raman.shinde15@gmail.com_23

July 21, 2019

Social network Graph Link Prediction - Facebook Challenge

0.0.1 Problem statement:

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

0.0.2 Data Overview

Taken data from facebook's recruting challenge on kaggle https://www.kaggle.com/c/FacebookRecruiting
data contains two columns source and destination eac edge in graph - Data columns (total 2 columns):
- source_node int64
- destination_node int64

0.0.3 Mapping the problem into supervised learning problem:

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

0.0.4 Business objectives and constraints:

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

0.0.5 Performance metric for supervised learning:

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

- Confusion matrix

In [1]:
```
#Importing Libraries
# please do go through this python notebook:
import warnings
warnings.filterwarnings("ignore")

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

import warnings
import networkx as nx
import pdb
import pickle

from pandas import HDFStore,DataFrame
from pandas import read_hdf

import gc
from tqdm import tqdm
from scipy.sparse.linalg import svds, eigs

from sklearn.metrics import f1_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint as sp_randint
from scipy.stats import uniform
from sklearn.metrics import f1_score
```

In [2]:
```
#reading graph
if not os.path.isfile('data/after_eda/train_woheader.csv'):
    traincsv = pd.read_csv('data/train.csv')
```

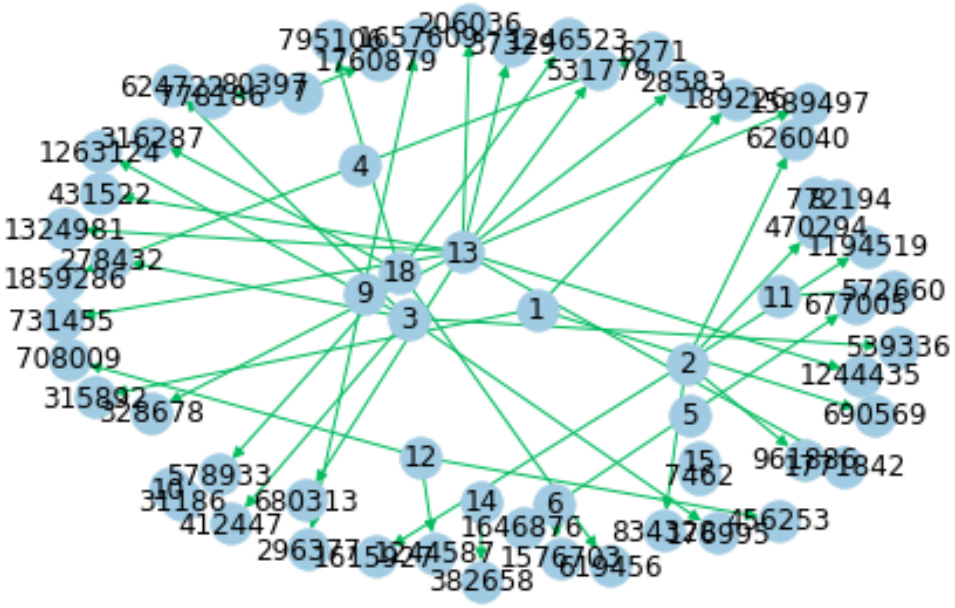
```
            print(traincsv[traincsv.isna().any(1)])
            print(traincsv.info())
            print("Number of diplicate entries: ",sum(traincsv.duplicated()))
            traincsv.to_csv('data/after_eda/train_woheader.csv',header=False,index=False)
            print("saved the graph into file")
        else:
            g=nx.read_edgelist('data/after_eda/train_woheader.csv',delimiter=',',create_using=n
            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=F

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

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

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



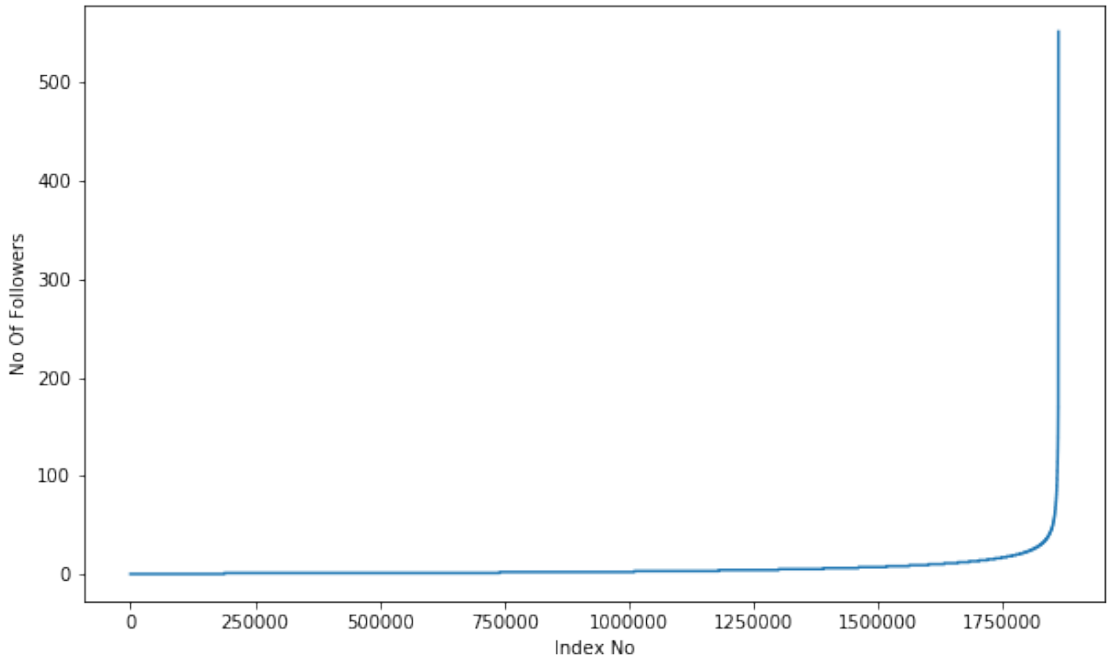
1 1. Exploratory Data Analysis

In [11]:
```
# No of Unique persons
print("The number of unique persons",len(g.nodes()))
```

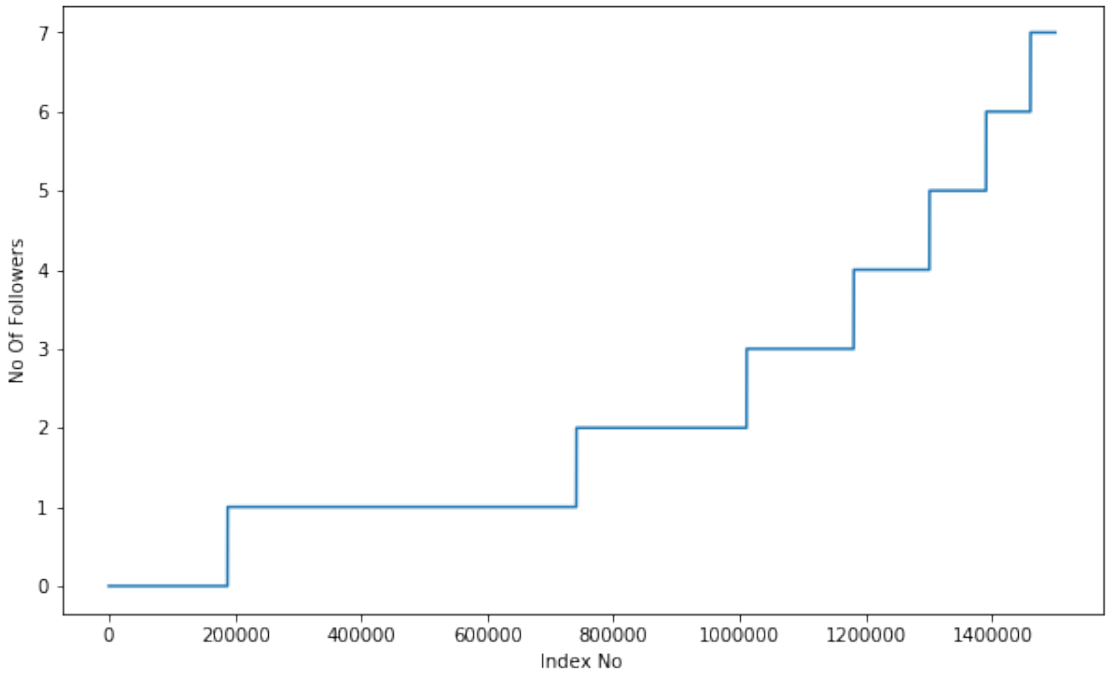
```
The number of unique persons 1862220
```

1.1 1.1 No of followers for each person

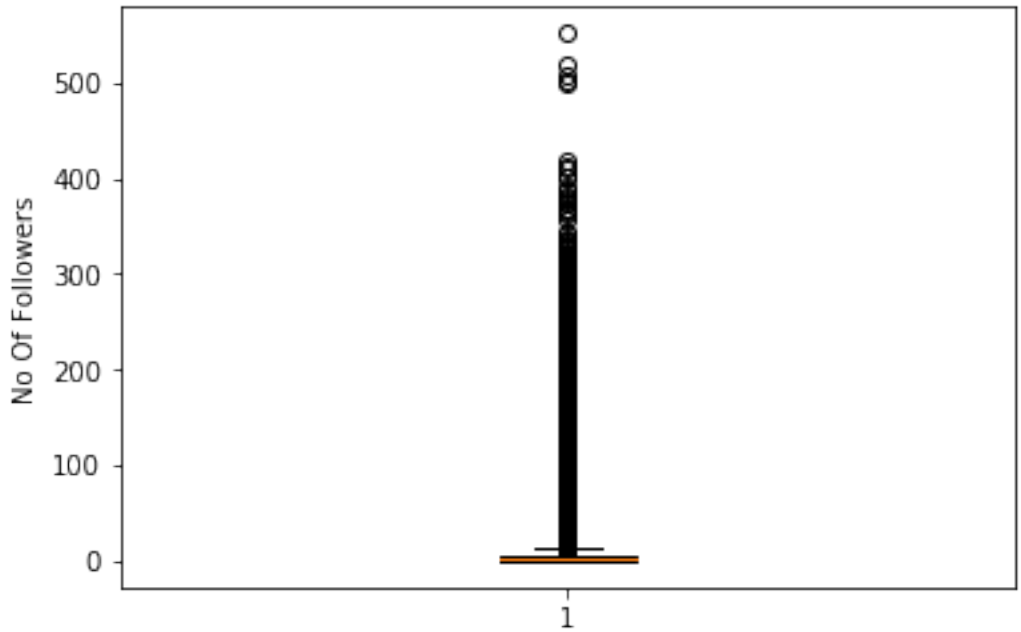
In [12]:
```
indegree_dist = list(dict(g.in_degree()).values())
indegree_dist.sort()
plt.figure(figsize=(10,6))
plt.plot(indegree_dist)
plt.xlabel('Index No')
plt.ylabel('No Of Followers')
plt.show()
```




```
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
```

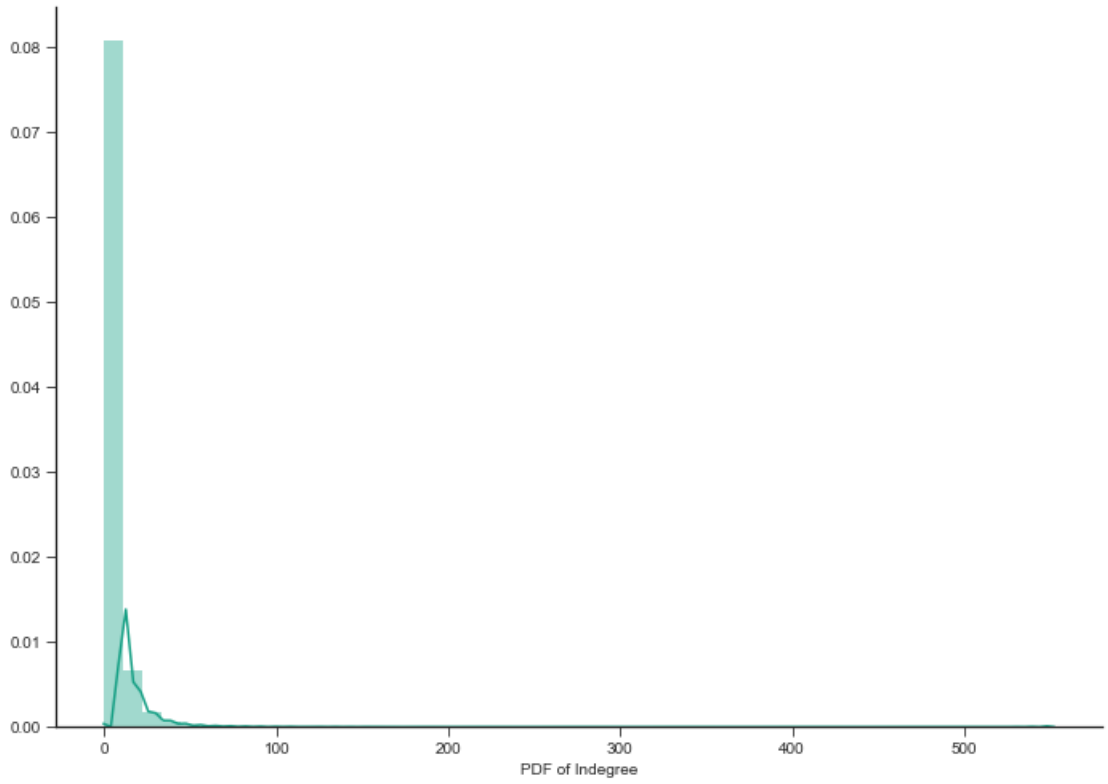
99% of data having followers of 40 only.

```
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
```

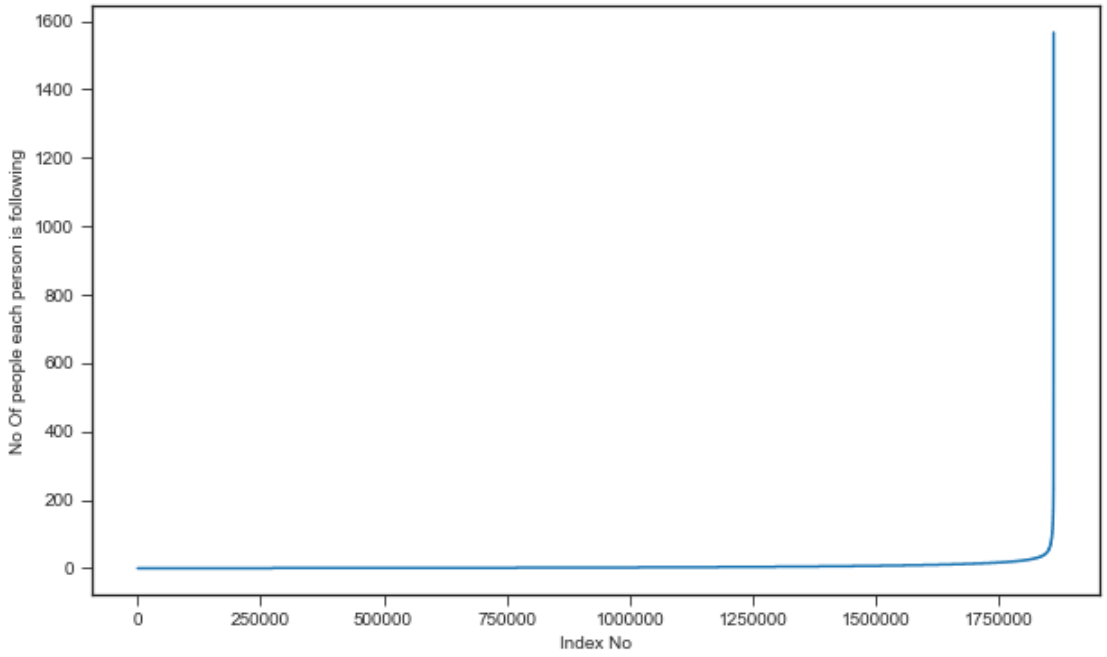
```
In [10]: %matplotlib inline
         sns.set_style('ticks')
         fig, ax = plt.subplots()
         fig.set_size_inches(11.7, 8.27)
         sns.distplot(indegree_dist, color='#16A085')
         plt.xlabel('PDF of Indegree')
         sns.despine()
         #plt.show()
```

```
D:\installed\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6571: UserWarning: The 'norme
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

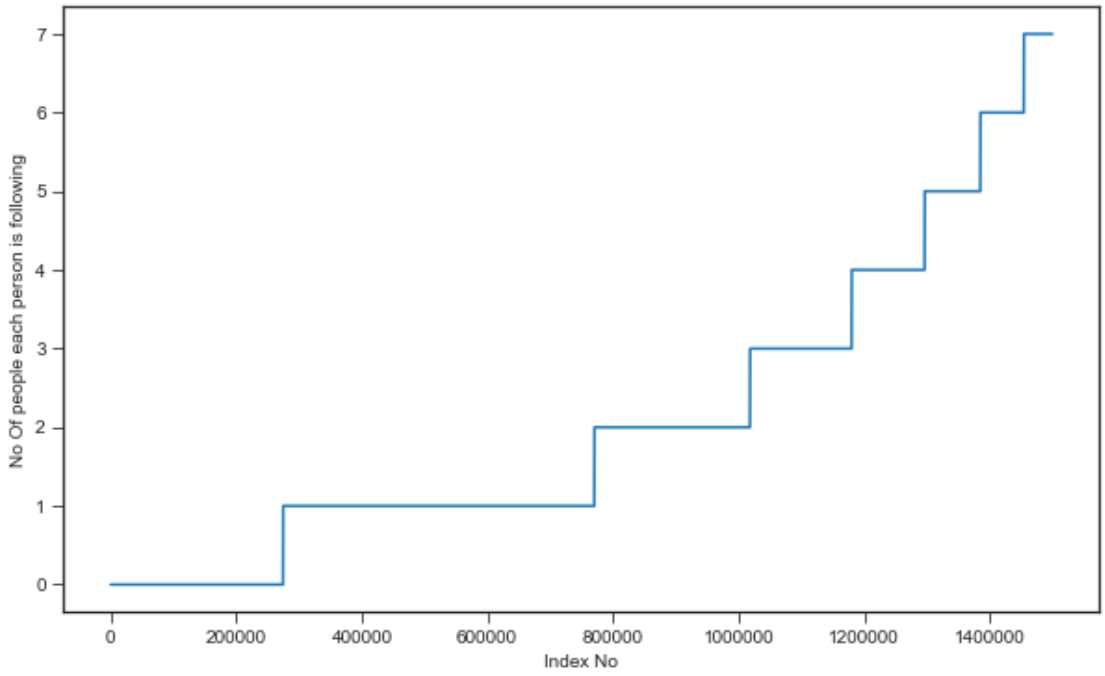


1.2 1.2 No of people each person is following

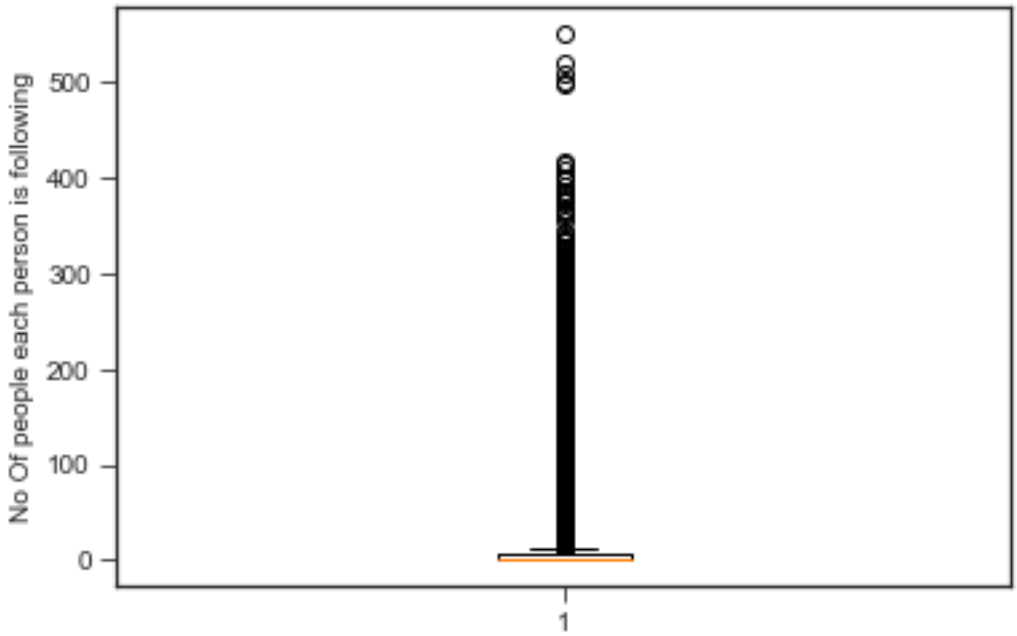
```
In [11]: outdegree_dist = list(dict(g.out_degree()).values())
         outdegree_dist.sort()
         plt.figure(figsize=(10,6))
         plt.plot(outdegree_dist)
         plt.xlabel('Index No')
         plt.ylabel('No Of people each person is following')
         plt.show()
```




```
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))
```

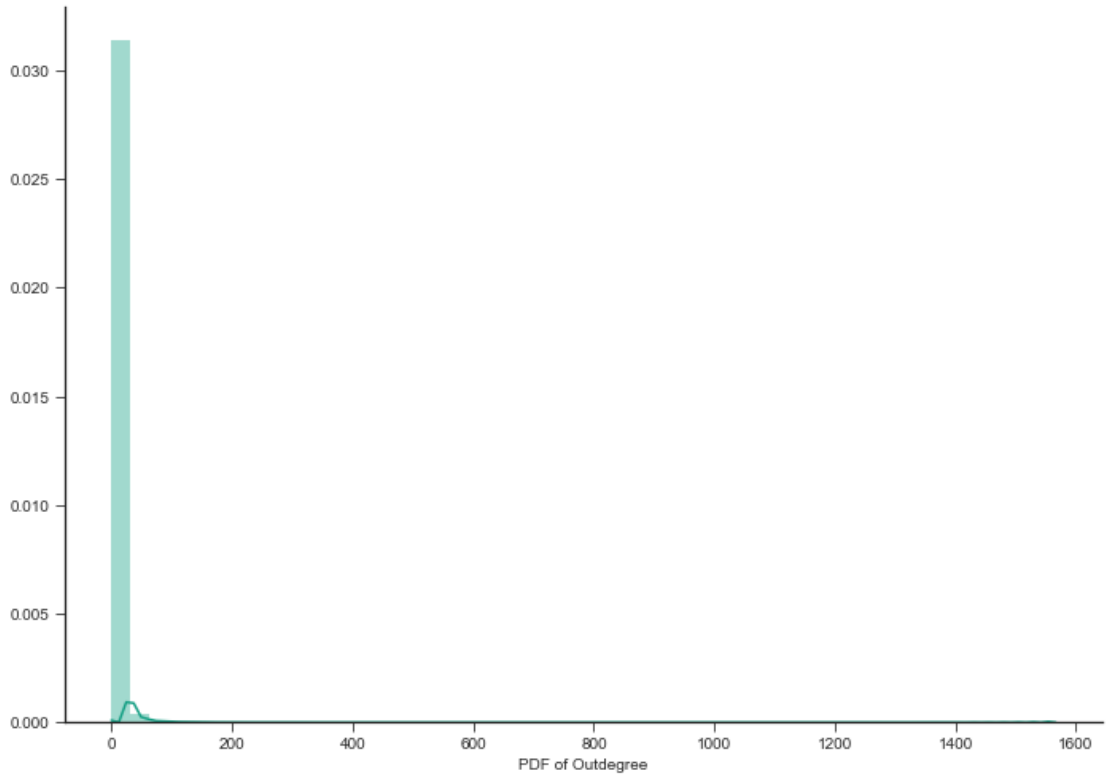
```
90 percentile value is 12.0
91 percentile value is 13.0
92 percentile value is 14.0
93 percentile value is 15.0
94 percentile value is 17.0
95 percentile value is 19.0
96 percentile value is 21.0
97 percentile value is 24.0
98 percentile value is 29.0
99 percentile value is 40.0
100 percentile value 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 'norme
  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)
                                                sum(np.array(outdegree_dist)==0)*100/len(outdegree_dis
```

```
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
                                                sum(np.array(indegree_dist)==0)*100/len(indegree_dist)
```

```
No of persons having zero followers are 188043 and % is 10.097786512871734
```

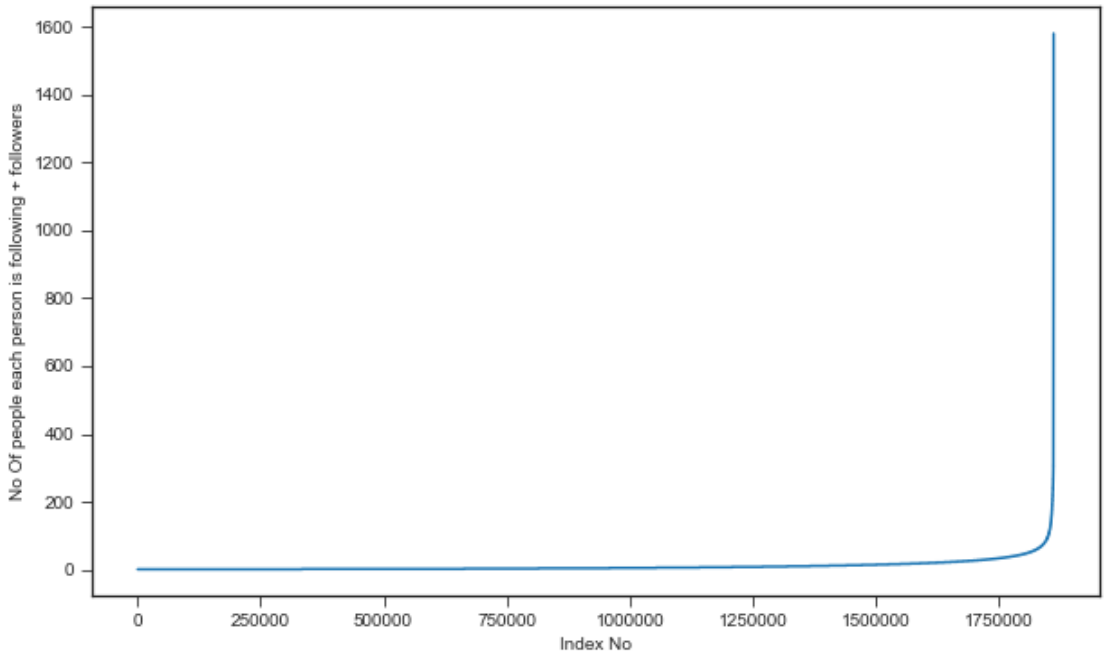
```
In [19]: count=0
         for i in g.nodes():
             if len(list(g.predecessors(i)))==0 :
                 if len(list(g.successors(i)))==0:
                     count+=1
         print('No of persons those are not not following anyone and also not having any follow
```

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

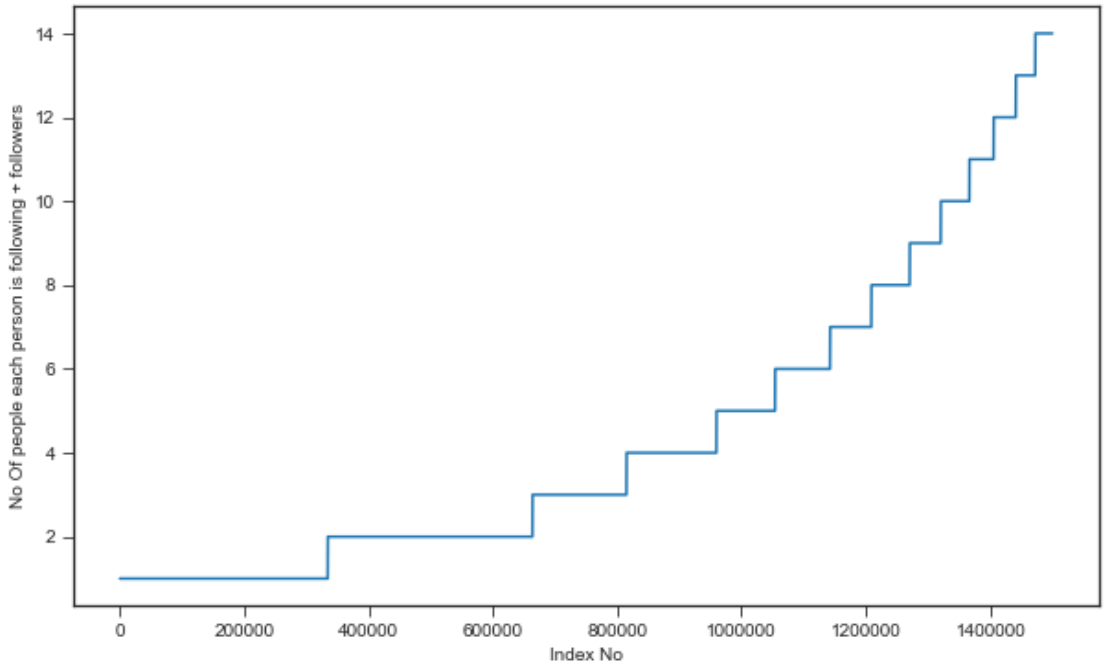
1.3 1.3 both followers + following

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 follo
```

```
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 follo
```

```
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_degr
```

```
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 2. Posing a problem as classification problem

2.1 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 [7]: %%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: 3.17 s
```

2.2 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.isf
             #reading total data df
             df_pos = pd.read_csv('data/train.csv')
             df_neg = pd.DataFrame(list(missing_edges), columns=['source_node', 'destination_no

             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 da
             #and for feature generation
             X_train_pos, X_test_pos, y_train_pos, y_test_pos  = train_test_split(df_pos,np.one
```

```
            X_train_neg, X_test_neg, y_train_neg, y_test_neg  = train_test_split(df_neg,np.zer

            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[
            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]

            #removing header and saving
            X_train_pos.to_csv('data/after_eda/train_pos_after_eda.csv',header=False, index=Fa
            X_test_pos.to_csv('data/after_eda/test_pos_after_eda.csv',header=False, index=Fals
            X_train_neg.to_csv('data/after_eda/train_neg_after_eda.csv',header=False, index=Fa
            X_test_neg.to_csv('data/after_eda/test_neg_after_eda.csv',header=False, index=Fals
        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('dat
            train_graph=nx.read_edgelist('data/after_eda/train_pos_after_eda.csv',delimiter='
            test_graph=nx.read_edgelist('data/after_eda/test_pos_after_eda.csv',delimiter=','
            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 {
```

```
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
             X_test_pos = pd.read_csv('data/after_eda/test_pos_after_eda.csv', names=['source_n
             X_train_neg = pd.read_csv('data/after_eda/train_neg_after_eda.csv', names=['source
             X_test_neg = pd.read_csv('data/after_eda/test_neg_after_eda.csv', names=['source_n

             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=Fa
            pd.DataFrame(y_test.astype(int)).to_csv('data/test_y.csv',header=False,index=Fals
```

```
=============================================================
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,)
```

```
In [53]: # computed and store the data for featurization
         # please check out FB_featurization.ipynb
```

3 3. Featurization

```
In [2]: if os.path.isfile('data/after_eda/train_pos_after_eda.csv'):
            train_graph=nx.read_edgelist('data/after_eda/train_pos_after_eda.csv',delimiter=',
            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
```

3.1 3. Similarity measures

3.2 3.1 Jaccard Distance:

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

```
In [3]: #for followees
        def jaccard_for_followees(a,b):
```

```
            try:
                if len(set(train_graph.successors(a))) == 0  | len(set(train_graph.successors(
                    return 0
                sim = (len(set(train_graph.successors(a)).intersection(set(train_graph.successo
                                            (len(set(train_graph.successors(a)).union(set(trai
            except:
                return 0
            return sim
```

```
In [4]: #for followers
        def jaccard_for_followers(a,b):
            try:
                if len(set(train_graph.predecessors(a))) == 0  | len(set(g.predecessors(b))) ==
                    return 0
                sim = (len(set(train_graph.predecessors(a)).intersection(set(train_graph.predec
                                        (len(set(train_graph.predecessors(a)).union(set(train_
                return sim
            except:
                return 0
```

3.3 3.2 Cosine distance

```
In [5]: #for followees
        def cosine_for_followees(a,b):
            try:
                if len(set(train_graph.successors(a))) == 0  | len(set(train_graph.successors(
                    return 0
                sim = (len(set(train_graph.successors(a)).intersection(set(train_graph.successo
                                            (math.sqrt(len(set(train_graph.successors(a)))*len
                return sim
            except:
                return 0
```

```
In [6]: def cosine_for_followers(a,b):
            try:

                if len(set(train_graph.predecessors(a))) == 0  | len(set(train_graph.predecesso
                    return 0
                sim = (len(set(train_graph.predecessors(a)).intersection(set(train_graph.predec
                                             (math.sqrt(len(set(train_graph.predecessors(a))))*
                return sim
            except:
                return 0
```

3.4 3.3 Ranking Measures

3.4.1 3.3.1 Page Ranking

```
In [7]: 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 [8]: 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 [9]: #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
```

3.5 4. Other Graph Features

3.5.1 4.1 Shortest path:

```
In [10]: #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
```

3.5.2 4.2 Checking for same community

```
In [11]: #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
```

3.5.3 4.3 Adamic/Adar Index:

```
In [12]: #adar index
         def calc_adar_in(a,b):
             sum=0
             try:
                 n=list(set(train_graph.successors(a)).intersection(set(train_graph.successors
                 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
```

3.5.4 4.4 Is persion was following back:

```
In [13]: def follows_back(a,b):
             if train_graph.has_edge(b,a):
                 return 1
             else:
                 return 0
```

3.5.5 4.5 Katz Centrality:

```
In [14]: 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 [15]: 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 [16]: mean_katz = float(sum(katz.values())) / len(katz)
         print(mean_katz)

0.0007483800935562018
```

3.5.6 4.6 Hits Score

```
In [17]: 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 [18]: 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
```

3.6 5. Featurization

3.6.1 5. 1 Reading a sample of Data from both train and test

```
In [19]: import random
         if os.path.isfile('data/after_eda/train_after_eda.csv'):
             filename = "data/after_eda/train_after_eda.csv"
             # you uncomment this line, if you dont know the lentgh of the file name
             # here we have hardcoded the number of lines as 15100030
             # n_train = sum(1 for line in open(filename)) #number of records in file (exclude
```

```
            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 [20]: if os.path.isfile('data/after_eda/test_after_eda.csv'):
            filename = "data/after_eda/test_after_eda.csv"
            # you uncomment this line, if you dont know the lentgh of the file name
            # here we have hardcoded the number of lines as 3775008
            # n_test = sum(1 for line in open(filename)) #number of records in file (excludes
            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 [21]: 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 [22]: df_final_train = pd.read_csv('data/after_eda/train_after_eda.csv', skiprows=skip_trai
         df_final_train['indicator_link'] = pd.read_csv('data/train_y.csv', skiprows=skip_trai
         print("Our train matrix size ",df_final_train.shape)
         df_final_train.head(2)
```

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

```
Out[22]:    source_node  destination_node  indicator_link
         0       273084           1505602               1
         1       335122           1021192               1
```

```
In [23]: df_final_test = pd.read_csv('data/after_eda/test_after_eda.csv', skiprows=skip_test, 
         df_final_test['indicator_link'] = pd.read_csv('data/test_y.csv', skiprows=skip_test, 
         print("Our test matrix size ",df_final_test.shape)
         df_final_test.head(2)
```

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

```
Out[23]:    source_node  destination_node  indicator_link
         0       848424            784690               1
         1      1450917           1777605               1
```

3.6.2 5.2 Adding a set of features

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

jaccard_followers
jaccard_followees
cosine_followers
cosine_followees
num_followers_s
num_followees_s
num_followers_d
num_followees_d
inter_followers
inter_followees

```
In [24]: 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'],
             df_final_test['jaccard_followers'] = df_final_test.apply(lambda row:
                                                   jaccard_for_followers(row['source_node'],

             #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'],
             df_final_test['jaccard_followees'] = df_final_test.apply(lambda row:
                                                   jaccard_for_followees(row['source_node'],


                 #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'],r
             df_final_test['cosine_followers'] = df_final_test.apply(lambda row:
                                                   cosine_for_followers(row['source_node'],r

             #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'],r
             df_final_test['cosine_followees'] = df_final_test.apply(lambda row:
                                                   cosine_for_followees(row['source_node'],r

In [25]: 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_
```

```
In [26]: if not os.path.isfile('storage_sample_stage1.h5'):
            df_final_train['num_followers_s'], df_final_train['num_followers_d'], \
            df_final_train['num_followees_s'], df_final_train['num_followees_d'], \
            df_final_train['inter_followers'], df_final_train['inter_followees']= compute_fea

            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_featu

            hdf = HDFStore('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('storage_sample_stage1.h5', 'train_df',mode='r')
            df_final_test = read_hdf('storage_sample_stage1.h5', 'test_df',mode='r')
```

3.6.3 5.3 Adding new set of features

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

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

```
In [27]: 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[
             #mapping adar index on test
             df_final_test['adar_index'] = df_final_test.apply(lambda row: calc_adar_in(row['so

             #------------------------------------------------------------------------------
             #mapping followback or not on train
             df_final_train['follows_back'] = df_final_train.apply(lambda row: follows_back(rou

             #mapping followback or not on test
             df_final_test['follows_back'] = df_final_test.apply(lambda row: follows_back(row[

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

             ##mapping same component of wcc or not on train
             df_final_test['same_comp'] = df_final_test.apply(lambda row: belongs_to_same_wcc(r

             #------------------------------------------------------------------------------
             #mapping shortest path on train
             df_final_train['shortest_path'] = df_final_train.apply(lambda row: compute_shortes
             #mapping shortest path on test
             df_final_test['shortest_path'] = df_final_test.apply(lambda row: compute_shortest_

             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',m
             df_final_test = read_hdf('data/fea_sample/storage_sample_stage2.h5', 'test_df',mod
```

3.6.4 5.4 Adding new set of features

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

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
Page Ranking of source
Page Ranking of dest
katz of source
katz of dest

hubs of source
hubs of dest
authorities_s of source
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

```
In [28]: #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 [00:13<00:00, 133800.67it/s]
```

```
In [29]: if not os.path.isfile('data/fea_sample/storage_sample_stage3.h5'):
             print("File not present")
             #mapping to pandas train
             df_final_train['weight_in'] = df_final_train.destination_node.apply(lambda x: Wei
             df_final_train['weight_out'] = df_final_train.source_node.apply(lambda x: Weight_

             #mapping to pandas test
             df_final_test['weight_in'] = df_final_test.destination_node.apply(lambda x: Weight
             df_final_test['weight_out'] = df_final_test.source_node.apply(lambda x: 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.weigh
             df_final_train['weight_f4'] = (1*df_final_train.weight_in + 2*df_final_train.weigh
```

```
        #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_
        df_final_test['weight_f4'] = (1*df_final_test.weight_in + 2*df_final_test.weight_
```

```
File not present
```

```
In [30]: 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(
        df_final_train['page_rank_d'] = df_final_train.destination_node.apply(lambda x:pr

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

        #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,
        df_final_train['katz_d'] = df_final_train.destination_node.apply(lambda x: katz.g

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

        #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
        df_final_train['hubs_d'] = df_final_train.destination_node.apply(lambda x: hits[0

        df_final_test['hubs_s'] = df_final_test.source_node.apply(lambda x: hits[0].get(x
        df_final_test['hubs_d'] = df_final_test.destination_node.apply(lambda x: hits[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
        df_final_train['authorities_d'] = df_final_train.destination_node.apply(lambda x:

        df_final_test['authorities_s'] = df_final_test.source_node.apply(lambda x: hits[1
        df_final_test['authorities_d'] = df_final_test.destination_node.apply(lambda x: h
        #================================================================================

        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',m
        df_final_test = read_hdf('data/fea_sample/storage_sample_stage3.h5', 'test_df',mod
```

In [32]:
```
df_final_train.head()
```

Out[32]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	num_followers_s	num_followees_s	num_followees_d	...	weight_f3	weight_f4	page_rank_s	page_rank_d	katz_s	katz_d	hubs_s	hubs_d	authorities_s	authorities_d
0	273084	1505602	1	0	0.000000	0.000000	0.000000	11	15	8	...	1.005929	0.877964	2.045290e-06	3.459963e-07	0.000773	0.000756	1.943132e-13	1.941103e-13	9.226339e-16	2.231877e-15
1	832016	1543415	1	0	0.187135	0.028382	0.343828	17	61	142	...	0.332196	0.356598	2.353458e-07	6.427660e-07	0.000845	0.001317	3.906648e-11	9.424102e-11	1.208074e-11	1.273080e-10
2	1325247	760242	1	0	0.369565	0.156957	0.566038	35	41	22	...	0.525694	0.494302	6.211019e-07	5.179801e-07	0.000885	0.000855	7.730764e-114	4.067322e-114	2.681298e-113	2.199205e-113
3	1368400	1006992	1	0	0.000000	0.000000	0.000000	2	5	7	...	0.985599	1.105172	2.998153e-07	1.704245e-06	0.000739	0.000773	5.443738e-17	4.139999e-16	2.413250e-14	6.688064e-15
4	140165	1708748	1	0	0.000000	0.000000	0.000000	5	11	3	...	2.301511	1.603023	4.349180e-07	2.089590e-07	0.000751	0.000735	3.887821e-16	4.721269e-16	7.552255e-16	2.734009e-18

[5 rows x 31 columns]

3.6.5 5.5 Adding new set of features

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

SVD features for both source and destination

```
In [33]: def svd(x, S):
             try:
                 z = sadj_dict[x]
                 return S[z]
             except:
                 return [0,0,0,0,0,0]

In [34]: #for svd features to get feature vector creating a dict node val and inedx in svd vec
         sadj_col = sorted(train_graph.nodes())
         sadj_dict = { val:idx for idx,val in enumerate(sadj_col)}

In [35]: Adj = nx.adjacency_matrix(train_graph,nodelist=sorted(train_graph.nodes())).asfptype()

In [36]: 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 [37]: 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', 's
             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','s
             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', 's
             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','s
             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', 'sv
             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','sv
             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', 'sv
            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','sv
            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 [38]: df_final_train.columns
```

```
Out[38]: 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',
                'num_followers_d', 'adar_index', 'follows_back', 'same_comp',
                'shortest_path', 'weight_in', 'weight_out', 'weight_f1', 'weight_f2',
                'weight_f3', 'weight_f4', 'page_rank_s', 'page_rank_d', 'katz_s',
                'katz_d', 'hubs_s', 'hubs_d', 'authorities_s', 'authorities_d',
                'svd_u_s_1', 'svd_u_s_2', 'svd_u_s_3', 'svd_u_s_4', 'svd_u_s_5',
                'svd_u_s_6', 'svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3', 'svd_u_d_4',
                'svd_u_d_5', 'svd_u_d_6', 'svd_v_s_1', 'svd_v_s_2', 'svd_v_s_3',
                'svd_v_s_4', 'svd_v_s_5', 'svd_v_s_6', 'svd_v_d_1', 'svd_v_d_2',
                'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5', 'svd_v_d_6'],
               dtype='object')
```

3.6.6 5.6 Preferential Attachment

Preferential Attachement for followers

```
In [39]: #for train dataset

         train_followers_s = np.array(df_final_train['num_followers_s'])
         train_followers_d = np.array(df_final_train['num_followers_d'])
         preferential_followers=[]
         for i in range(len(train_followers_s)):
             preferential_followers.append(train_followers_d[i]*train_followers_s[i])
         df_final_train['preferential_followers']= preferential_followers
         df_final_train.head()
```

```
Out[39]:    source_node  destination_node  indicator_link  jaccard_followers  \
         0       273084           1505602               1                  0
         1       832016           1543415               1                  0
```

```
2      1325247           760242               1                   0
3      1368400          1006992               1                   0
4       140165          1708748               1                   0

   jaccard_followees  cosine_followers  cosine_followees  num_followers_s  \
0           0.000000          0.000000          0.000000               11
1           0.187135          0.028382          0.343828               17
2           0.369565          0.156957          0.566038               35
3           0.000000          0.000000          0.000000                2
4           0.000000          0.000000          0.000000                5

   num_followees_s  num_followees_d  ...     svd_v_s_4     svd_v_s_5  \
0               15                8  ...  1.545075e-13  8.108434e-13
1               61              142  ...  1.345726e-02  3.703479e-12
2               41               22  ... -7.021227e-19  1.940403e-19
3                5                7  ...  1.514614e-11  1.513483e-12
4               11                3  ...  1.999809e-14  3.360247e-13

      svd_v_s_6     svd_v_d_1     svd_v_d_2     svd_v_d_3     svd_v_d_4  \
0  1.719702e-14 -1.355368e-12  4.675307e-13  1.128591e-06  6.616550e-14
1  2.251737e-10  1.245101e-12 -1.636948e-10 -3.112650e-10  6.738902e-02
2 -3.365389e-19 -1.238370e-18  1.438175e-19 -1.852863e-19 -5.901864e-19
3  4.498061e-13 -9.818087e-10  3.454672e-11  5.213635e-08  9.595823e-13
4  1.407670e-14  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00

      svd_v_d_5     svd_v_d_6  preferential_followers
0  9.771077e-13  4.159752e-14                      66
1  2.607801e-11  2.372904e-09                    1598
2  1.629341e-19 -2.572452e-19                     980
3  3.047045e-10  1.246592e-13                      22
4  0.000000e+00  0.000000e+00                       5

[5 rows x 56 columns]
```

In [40]:
```
# for test dataset

test_followers_s = np.array(df_final_test['num_followers_s'])
test_followers_d = np.array(df_final_test['num_followers_d'])
preferential_followers=[]
for i in range(len(test_followers_s)):
    preferential_followers.append(test_followers_d[i]*test_followers_s[i])
df_final_test['preferential_followers']= preferential_followers
df_final_test.head()
```

Out[40]:
```
   source_node  destination_node  indicator_link  jaccard_followers  \
0       848424            784690               1                  0
1       483294           1255532               1                  0
2       626190           1729265               1                  0
```

```
3       947219           425228               1                   0
4       991374           975044               1                   0

   jaccard_followees  cosine_followers  cosine_followees  num_followers_s  \
0                0.0          0.029161          0.000000                6
1                0.0          0.000000          0.000000                2
2                0.0          0.000000          0.000000               15
3                0.0          0.000000          0.000000               11
4                0.2          0.042767          0.347833               12

   num_followees_s  num_followees_d  ...     svd_v_s_4     svd_v_s_5  \
0                6                9  ...  2.701538e-12  4.341620e-13
1                1               19  ...  2.248568e-14  3.600957e-13
2               16                9  ...  1.778927e-12  2.740535e-13
3               10               34  ...  7.917166e-13  4.020707e-12
4               15               27  ...  1.361574e-13  1.154623e-12

      svd_v_s_6     svd_v_d_1     svd_v_d_2     svd_v_d_3     svd_v_d_4  \
0  5.535503e-14 -9.994076e-10  5.791910e-10  3.512364e-07  2.486658e-09
1  4.701436e-15 -9.360516e-12  3.206809e-10  4.668696e-08  6.665777e-12
2  4.199834e-14 -4.253075e-13  4.789463e-13  3.479824e-07  1.630549e-13
3  2.817657e-13 -2.162590e-11  6.939194e-12  1.879861e-05  4.384816e-12
4  9.656662e-14 -8.742904e-12  7.467370e-12  1.256880e-05  3.636983e-12

      svd_v_d_5     svd_v_d_6  preferential_followers
0  2.771146e-09  1.727694e-12                      84
1  1.495979e-10  9.836670e-14                      34
2  3.954708e-13  3.875785e-14                     150
3  1.239414e-11  6.483485e-13                     407
4  3.948463e-12  2.415863e-13                     324

[5 rows x 56 columns]
```

Preferential Attachement for followees

In [41]:
```
#for train dataset

train_followees_s = np.array(df_final_train['num_followees_s'])
train_followees_d = np.array(df_final_train['num_followees_d'])
preferential_followees=[]
for i in range(len(train_followees_s)):
    preferential_followees.append(train_followees_d[i]*train_followees_s[i])
df_final_train['preferential_followees']= preferential_followees
df_final_train.head()
```

Out[41]:
```
   source_node  destination_node  indicator_link  jaccard_followers  \
0       273084           1505602               1                  0
1       832016           1543415               1                  0
```

```
2      1325247            760242               1                   0
3      1368400           1006992               1                   0
4       140165           1708748               1                   0

   jaccard_followees  cosine_followers  cosine_followees  num_followers_s  \
0           0.000000          0.000000          0.000000               11
1           0.187135          0.028382          0.343828               17
2           0.369565          0.156957          0.566038               35
3           0.000000          0.000000          0.000000                2
4           0.000000          0.000000          0.000000                5

   num_followees_s  num_followees_d  ...     svd_v_s_5     svd_v_s_6  \
0               15                8  ...  8.108434e-13  1.719702e-14
1               61              142  ...  3.703479e-12  2.251737e-10
2               41               22  ...  1.940403e-19 -3.365389e-19
3                5                7  ...  1.513483e-12  4.498061e-13
4               11                3  ...  3.360247e-13  1.407670e-14

      svd_v_d_1     svd_v_d_2     svd_v_d_3     svd_v_d_4     svd_v_d_5  \
0 -1.355368e-12  4.675307e-13  1.128591e-06  6.616550e-14  9.771077e-13
1  1.245101e-12 -1.636948e-10 -3.112650e-10  6.738902e-02  2.607801e-11
2 -1.238370e-18  1.438175e-19 -1.852863e-19 -5.901864e-19  1.629341e-19
3 -9.818087e-10  3.454672e-11  5.213635e-08  9.595823e-13  3.047045e-10
4  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00

      svd_v_d_6  preferential_followers  preferential_followees
0  4.159752e-14                      66                     120
1  2.372904e-09                    1598                    8662
2 -2.572452e-19                     980                     902
3  1.246592e-13                      22                      35
4  0.000000e+00                       5                      33

[5 rows x 57 columns]
```

In [42]:
```
#for test dataset

test_followees_s = np.array(df_final_test['num_followees_s'])
test_followees_d = np.array(df_final_test['num_followees_d'])
preferential_followees=[]
for i in range(len(test_followees_s)):
    preferential_followees.append(test_followees_d[i]*test_followees_s[i])
df_final_test['preferential_followees']= preferential_followees
df_final_test.head()
```

Out[42]:
```
   source_node  destination_node  indicator_link  jaccard_followers  \
0       848424            784690               1                  0
1       483294           1255532               1                  0
2       626190           1729265               1                  0
```

```
3       947219           425228               1                   0
4       991374           975044               1                   0

   jaccard_followees  cosine_followers  cosine_followees  num_followers_s  \
0                0.0          0.029161          0.000000                6
1                0.0          0.000000          0.000000                2
2                0.0          0.000000          0.000000               15
3                0.0          0.000000          0.000000               11
4                0.2          0.042767          0.347833               12

   num_followees_s  num_followees_d  ...     svd_v_s_5     svd_v_s_6  \
0                6                9  ...  4.341620e-13  5.535503e-14
1                1               19  ...  3.600957e-13  4.701436e-15
2               16                9  ...  2.740535e-13  4.199834e-14
3               10               34  ...  4.020707e-12  2.817657e-13
4               15               27  ...  1.154623e-12  9.656662e-14

      svd_v_d_1     svd_v_d_2     svd_v_d_3     svd_v_d_4     svd_v_d_5  \
0 -9.994076e-10  5.791910e-10  3.512364e-07  2.486658e-09  2.771146e-09
1 -9.360516e-12  3.206809e-10  4.668696e-08  6.665777e-12  1.495979e-10
2 -4.253075e-13  4.789463e-13  3.479824e-07  1.630549e-13  3.954708e-13
3 -2.162590e-11  6.939194e-12  1.879861e-05  4.384816e-12  1.239414e-11
4 -8.742904e-12  7.467370e-12  1.256880e-05  3.636983e-12  3.948463e-12

      svd_v_d_6  preferential_followers  preferential_followees
0  1.727694e-12                     84                      54
1  9.836670e-14                     34                      19
2  3.875785e-14                    150                     144
3  6.483485e-13                    407                     340
4  2.415863e-13                    324                     405

[5 rows x 57 columns]
```

3.6.7 5.7 SVD_dot

```
In [43]: #for train datasets
         su1,su2,su3,su4,su5,su6=df_final_train['svd_u_s_1'],df_final_train['svd_u_s_2'],df_fi
         sv1,sv2,sv3,sv4,sv5,sv6=df_final_train['svd_v_s_1'],df_final_train['svd_v_s_2'],df_fi

         du1,du2,du3,du4,du5,du6=df_final_train['svd_u_d_1'],df_final_train['svd_u_d_2'],df_fi
         dv1,dv2,dv3,dv4,dv5,dv6=df_final_train['svd_v_d_1'],df_final_train['svd_v_d_2'],df_fi


In [44]: suv = np.array([su1,su2,su3,su4,su5,su6,sv1,sv2,sv3,sv4,sv5,sv6]).T
         print(suv.shape)

(100002, 12)
```

```
In [45]: duv = np.array([du1,du2,du3,du4,du5,du6,dv1,dv2,dv3,dv4,dv5,dv6]).T
         print(duv.shape)

(100002, 12)


In [46]: svd_dot = []
         for ea in range(suv.shape[0]):
             svd_dot.append(np.dot(suv[ea],duv[ea]))
         df_final_train['svd_dot']=svd_dot

In [47]: #for test datasets
         su1,su2,su3,su4,su5,su6=df_final_test['svd_u_s_1'],df_final_test['svd_u_s_2'],df_final
         sv1,sv2,sv3,sv4,sv5,sv6=df_final_test['svd_v_s_1'],df_final_test['svd_v_s_2'],df_final

         du1,du2,du3,du4,du5,du6=df_final_test['svd_u_d_1'],df_final_test['svd_u_d_2'],df_final
         dv1,dv2,dv3,dv4,dv5,dv6=df_final_test['svd_v_d_1'],df_final_test['svd_v_d_2'],df_final


In [48]: suv = np.array([su1,su2,su3,su4,su5,su6,sv1,sv2,sv3,sv4,sv5,sv6]).T
         print(suv.shape)

         duv = np.array([du1,du2,du3,du4,du5,du6,dv1,dv2,dv3,dv4,dv5,dv6]).T
         print(duv.shape)

(50002, 12)
(50002, 12)


In [49]: svd_dot = []
         for ea in range(suv.shape[0]):
             svd_dot.append(np.dot(suv[ea],duv[ea]))
         df_final_test['svd_dot']=svd_dot

In [91]: hdf = HDFStore('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()
```

4 Models

```
In [92]: #reading
         from pandas import read_hdf
         df_final_train = read_hdf('data/fea_sample/storage_sample_stage4.h5', 'train_df',mode=
         df_final_test = read_hdf('data/fea_sample/storage_sample_stage4.h5', 'test_df',mode='r

In [50]: df_final_train.columns
```

```
Out[50]: 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',
                'num_followers_d', 'adar_index', 'follows_back', 'same_comp',
                'shortest_path', 'weight_in', 'weight_out', 'weight_f1', 'weight_f2',
                'weight_f3', 'weight_f4', 'page_rank_s', 'page_rank_d', 'katz_s',
                'katz_d', 'hubs_s', 'hubs_d', 'authorities_s', 'authorities_d',
                'svd_u_s_1', 'svd_u_s_2', 'svd_u_s_3', 'svd_u_s_4', 'svd_u_s_5',
                'svd_u_s_6', 'svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3', 'svd_u_d_4',
                'svd_u_d_5', 'svd_u_d_6', 'svd_v_s_1', 'svd_v_s_2', 'svd_v_s_3',
                'svd_v_s_4', 'svd_v_s_5', 'svd_v_s_6', 'svd_v_d_1', 'svd_v_d_2',
                'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5', 'svd_v_d_6',
                'preferential_followers', 'preferential_followees', 'svd_dot'],
               dtype='object')
```

```
In [51]: y_train = df_final_train.indicator_link
         y_test = df_final_test.indicator_link
```

```
In [52]: df_final_train.drop(['source_node', 'destination_node','indicator_link'],axis=1,inpla
         df_final_test.drop(['source_node', 'destination_node','indicator_link'],axis=1,inplac
```

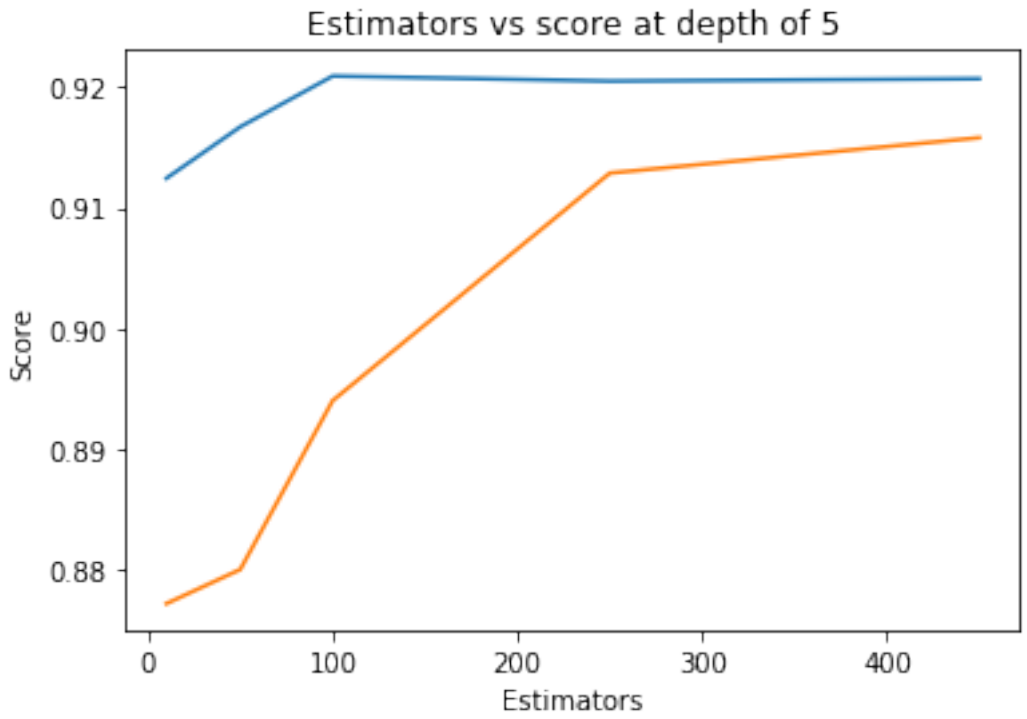
4.1 6.1 Random Forest

```
In [53]: 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,v
             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 0.9124443324090412 test Score 0.8771579769237442
Estimators =  50 Train Score 0.9167164460653316 test Score 0.8799948021571048
Estimators =  100 Train Score 0.9209269266548903 test Score 0.8940017137960582
Estimators =  250 Train Score 0.920510406861207 test Score 0.9128742451976688
```

```
Estimators =  450 Train Score 0.9207039250930067 test Score 0.9158117775400172
```

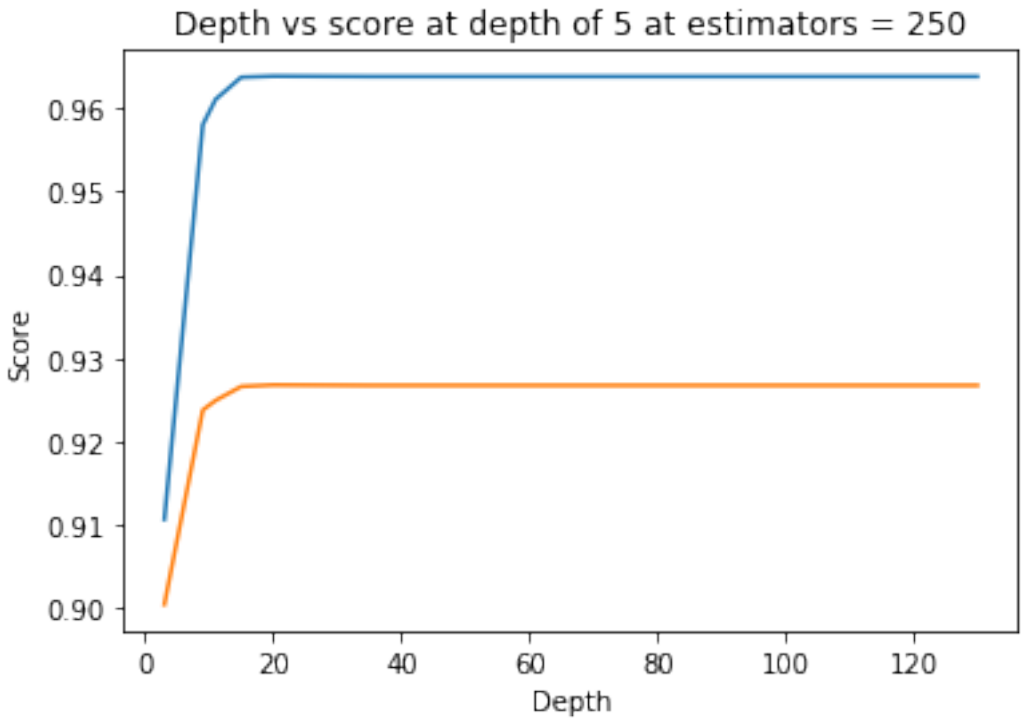
Out[53]: Text(0.5, 1.0, 'Estimators vs score at depth of 5')




```
In [56]: 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=250, n_jobs=-1,random_state=25
             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 = 250')
        plt.show()
```

```
depth =  3 Train Score 0.9106114219881501 test Score 0.9003798808461171
depth =  9 Train Score 0.9580498750573715 test Score 0.9237484711737168
depth =  11 Train Score 0.9610948667966212 test Score 0.9249016845071607
depth =  15 Train Score 0.9637553920324791 test Score 0.9265817454468892
depth =  20 Train Score 0.9638803727691715 test Score 0.9267656036206715
depth =  35 Train Score 0.9638380578306894 test Score 0.9267132513891226
depth =  50 Train Score 0.9638380578306894 test Score 0.9267132513891226
depth =  70 Train Score 0.9638380578306894 test Score 0.9267132513891226
depth =  130 Train Score 0.9638380578306894 test Score 0.9267132513891226
```



In [58]:
```
        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(220,250),
                      "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 [0.9619036  0.96238563 0.96008592 0.96151424 0.96270879]
mean train scores [0.96305705 0.96338713 0.96079672 0.96233091 0.96402254]
```

```
In [59]: 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=242, n_jobs=-1,
            oob_score=False, random_state=25, verbose=0, warm_start=False)
```

```
In [60]: 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=242, n_jobs=-1,
                    oob_score=False, random_state=25, verbose=0, warm_start=False)
```

```
In [61]: clf.fit(df_final_train,y_train)
         y_train_pred = clf.predict(df_final_train)
         y_test_pred = clf.predict(df_final_test)
```

```
In [62]: 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 0.9642723057097341
Test f1 score 0.9265776954489959
```

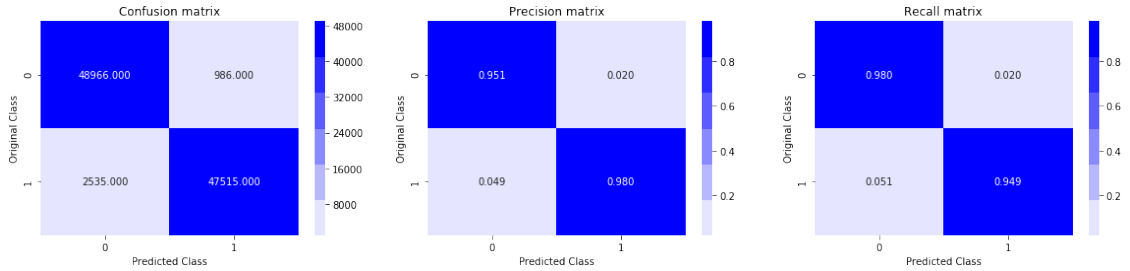
Confusion Matrix

```
In [63]: from sklearn.metrics import confusion_matrix
         def plot_confusion_matrix(test_y, predict_y):
             C = confusion_matrix(test_y, predict_y)
```

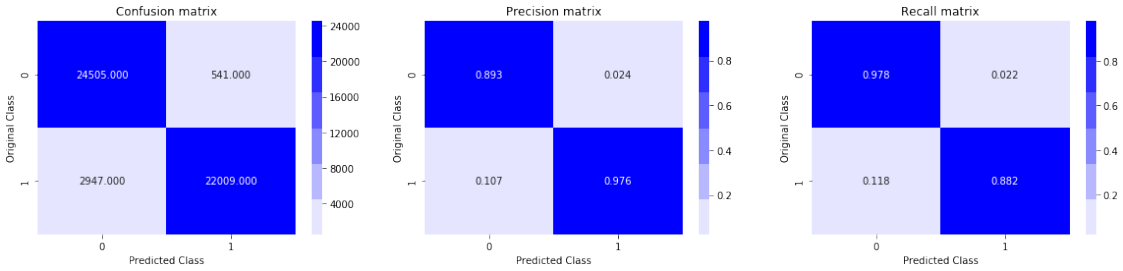
```
    A =(((C.T)/(C.sum(axis=1))).T)

    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=l
    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=l
    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=l
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()
```

In [64]:
```
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
```

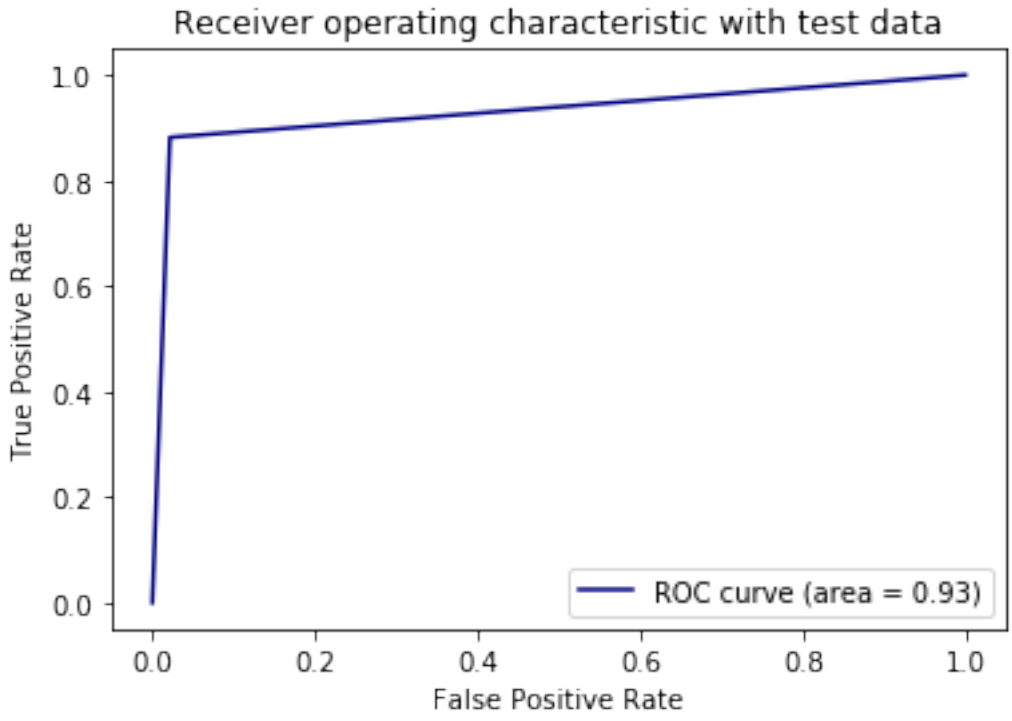


Test confusion_matrix

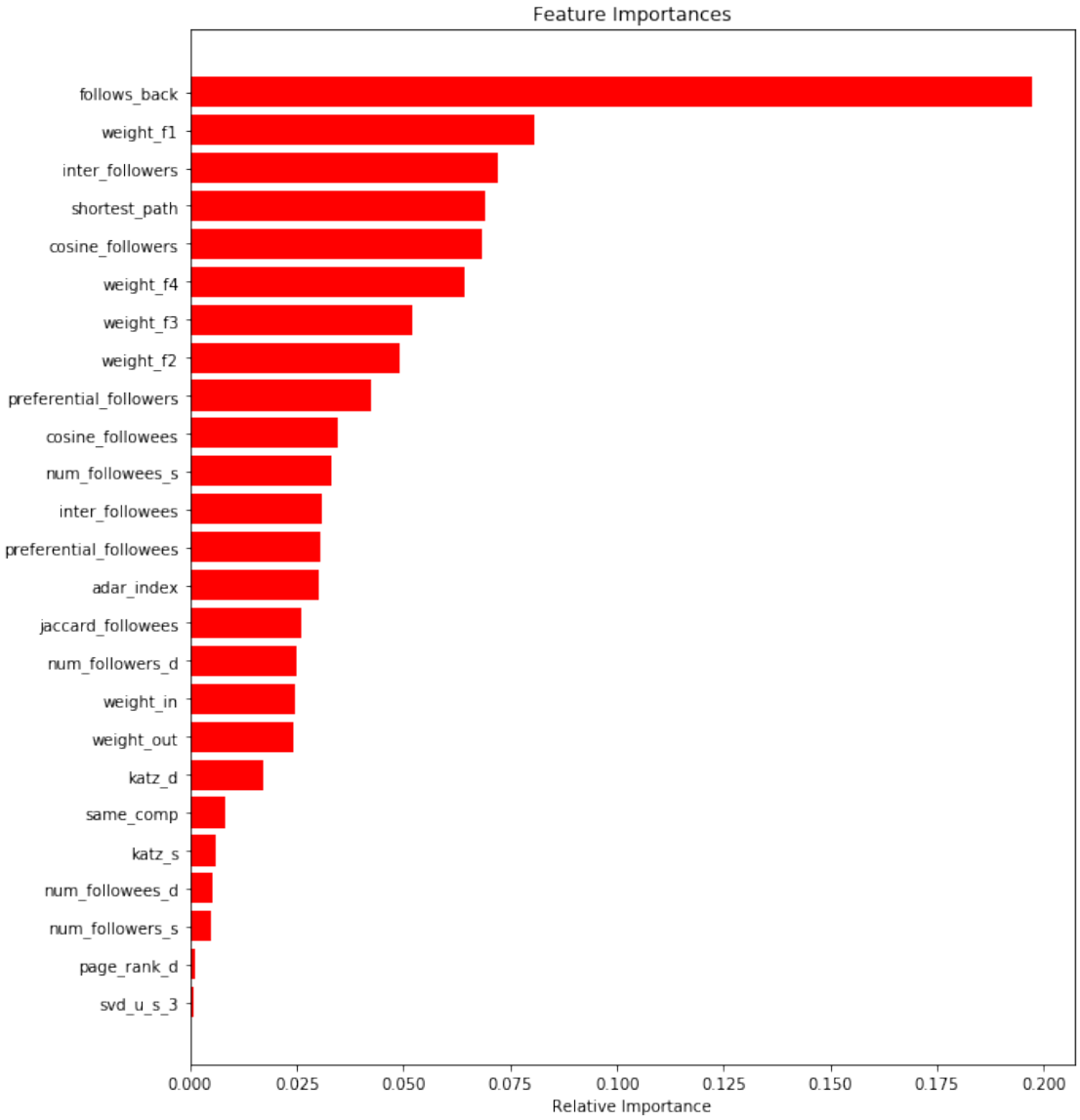


ROC/AUC Curve

```
In [65]: 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 [66]:
```
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()
```



4.2 6.2 XGBOOST (Tuning)

```
In [67]: import xgboost as xgb
         clf = xgb.XGBClassifier()
         param_dist = {"n_estimators":sp_randint(220,250),
                       "max_depth": sp_randint(10,15)
                       }
         model = RandomizedSearchCV(clf, param_distributions=param_dist,
                                            n_iter=5,cv=3,scoring='f1',random_state=25)


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

```
mean test scores [0.98087193 0.98086134 0.98145673 0.98086129 0.98080878]
mean train scores [1. 1. 1. 1. 1.]
```

```
In [68]: print(model.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=248, 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 [76]: clf=xgb.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=10, min_child_weight=1, missing=None, n_estimators=240,
                n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
                reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
                silent=True, subsample=1)
```

```
In [77]: clf.fit(df_final_train,y_train)
         y_train_pred = clf.predict(df_final_train)
         y_test_pred = clf.predict(df_final_test)
```

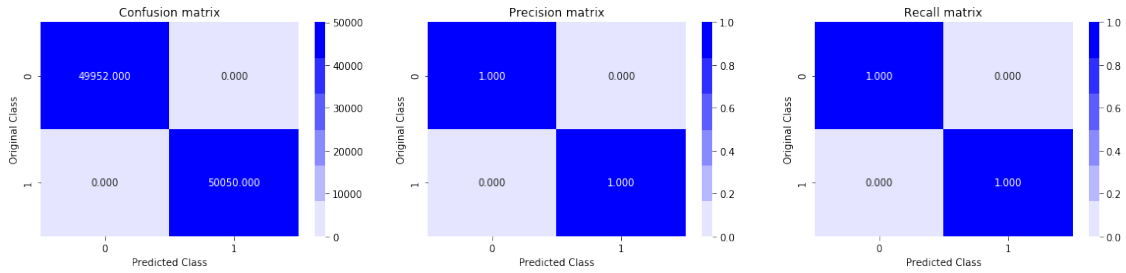
```
In [78]: from sklearn.metrics import f1_score
         print('Train f1 score',f1_score(y_train,y_train_pred))
         print('Test f1 score',f1_score(y_test,y_test_pred))
```

```
Train f1 score 1.0
Test f1 score 0.9266520806338552
```

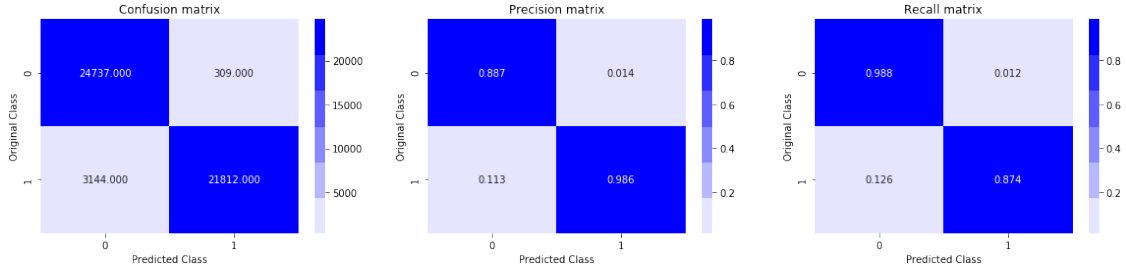
Confusion Matrix

```
In [79]: 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
```

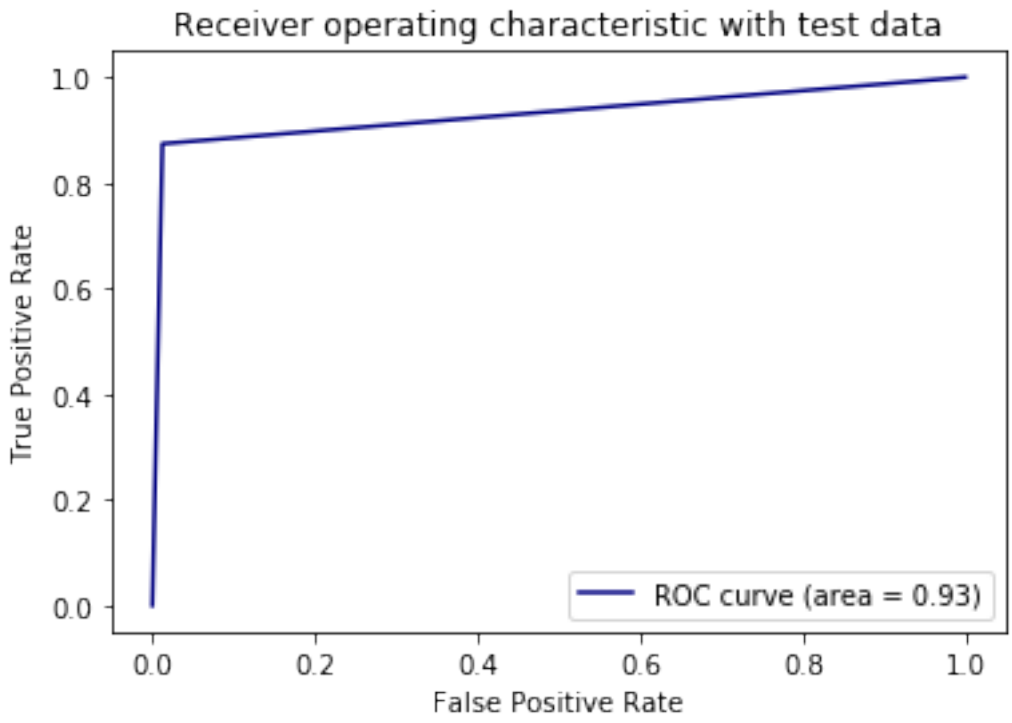



```
Test confusion_matrix
```



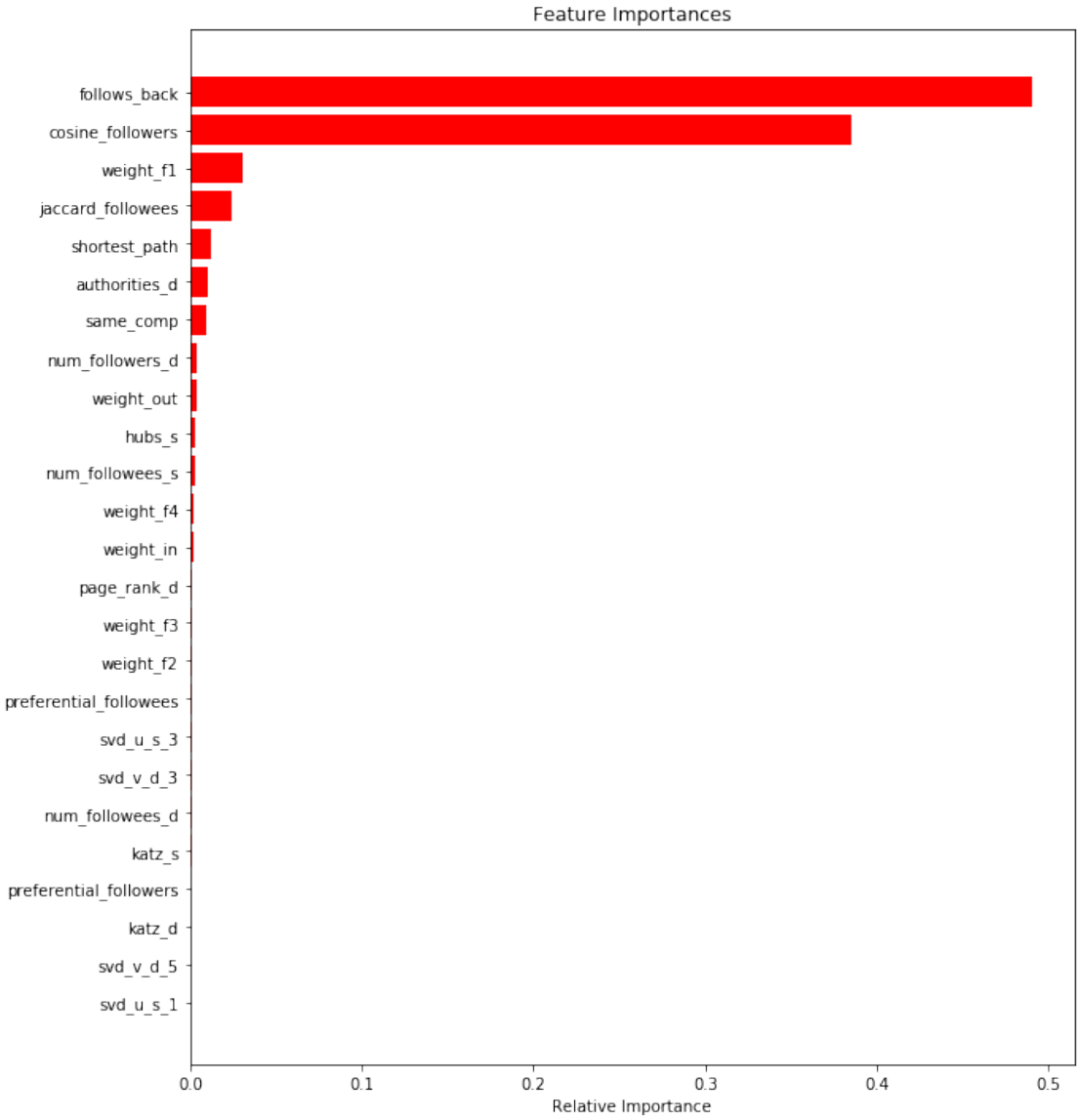
ROC/AUC Curve

```
In [80]: 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 [81]: 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()
```



5 ## Observations:

1. Understanding of graph and feature engineering was the most important part of this case study.
2. For Random Forest, Follow_back was the most important feature found, followed by weight, inter_follower and shortest_path.
3. For XGBOOST, page_rank followed by shortest_path was the most important feature.
4. Best result was obtained in case of XGBOOST.
5. XGBOOST took most of time.
6. For XGBOOST, follows_back was the most important feature. Followed by cosine_follower and weight_f1.

5.1 Summary:

```
In [86]: from prettytable import PrettyTable
         summary = PrettyTable()
         summary.field_names = ["Model", "n_estimators", "max_depth", "Train f1-Score","Test f

In [87]: summary.add_row(['Random Forest','242','14','0.964','0.926'])
         summary.add_row(['XGBOOST','248','10','1','0.926'])
         print(summary)
```

```
+---------------+--------------+-----------+----------------+---------------+
|     Model     | n_estimators | max_depth | Train f1-Score | Test f1-Score |
+---------------+--------------+-----------+----------------+---------------+
| Random Forest |     242      |     14    |     0.964      |     0.926     |
|    XGBOOST    |     248      |     10    |       1        |     0.926     |
+---------------+--------------+-----------+----------------+---------------+
```

6 ## Case Study Flow:

1. The dataset provided is directed graph data
2. For the given dataset, We have approx. 1.86M nodes and 9.43M edges.
3. Data was obtained from kaggle. You can get data from here https://www.kaggle.com/c/FacebookRecruiting
4. We have provided only connected nodes. i.e. 9.43M edges. But for each user among n user's, there is n-1 edges. So, for n nodes total possible edges are of 10^12 order.
5. On EDA, it is found that, number of followers are less than 12 for 90% of users.
6. The given dataset was highly imbalanced , as only one classification label is present.
7. We decided y = 0 , is link is not present and took random sample from it.
8. In training and test dataset were exactly balanced.
9. Featurization is the most important part of this case study. We extracted various features types of features...
    - Similarity measures
    - Ranking Measure
    - Various Graph Features
    - Various Weight Features
    - SVD features using Adjancy matrix. (n_components = 6)
10. We Trained two models Random Forest and XGBOOST
11. XGBOST took most time for run.

```
In [ ]:
```