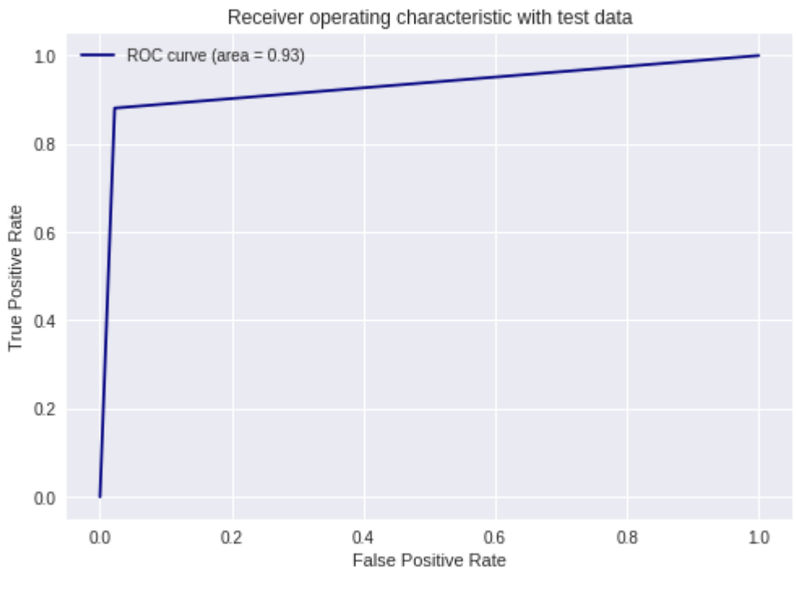
In [0]:
```
print('Train confusion_matrix')
plot_confusion_matrix(y_train,y_train_pred)
print('Test confusion_matrix')
plot_confusion_matrix(y_test,y_test_pred)
```

```
Train confusion_matrix
```

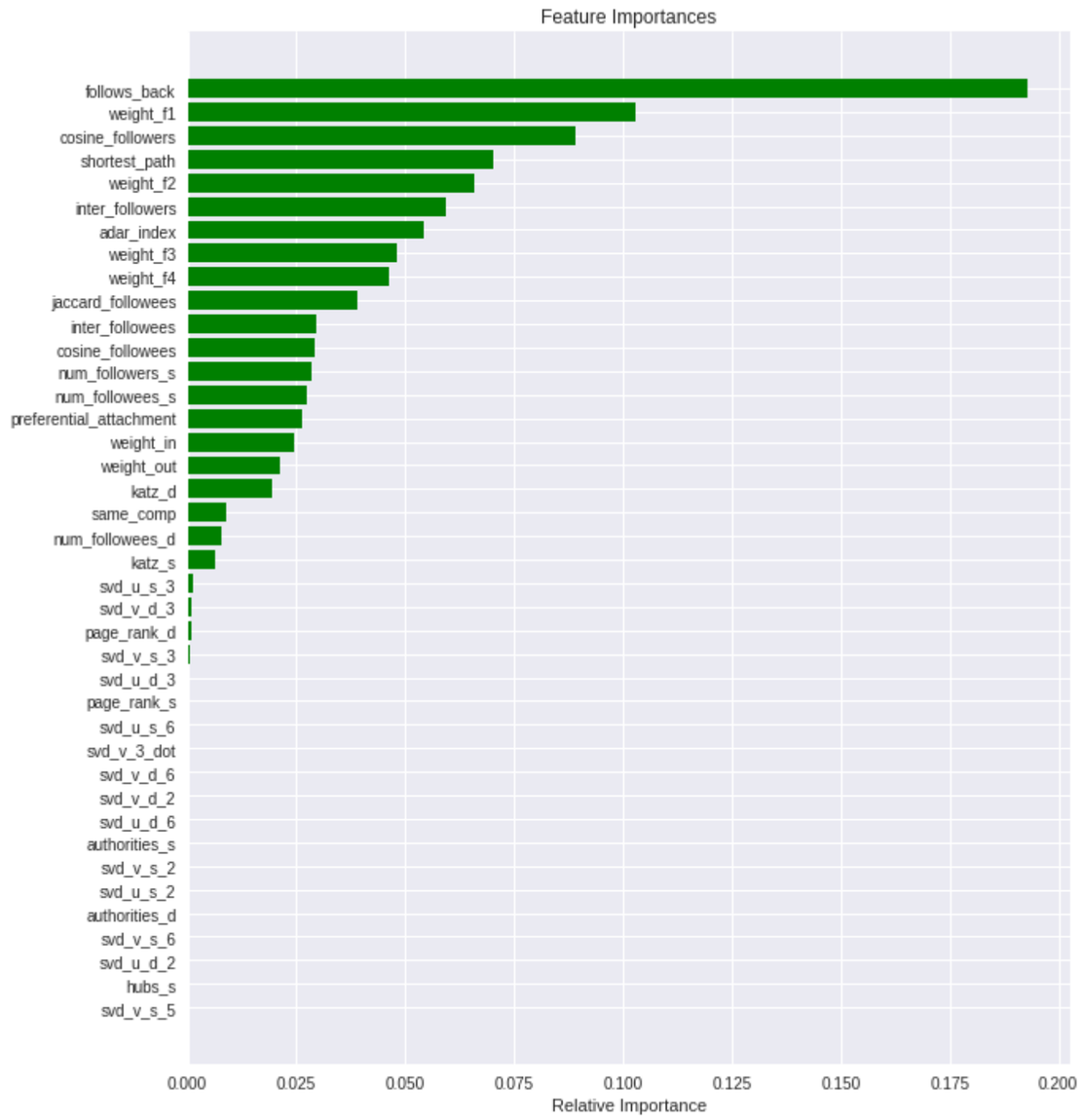



```
Test confusion_matrix
```

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))[-40:]
plt.figure(figsize=(10,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='g', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



XG boost

In [0]:
```
#reading
from pandas import read_hdf
df_final_train = read_hdf('/content/drive/My Drive/Colab Notebooks/data_facebook/storage_sample_stage6.h5', 'train_df',mode='r')
df_final_test = read_hdf('/content/drive/My Drive/Colab Notebooks/data_facebook/storage_sample_stage6.h5', 'test_df',mode='r')
```

In [0]:
```
df_final_train.columns
```

Out[0]:
```
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_index',
       'follows_back', 'same_comp', 'shortest_path', 'weight_in', 'weight_out',
       'weight_f1', 'weight_f2', 'weight_f3', 'weight_f4', 'page_rank_s',
       'page_rank_d', 'katz_s', 'katz_d', 'hubs_s', 'hubs_d', 'authorities_s',
       'authorities_d', 'svd_u_s_1', 'svd_u_s_2', 'svd_u_s_3', 'svd_u_s_4',
       'svd_u_s_5', 'svd_u_s_6', 'svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3',
       'svd_u_d_4', 'svd_u_d_5', 'svd_u_d_6', 'svd_v_s_1', 'svd_v_s_2',
       'svd_v_s_3', 'svd_v_s_4', 'svd_v_s_5', 'svd_v_s_6', 'svd_v_d_1',
       'svd_v_d_2', 'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5', 'svd_v_d_6',
       'preferential_attachment', 'svd_u_1_dot', 'svd_v_1_dot', 'svd_u_2_dot',
       'svd_v_2_dot', 'svd_u_3_dot', 'svd_v_3_dot', 'svd_u_4_dot',
       'svd_v_4_dot', 'svd_u_5_dot', 'svd_v_5_dot', 'svd_u_6_dot',
       'svd_v_6_dot'],
      dtype='object')
```

In [0]:
```
y_train = df_final_train.indicator_link
y_test = df_final_test.indicator_link
```

In [0]:
```
#droping source code, destination code and indicator link

df_final_train.drop(['source_node', 'destination_node','indicator_link'],axis=1,inplace=True)
df_final_test.drop(['source_node', 'destination_node','indicator_link'],axis=1,inplace=True)
```

In [0]:
```
%%time
from sklearn.metrics import f1_score

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)}

xgb_clf= xgb.XGBClassifier(random_state=25,n_jobs=-1)

rf_random = RandomizedSearchCV(xgb_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 [0.98029406 0.98025694 0.97989429 0.98006164 0.98012339]
mean train scores [0.99994339 0.99997558 0.99286112 0.99557341 0.99591635]
CPU times: user 1h 40min 48s, sys: 2.86 s, total: 1h 40min 51s
Wall time: 1h 40min 52s
```

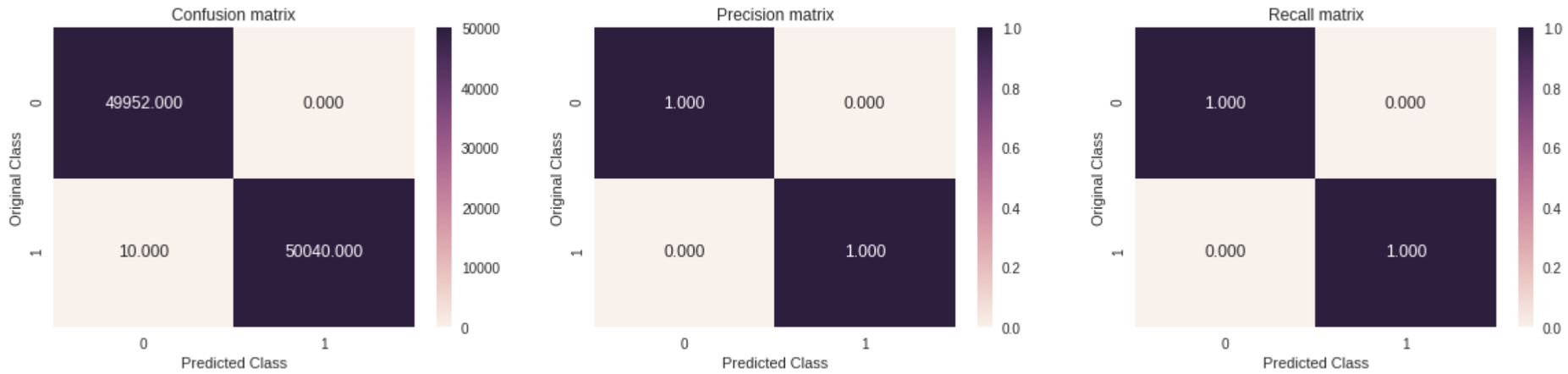
In [0]:
```
print(rf_random.best_estimator_)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=14, min_child_weight=1, missing=None, n_estimators=120,
       n_jobs=-1, nthread=None, objective='binary:logistic',
       random_state=25, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
       seed=None, silent=True, subsample=1)
```

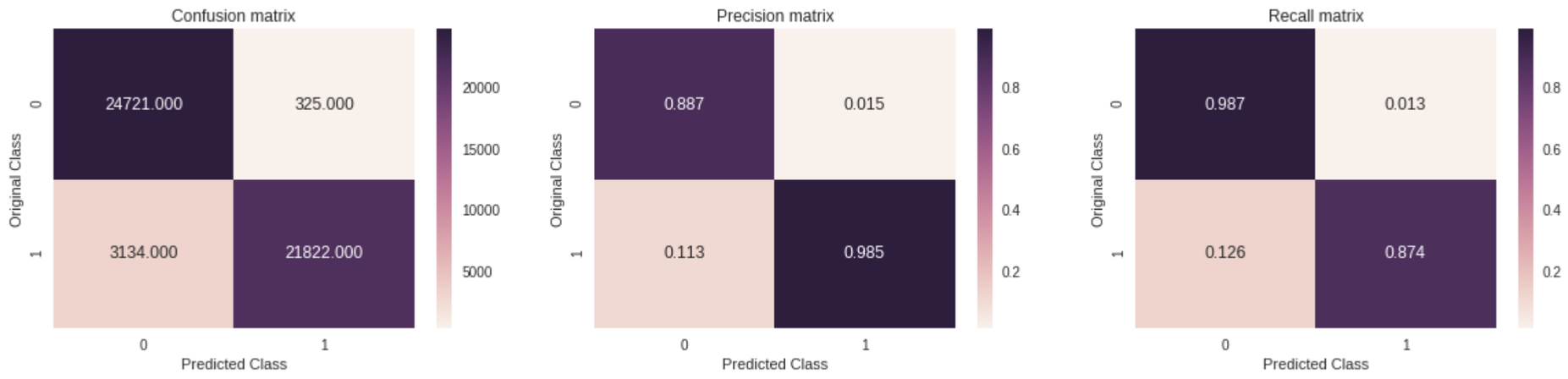
In [0]:
```
#fitting and predicting through model

xgb_clf = xgb.XGBClassifier(max_depth=14,n_estimators=120)

xgb_clf.fit(df_final_train,y_train)

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

In [0]:
```
#confusion matrix
print('Train confusion_matrix')
plot_confusion_matrix(y_train,y_train_pred)
print('Test confusion_matrix')
plot_confusion_matrix(y_test,y_test_pred)
```

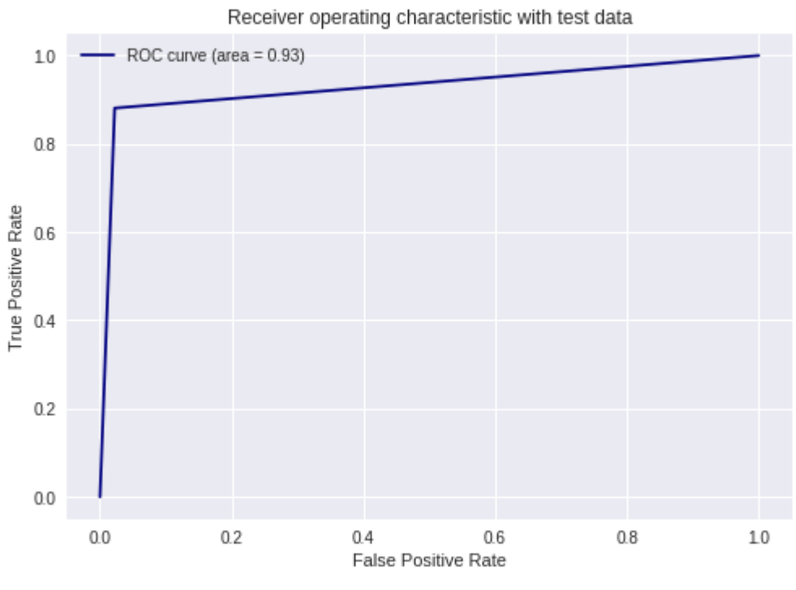
```
Train confusion_matrix
```




```
Test confusion_matrix
```

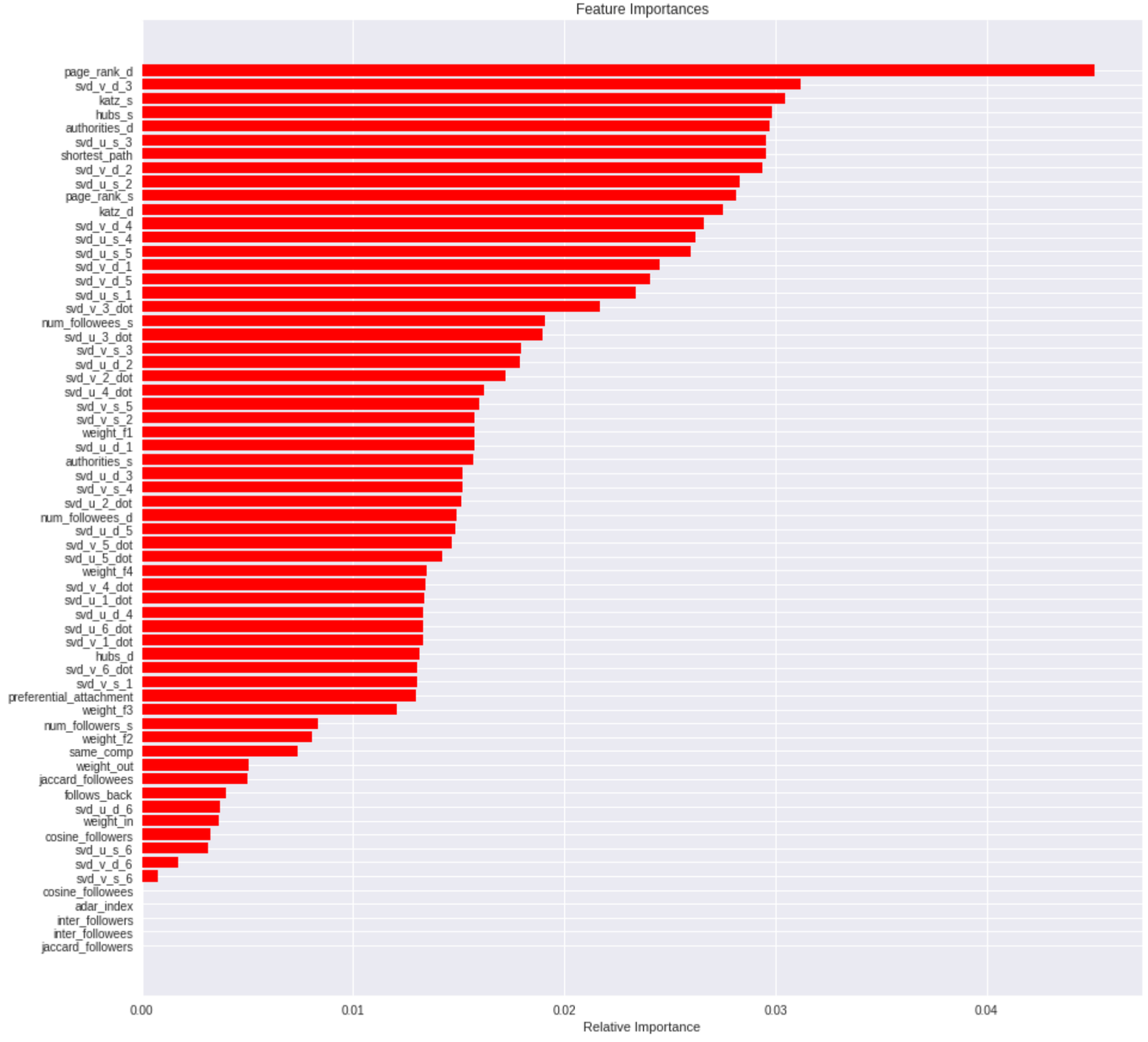


In [0]:
```
from sklearn.metrics import roc_curve, auc
fpr,tpr,ths = roc_curve(y_test,y_test_pred)
auc_sc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='navy',label='ROC curve (area = %0.2f)' % auc_sc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic with test data')
plt.legend()
plt.show()
```



Feature importance

In [0]:
```
#Feature importance
features = df_final_train.columns
importances = xgb_clf.feature_importances_
indices = (np.argsort(importances))[-70:]
plt.figure(figsize=(15,15))
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()
```


Feature Importances
page_rank_d
svd_v_d_3
katz_s
hubs_s
authorities_d
svd_u_s_3
shortest_path
svd_v_d_2
svd_u_s_2
page_rank_s
katz_d
svd_v_d_4
svd_u_s_4
svd_u_s_5
svd_v_d_1
svd_v_d_5
svd_u_s_1
svd_v_3_dot
num_followees_s
svd_u_3_dot
svd_v_s_3
svd_u_d_2
svd_v_2_dot
svd_u_4_dot
svd_v_s_5
svd_v_s_2
weight_f1
svd_u_d_1
authorities_s
svd_u_d_3
svd_v_s_4
svd_u_2_dot
num_followees_d
svd_u_d_5
svd_v_5_dot
svd_u_5_dot
weight_f4
svd_v_4_dot
svd_u_1_dot
svd_u_d_4
svd_u_6_dot
svd_v_1_dot
hubs_d
svd_v_6_dot
svd_v_s_1
preferential_attachment
weight_f3
num_followers_s
weight_f2
same_comp
weight_out
jaccard_followees
follows_back
svd_u_d_6
weight_in
cosine_followers
svd_u_s_6
svd_v_d_6
svd_v_s_6
cosine_followees
adar_index
inter_followers
inter_followees
jaccard_followers
0.00
0.01
0.02
0.03
0.04
Relative Importance

Conclution

In [0]:
```
from prettytable import PrettyTable

x = PrettyTable(["Sno", "Model","Scoring","Accuracy","Precision","Recall","F1-Score"])

x.add_row([" 1. ","Random Forest","f1","92.94%" ,"97.52%","88.09%","92.57%"])

x.add_row([" 2. ","XG Boost","f1","93.08%" ,"98.53%","87.44%","92.65%"])


print(x.get_string(title=" PrettyTable "))
```

```
+------+---------------+---------+----------+-----------+--------+----------+
| Sno  |     Model     | Scoring | Accuracy | Precision | Recall | F1-Score |
+------+---------------+---------+----------+-----------+--------+----------+
|  1.  | Random Forest |    f1   |  92.94%  |   97.52%  | 88.09% |  92.57%  |
|  2.  |    XG Boost   |    f1   |  93.08%  |   98.53%  | 87.44% |  92.65%  |
+------+---------------+---------+----------+-----------+--------+----------+
```

1. XG Boost shows best performance.

2. We have added Preferential Attachment feature , it shows great result in linking nodes.

3. SVD DOT features also played good role we can see in feature importance graph.

Steps for solving this approch:

1. Fislty we must perform finest EDA on raw data, so that we could get simple or filtered data for processing.

2. After EDA we should perform all basics algos related to graph theory and pridictions. like we have used

```
a)Follow back well scroed

b)Cosine followers also well.

c)SVD is also perform well

d)Shortest path. etc etc
```

3. After going through all basics algo test them in best classifier like Random forest or Xgboost, we have done both random forest and xgboost where xgboost stood best .

4. After modeling it will give feature importance by which we came to know which algo is best and worst.

```
So on random forest(best algo):

 a) follow back

 b) weight

 c) Shortest path

 d) cosine followers

So on Xgboost(best algo):

a)Page rank

b)SVD

c)Katz

d)Hits Score
```

5. At the end i will conclude XgBoost gives best results for this type of approch.

---xxx----