Social network Graph Link Prediction - Facebook Challenge

Problem statement:

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

Data Overview

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

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

Mapping the problem into supervised learning problem:

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

- https://kaggle2.blob.core.windows.net/forum-message-attachments/2594/supervised_link_prediction.pdf
- https://www.youtube.com/watch?v=2M77Hgy17cg

Business objectives and constraints:

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

Performance metric for supervised learning:

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

In [1]:

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

import csv
import pandas as pd#pandas to create small dataframes
import datetime #Convert to unix time
import time #Convert to unix time
# if numpy is not installed already : pip3 install numpy
import numpy as np#Do 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
from sklearn.model_selection import train_test_split
import math
import pickle
```

```
import os
# to install xgboost: pip3 install xgboost
import xgboost as xgb

import warnings
import networkx as nx
import pdb
import pickle

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

from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score, auc, roc_auc_score, roc_curve
from xgboost import XGBClassifier
pd.set_option('max_colwidth', 800)
```

In [2]:

```
#reading graph
if not os.path.isfile('data/after_eda/train_woheader.csv'):
    traincsv = pd.read_csv('data/train.csv')
    print(traincsv[traincsv.isna().any(1)])
    print(traincsv.info())
    print("Number of diplicate entries: ",sum(traincsv.duplicated()))
    traincsv.to_csv('data/after_eda/train_woheader.csv',header=False,in
dex=False)
    print("saved the graph into file")

g=nx.read_edgelist('data/after_eda/train_woheader.csv',delimiter=',',cr
eate_using=nx.DiGraph(),nodetype=int)
print(nx.info(g))
```

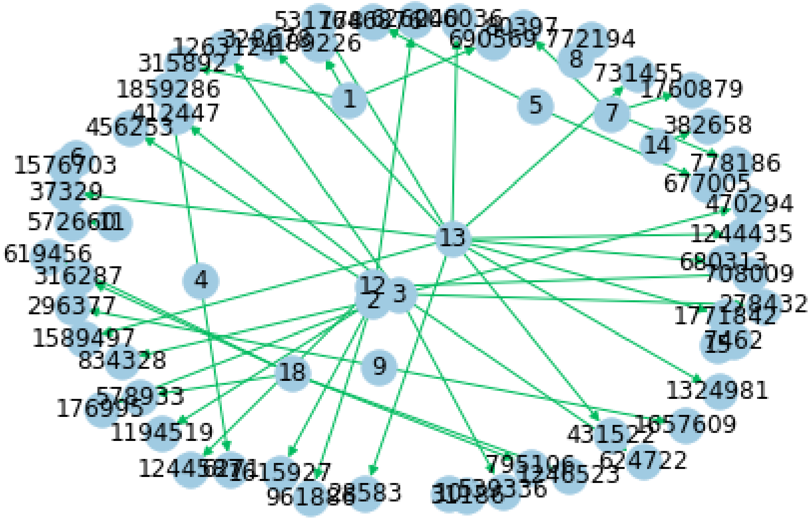
```
Name:
Type: DiGraph
Number of nodes: 1862220
```

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

> Displaying a sub graph

In [3]:
```
if not os.path.isfile('train_woheader_sample.csv'):
    pd.read_csv('data/train.csv', nrows=50).to_csv('train_woheader_samp
le.csv',header=False,index=False)

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

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

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



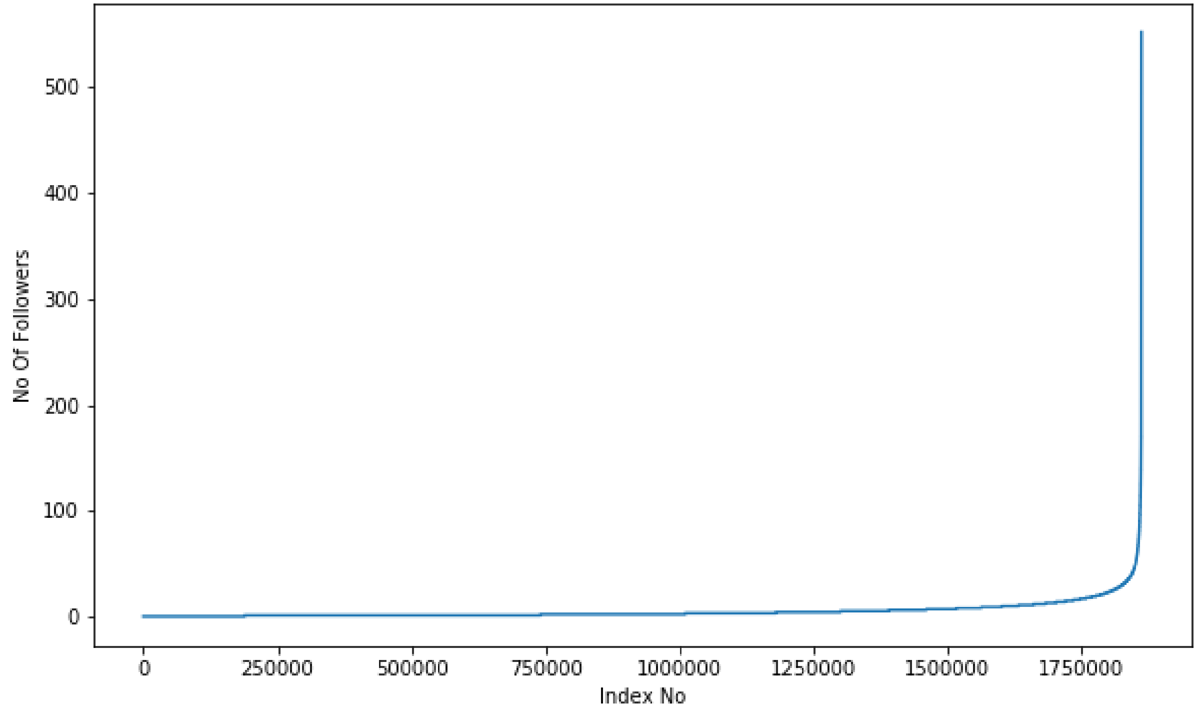
1. Exploratory Data Analysis

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

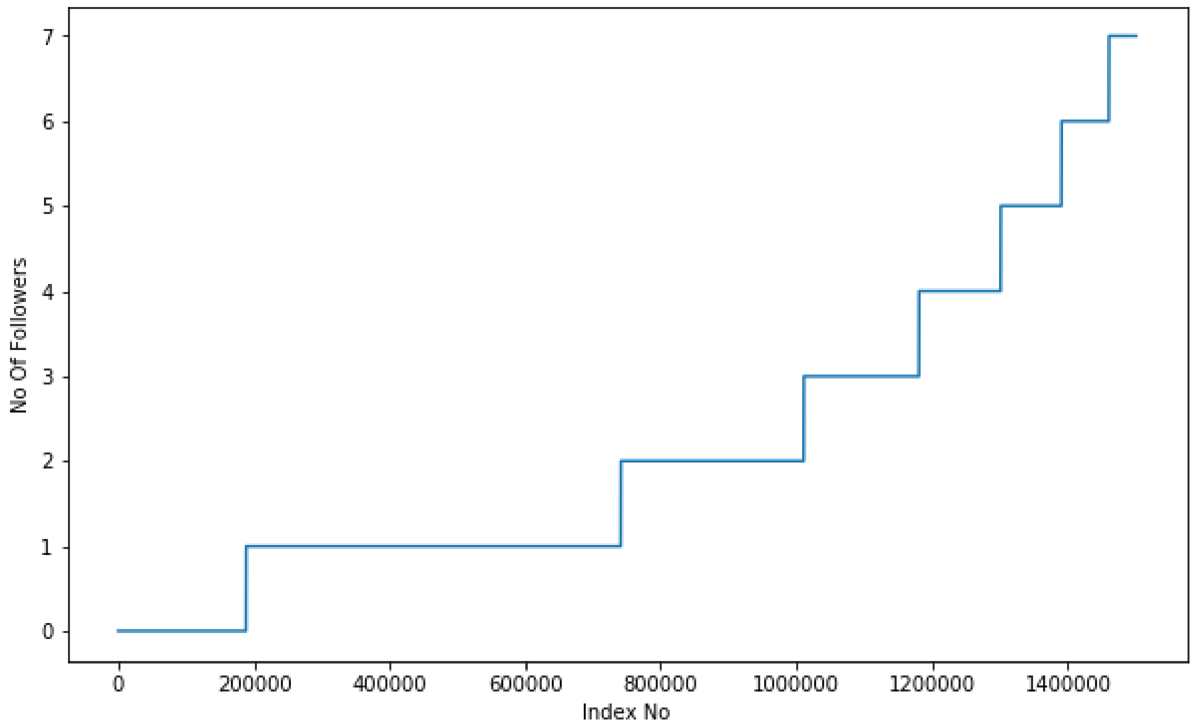
```
The number of unique persons 1862220
```

1.1 No of followers for each person

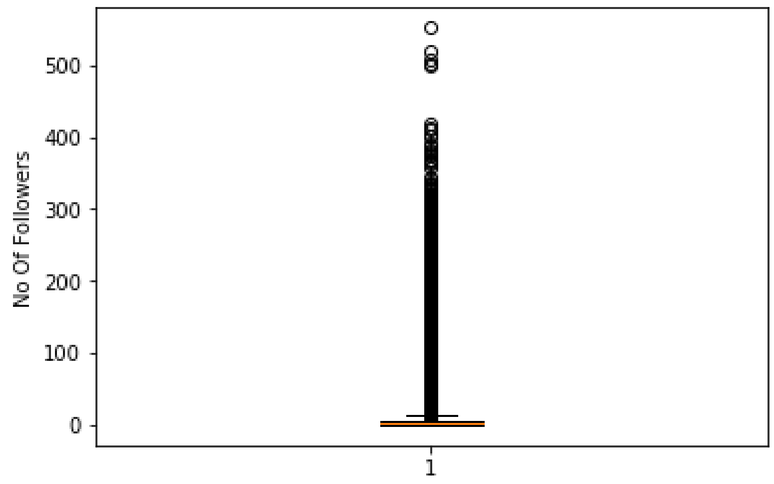
In [5]:
```
indegree_dist = list(dict(g.in_degree()).values())
indegree_dist.sort()
plt.figure(figsize=(10,6))
plt.plot(indegree_dist)
plt.xlabel('Index No')
plt.ylabel('No Of Followers')
plt.show()
```



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



In [7]:
```
plt.boxplot(indegree_dist)
plt.ylabel('No Of Followers')
plt.show()
```



In [8]:
```
### 90-100 percentile
for i in range(0,11):
    print(90+i,'percentile value is',np.percentile(indegree_dist,90+i))
```

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

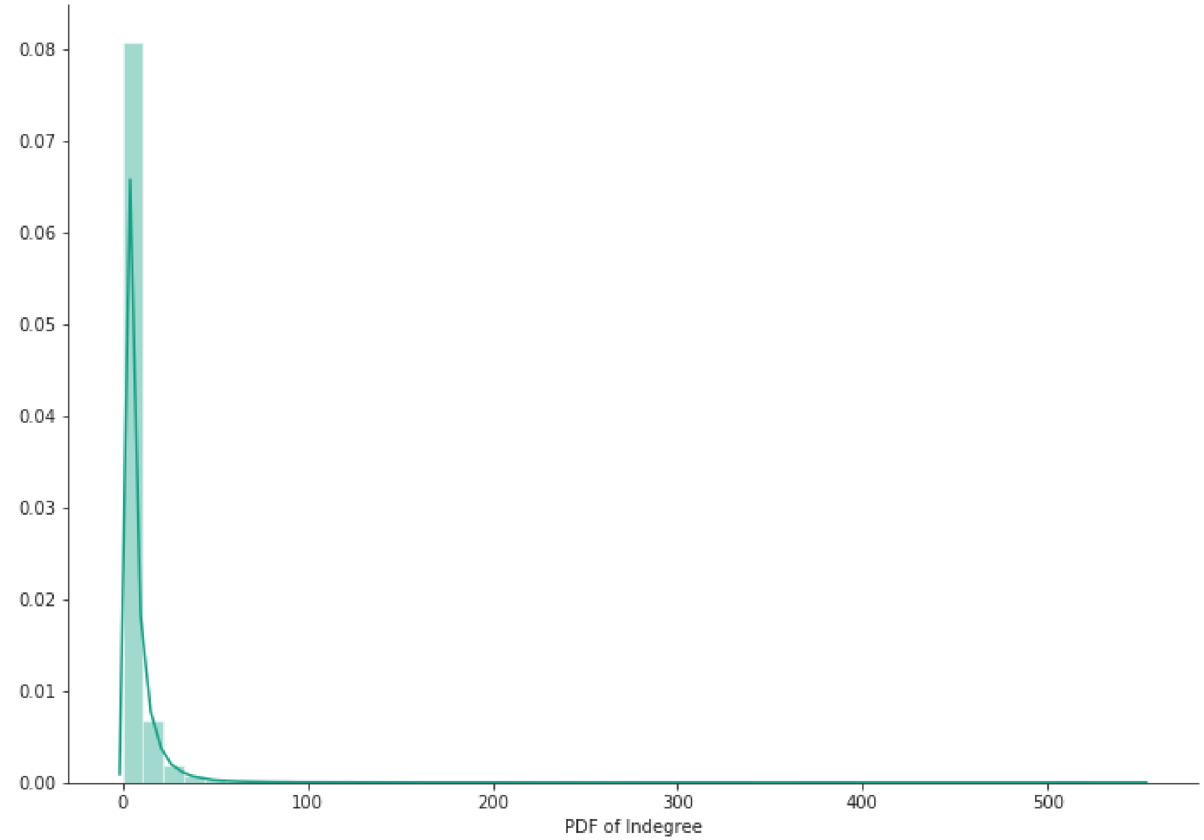
99% of data having followers of 40 only.

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

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

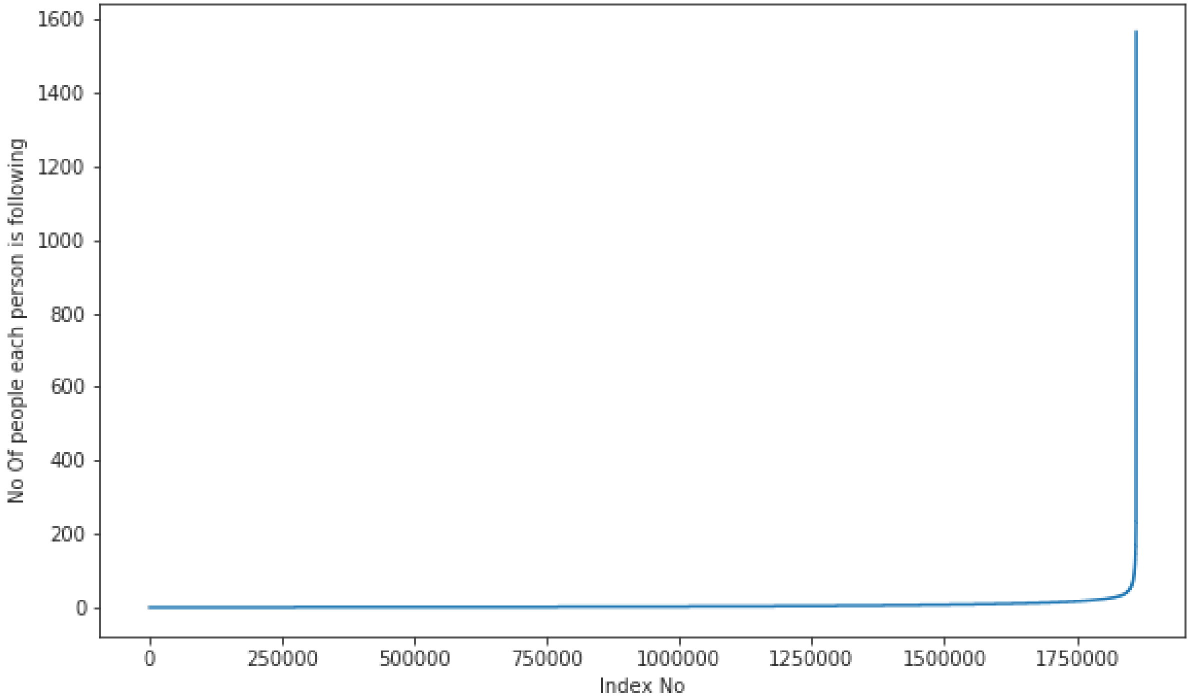
In [10]:

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



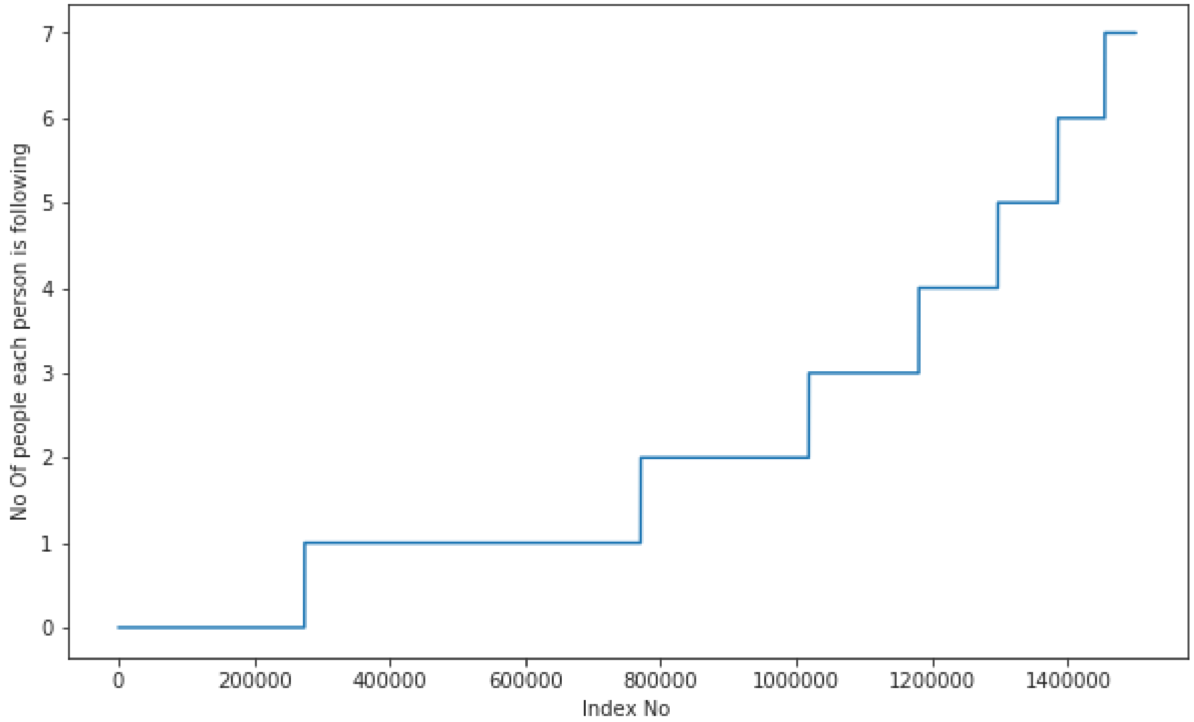
1.2 No of people each person is following

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

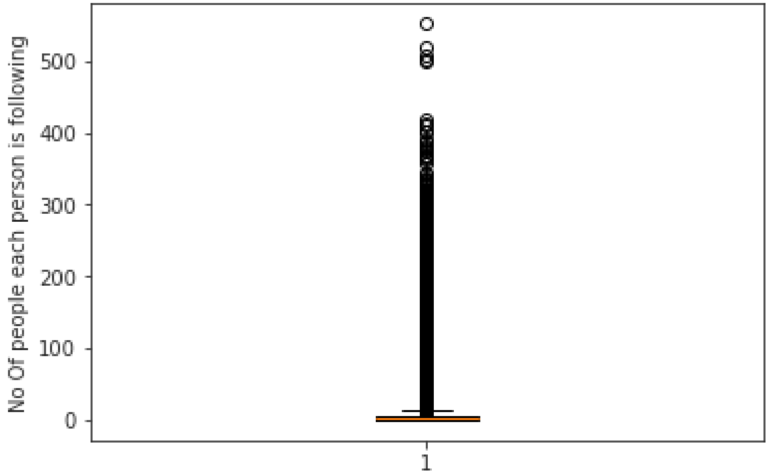


In [12]:

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



In [13]:
```
plt.boxplot(indegree_dist)
plt.ylabel('No Of people each person is following')
plt.show()
```



In [14]:

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

```
90 percentile value is 12.0
91 percentile value is 13.0
92 percentile value is 14.0
93 percentile value is 15.0
94 percentile value is 17.0
95 percentile value is 19.0
96 percentile value is 21.0
97 percentile value is 24.0
98 percentile value is 29.0
99 percentile value is 40.0
100 percentile value is 1566.0
```

In [15]:

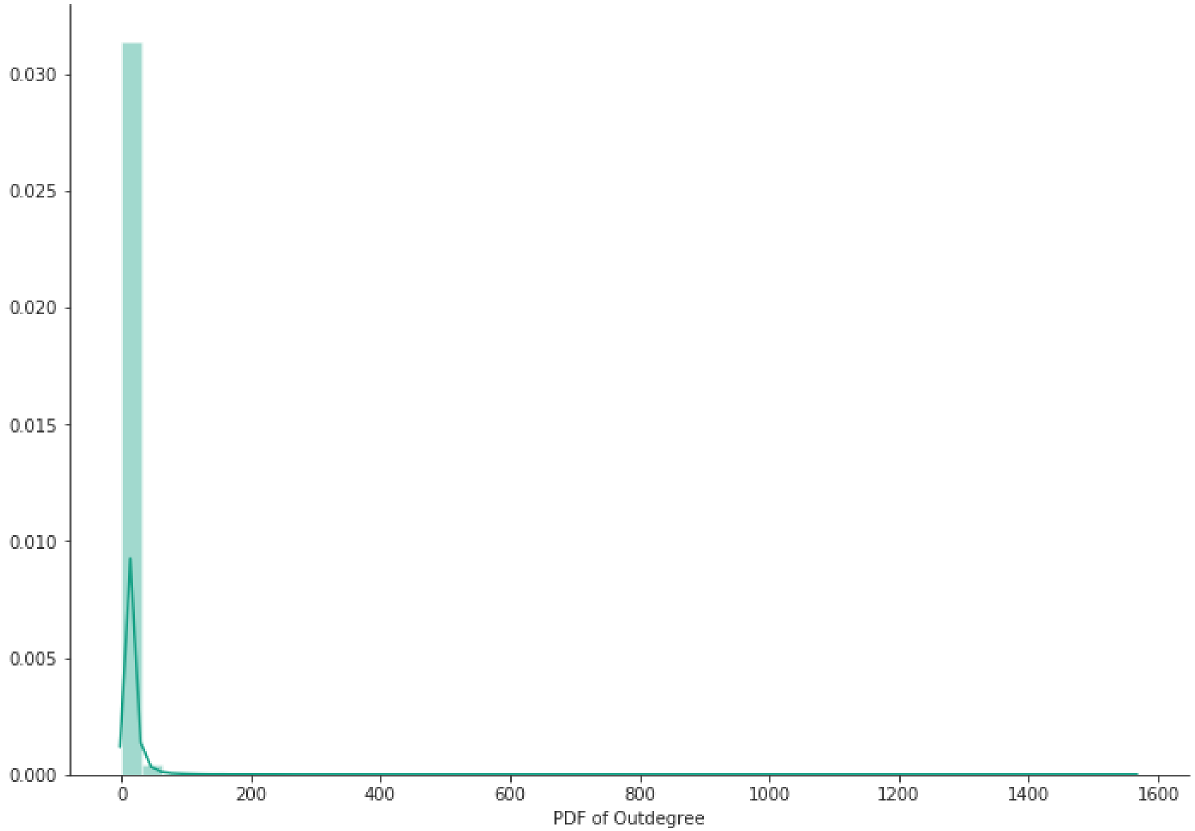
```
### 99-100 percentile
for i in range(10,110,10):
    print(99+(i/100),'percentile value is',np.percentile(outdegree_dist
,99+(i/100)))
```

```
99.1 percentile value is 42.0
99.2 percentile value is 45.0
```

```
99.3 percentile value is 48.0
99.4 percentile value is 52.0
99.5 percentile value is 56.0
99.6 percentile value is 63.0
99.7 percentile value is 73.0
99.8 percentile value is 90.0
99.9 percentile value is 123.0
100.0 percentile value is 1566.0
```

In [16]:

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



In [17]:
```
print('No of persons those are not following anyone are' ,sum(np.array(outdegree_dist)==0),'and % is',
                                sum(np.array(outdegree_dist)==0)*100/len(outdegree_dist) )
```

```
No of persons those are not following anyone are 274512 and % is 14.741115442858524
```

In [18]:
```
print('No of persons having zero followers are' ,sum(np.array(indegree_dist)==0),'and % is',
                                sum(np.array(indegree_dist)==0)*100/len(indegree_dist) )
```

```
No of persons having zero followers are 188043 and % is 10.097786512871
```

No of persons having zero followers are 188043 and % is 10.097786512871
734

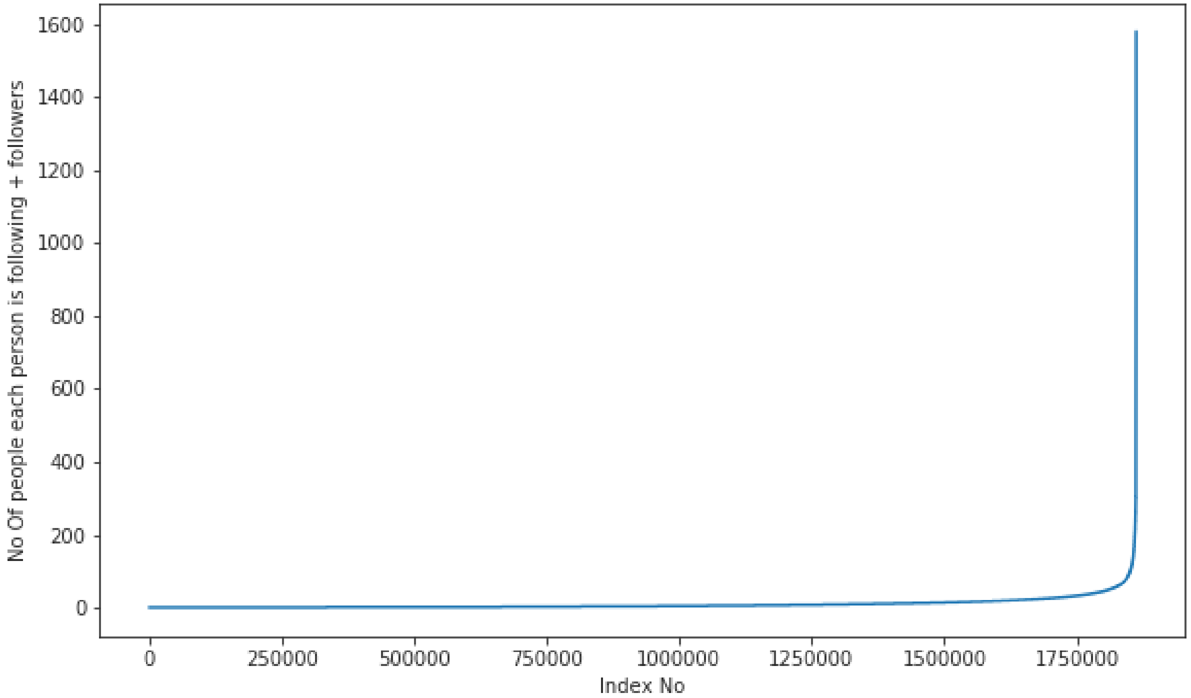
In [19]:
```
count=0
for i in g.nodes():
    if len(list(g.predecessors(i)))==0 :
        if len(list(g.successors(i)))==0:
            count+=1
print('No of persons those are not not following anyone and also not ha
ving any followers are',count)
```

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

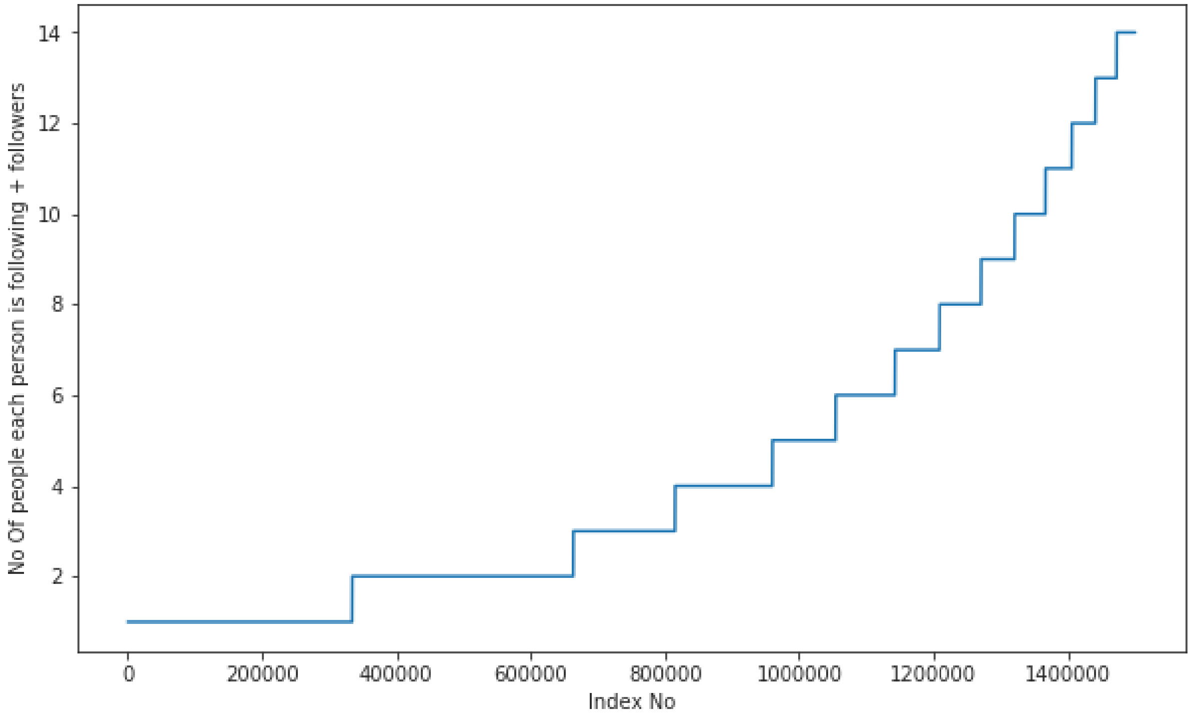
1.3 both followers + following

In [20]:
```
from collections import Counter
dict_in = dict(g.in_degree())
dict_out = dict(g.out_degree())
d = Counter(dict_in) + Counter(dict_out)
in_out_degree = np.array(list(d.values()))
```

In [21]:
```
in_out_degree_sort = sorted(in_out_degree)
plt.figure(figsize=(10,6))
plt.plot(in_out_degree_sort)
plt.xlabel('Index No')
plt.ylabel('No Of people each person is following + followers')
plt.show()
```



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



In [23]:
```
### 90-100 percentile
for i in range(0,11):
    print(90+i,'percentile value is',np.percentile(in_out_degree_sort,9
0+i))
```

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

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

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

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

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

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

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

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

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

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

```
for i in list(nx.weakly_connected_components(g)):
    if len(i)==2:
        count+=1
print('weakly connected components wit 2 nodes',count)
```

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

2. Posing a problem as classification problem

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

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

In [29]:

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


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

```
                    if nx.shortest_path_length(g,source=a,target=b) > 2:

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

```
CPU times: user 1.96 s, sys: 1.4 s, total: 3.36 s
Wall time: 3.83 s
```

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

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

2.2 Training and Test data split:

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

In [31]:
```
if (not os.path.isfile('data/after_eda/train_pos_after_eda.csv')) and (
not os.path.isfile('data/after_eda/test_pos_after_eda.csv')):
    #reading total data df
    df_pos = pd.read_csv('data/train.csv')
    df_neg = pd.DataFrame(list(missing_edges), columns=['source_node',
'destination_node'])

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

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

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

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

In [32]:

```
if (os.path.isfile('data/after_eda/train_pos_after_eda.csv')) and (os.p
ath.isfile('data/after_eda/test_pos_after_eda.csv')):
    train_graph=nx.read_edgelist('data/after_eda/train_pos_after_eda.cs
v',delimiter=',',create_using=nx.DiGraph(),nodetype=int)
```

```
    test_graph=nx.read_edgelist('data/after_eda/test_pos_after_eda.csv'
,delimiter=',',create_using=nx.DiGraph(),nodetype=int)
    print(nx.info(train_graph))
    print(nx.info(test_graph))

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

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

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

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

```
Name:
Type: DiGraph
Number of nodes: 1780722
Number of edges: 7550015
Average in degree:   4.2399
Average out degree:   4.2399
Name:
Type: DiGraph
Number of nodes: 1144623
Number of edges: 1887504
Average in degree:   1.6490
Average out degree:   1.6490
no of people common in train and test --  1063125
no of people present in train but not present in test --  717597
no of people present in test but not present in train --  81498
 % of people not there in Train but exist in Test in total Test data ar
e 7.1200735962845405 %
```

we have a cold start problem here

In [33]:

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

    X_train_pos = pd.read_csv('data/after_eda/train_pos_after_eda.csv', names=['source_node', 'destination_node'])
    X_test_pos = pd.read_csv('data/after_eda/test_pos_after_eda.csv', names=['source_node', 'destination_node'])
    X_train_neg = pd.read_csv('data/after_eda/train_neg_after_eda.csv', names=['source_node', 'destination_node'])
    X_test_neg = pd.read_csv('data/after_eda/test_neg_after_eda.csv', 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('data/after_eda/train_after_eda.csv',header=False,in
dex=False)
    X_test.to_csv('data/after_eda/test_after_eda.csv',header=False,inde
x=False)
    pd.DataFrame(y_train.astype(int)).to_csv('data/train_y.csv',header=
False,index=False)
    pd.DataFrame(y_test.astype(int)).to_csv('data/test_y.csv',header=Fa
lse,index=False)
```

In [ ]:

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

1. Reading Data

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.cs
v',delimiter=',',create_using=nx.DiGraph(),nodetype=int)
    print(nx.info(train_graph))
else:
    print("please run the FB_EDA.ipynb or download the files from driv
e")
```

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

2. Similarity measures

2.1 Jaccard Distance:

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

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

In [3]:

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

In [4]:

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

```
0.0
```

In [5]:

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

```
0.0
```

In [6]:

```
#for followers
def jaccard_for_followers(a,b):
    try:
        if len(set(train_graph.predecessors(a))) == 0  | len(set(g.pred
ecessors(b))) == 0:
```

```
            return 0
        sim = (len(set(train_graph.predecessors(a)).intersection(set(tr
ain_graph.predecessors(b)))))/\
                                    (len(set(train_graph.predecessors(a)).
union(set(train_graph.predecessors(b)))))
        return sim
    except:
        return 0
```

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

```
0
```

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

```
0
```

2.2 Cosine distance

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

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

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

```
0.0
```

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

```
0
```

In [12]:
```
def cosine_for_followers(a,b):
    try:

        if len(set(train_graph.predecessors(a))) == 0  | len(set(train_
graph.predecessors(b))) == 0:
            return 0
        sim = (len(set(train_graph.predecessors(a)).intersection(set(tr
ain_graph.predecessors(b)))))/\
                                     (math.sqrt(len(set(train_graph.pre
decessors(a))))*(len(set(train_graph.predecessors(b)))))
        return sim
    except:
        return 0
```

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

```
0.02886751345948129
```

In [14]:
```
print(cosine_for_followers(669354,1635354))
```

```
0
```

2.3 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 (|N(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.

$$PreferentialAttachment = |N(X)| \cdot |N(Y)|$$

For two nodes a,b, the preferential attachment= number of followers(a)* number of followers(b).

In [15]:
```
def preferentialAttachment(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.successors(a)))*len((set(train_graph
.successors(b)))))
        return sim
    except:
        return 0
```

In [16]:
```
print(preferentialAttachment(273084,1505602))
```

```
120
```

In [17]:
```
print(preferentialAttachment(273084,1635354))
```

```
0
```

3. Ranking Measures

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

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

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

3.1 Page Ranking

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

In [18]:
```
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 [19]:
```
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 2.7098251341935817e-05
mean 5.615699699365892e-07
```

In [20]:
```
#for imputing to nodes which are not there in Train data
mean_pr = float(sum(pr.values())) / len(pr)
print(mean_pr)
```

```
5.615699699365892e-07
```

4. Other Graph Features

4.1 Shortest path:

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

In [21]:
```
#if has direct edge then deleting that edge and calculating shortest pa
th
def compute_shortest_path_length(a,b):
    p=-1
    try:
        if train_graph.has_edge(a,b):
            train_graph.remove_edge(a,b)
            p= nx.shortest_path_length(train_graph,source=a,target=b)
            train_graph.add_edge(a,b)
        else:
            p= nx.shortest_path_length(train_graph,source=a,target=b)
        return p
    except:
        return -1
```

In [22]:
```
#testing
compute_shortest_path_length(77697, 826021)
```

Out[22]:
```
10
```

In [23]:
```
#testing
compute_shortest_path_length(669354,1635354)
```

Out[23]:
```
-1
```

4.2 Checking for same community

In [24]:
```
#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 [25]:
```
belongs_to_same_wcc(861, 1659750)
```

Out[25]: 0

In [26]:
```
belongs_to_same_wcc(669354,1635354)
```
Out[26]: 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 [27]:
```
#adar index
def calc_adar_in(a,b):
    sum=0
    try:
        n=list(set(train_graph.successors(a)).intersection(set(train_gr
aph.successors(b))))
        if len(n)!=0:
            for i in n:
                sum=sum+(1/np.log10(len(list(train_graph.predecessors(i
)))))
            return sum
        else:
            return 0
    except:
        return 0
```

In [28]:
```
calc_adar_in(1,189226)
```
Out[28]: 0

In [29]:
```
calc_adar_in(669354,1635354)
```
Out[29]: 0

4.4 Is persion was following back:

In [30]:
```
def follows_back(a,b):
    if train_graph.has_edge(b,a):
        return 1
    else:
        return 0
```

In [31]:
```
follows_back(1,189226)
```

Out[31]: 1

In [32]:
```
follows_back(669354,1635354)
```

Out[32]: 0

4.5 Katz Centrality:

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

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

$$x_i = \alpha \sum_j A_{ij} x_j + \beta,$$

where `A` is the adjacency matrix of the graph G with eigenvalues

$$\lambda$$

.

The parameter

$$\beta$$

controls the initial centrality and

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

In [33]:
```
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 [34]:
```
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 0.0007313532484062579
max 0.003394554981697573
mean 0.0007483800935501884
```

In [35]:
```
mean_katz = float(sum(katz.values())) / len(katz)
print(mean_katz)
```

```
0.0007483800935501884
```

4.6 Hits Score

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

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

In [36]:
```
if not os.path.isfile('data/fea_sample/hits.p'):
    hits = nx.hits(train_graph, max_iter=100, tol=1e-08, nstart=None, n
ormalized=True)
    pickle.dump(hits,open('data/fea_sample/hits.p','wb'))
else:
    hits = pickle.load(open('data/fea_sample/hits.p','rb'))
```

In [37]:
```
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 0.004868653378766884
mean 5.615699699337049e-07
```

5. Featurization

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

In [38]:
```
import random
if os.path.isfile('data/after_eda/train_after_eda.csv'):
    filename = "data/after_eda/train_after_eda.csv"
    # you uncomment this line, if you dont know the lentgh of the file name
    # here we have hardcoded the number of lines as 15100030
    # n_train = sum(1 for line in open(filename)) #number of records in file (excludes header)
    n_train =  15100028
    s = 100000 #desired sample size
    skip_train = sorted(random.sample(range(1,n_train+1),n_train-s))
    #https://stackoverflow.com/a/22259008/4084039
```

In [39]:
```
if os.path.isfile('data/after_eda/train_after_eda.csv'):
    filename = "data/after_eda/test_after_eda.csv"
    # you uncomment this line, if you dont know the lentgh of the file name
    # here we have hardcoded the number of lines as 3775008
    # n_test = sum(1 for line in open(filename)) #number of records in file (excludes header)
    n_test = 3775006
    s = 50000 #desired sample size
```

```
    skip_test = sorted(random.sample(range(1,n_test+1),n_test-s))
    #https://stackoverflow.com/a/22259008/4084039
```

In [40]:

```
print("Number of rows in the train data file:", n_train)
print("Number of rows we are going to elimiate in train data are",len(s
kip_train))
print("Number of rows in the test data file:", n_test)
print("Number of rows we are going to elimiate in test data are",len(sk
ip_test))
```

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

In [41]:

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

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

Out[41]:

	source_node	destination_node	indicator_link
0	273084	1505602	1
1	1757093	912379	1

In [42]:

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

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

Out[42]:

	source_node	destination_node	indicator_link
0	848424	784690	1
1	1227220	827479	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. preferentialAttachment
6. num_followers_s
7. num_followees_s
8. num_followers_d
9. num_followees_d
10. inter_followers
11. inter_followees

In [43]:

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

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

```
ow:
                                            jaccard_for_followees(row[
'source_node'],row['destination_node']),axis=1)
    df_final_test['jaccard_followees'] = df_final_test.apply(lambda row
:
                                            jaccard_for_followees(row[
'source_node'],row['destination_node']),axis=1)


    #mapping cosine followers to train and test data
    df_final_train['cosine_followers'] = df_final_train.apply(lambda ro
w:
                                            cosine_for_followers(row['s
ource_node'],row['destination_node']),axis=1)
    df_final_test['cosine_followers'] = df_final_test.apply(lambda row:
                                            cosine_for_followers(row['s
ource_node'],row['destination_node']),axis=1)

    #mapping cosine followees to train and test data
    df_final_train['cosine_followees'] = df_final_train.apply(lambda ro
w:
                                            cosine_for_followees(row['s
ource_node'],row['destination_node']),axis=1)
    df_final_test['cosine_followees'] = df_final_test.apply(lambda row:
                                            cosine_for_followees(row['s
ource_node'],row['destination_node']),axis=1)

    #mapping preferential Attachment to train and test data
    df_final_train['pref_attachment'] = df_final_train.apply(lambda row
:
                                            preferentialAttachment(row[
'source_node'],row['destination_node']),axis=1)
    df_final_test['pref_attachment'] = df_final_test.apply(lambda row:
                                            preferentialAttachment(row[
'source_node'],row['destination_node']),axis=1)
```

In [44]:

```
def compute_features_stage1(df_final):
    #calculating no of followers followees for source and destination
    #calculating intersection of followers and followees for source and
```

```
 destination
    num_followers_s=[]
    num_followees_s=[]
    num_followers_d=[]
    num_followees_d=[]
    inter_followers=[]
    inter_followees=[]
    for i,row in df_final.iterrows():
        try:
            s1=set(train_graph.predecessors(row['source_node']))
            s2=set(train_graph.successors(row['source_node']))
        except:
            s1 = set()
            s2 = set()
        try:
            d1=set(train_graph.predecessors(row['destination_node']))
            d2=set(train_graph.successors(row['destination_node']))
        except:
            d1 = set()
            d2 = set()
        num_followers_s.append(len(s1))
        num_followees_s.append(len(s2))

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

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

    return num_followers_s, num_followers_d, num_followees_s, num_follo
wees_d, inter_followers, inter_followees
```

In [45]:

```
if not os.path.isfile('data/fea_sample/storage_sample_stage1.h5'):
    df_final_train['num_followers_s'], df_final_train['num_followers_d'
], \
    df_final_train['num_followees_s'], df_final_train['num_followees_d'
], \
    df_final_train['inter_followers'], df_final_train['inter_followees'
]= compute_features_stage1(df_final_train)
```

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

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

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 [46]:

```
if not os.path.isfile('data/fea_sample/storage_sample_stage2.h5'):
    #mapping adar index on train
    df_final_train['adar_index'] = df_final_train.apply(lambda row: cal
c_adar_in(row['source_node'],row['destination_node']),axis=1)
    #mapping adar index on test
    df_final_test['adar_index'] = df_final_test.apply(lambda row: calc_
adar_in(row['source_node'],row['destination_node']),axis=1)
```

```
    #-------------------------------------------------------------------
--------------------------------------
    #mapping followback or not on train
    df_final_train['follows_back'] = df_final_train.apply(lambda row: f
ollows_back(row['source_node'],row['destination_node']),axis=1)

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

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

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

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

    hdf = HDFStore('data/fea_sample/storage_sample_stage2.h5')
    hdf.put('train_df',df_final_train, format='table', data_columns=Tru
e)
    hdf.put('test_df',df_final_test, format='table', data_columns=True)
    hdf.close()
else:
    df_final_train = read_hdf('data/fea_sample/storage_sample_stage2.h
5', 'train_df',mode='r')
```

```
    df_final_test = read_hdf('data/fea_sample/storage_sample_stage2.h5'
, 'test_df',mode='r')
```

5.4 Adding new set of features

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

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

Weight Features

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

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

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

In [64]:
```
#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 [00:16<00:00, 105780.58it/s]
```

In [65]:
```
if not os.path.isfile('data/fea_sample/storage_sample_stage3.h5'):
    #mapping to pandas train
    df_final_train['weight_in'] = df_final_train.destination_node.apply
(lambda x: Weight_in.get(x,mean_weight_in))
    df_final_train['weight_out'] = df_final_train.source_node.apply(lam
bda x: Weight_out.get(x,mean_weight_out))

    #mapping to pandas test
    df_final_test['weight_in'] = df_final_test.destination_node.apply(l
ambda x: Weight_in.get(x,mean_weight_in))
    df_final_test['weight_out'] = df_final_test.source_node.apply(lambd
a 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_t
rain.weight_out
    df_final_train['weight_f2'] = df_final_train.weight_in * df_final_t
rain.weight_out
    df_final_train['weight_f3'] = (2*df_final_train.weight_in + 1*df_fi
nal_train.weight_out)
    df_final_train['weight_f4'] = (1*df_final_train.weight_in + 2*df_fi
nal_train.weight_out)

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

In [66]:
```
if not os.path.isfile('data/fea_sample/storage_sample_stage3.h5'):

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

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

    #Katz centrality score for source and destination in Train and test
    #if anything not there in train graph then adding mean katz score
    df_final_train['katz_s'] = df_final_train.source_node.apply(lambda
```

```
x: katz.get(x,mean_katz))
    df_final_train['katz_d'] = df_final_train.destination_node.apply(la
mbda x: katz.get(x,mean_katz))

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

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

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

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

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

```
    hdf = HDFStore('data/fea_sample/storage_sample_stage3.h5')
    hdf.put('train_df',df_final_train, format='table', data_columns=Tru
e)
    hdf.put('test_df',df_final_test, format='table', data_columns=True)
    hdf.close()
else:
    df_final_train = read_hdf('data/fea_sample/storage_sample_stage3.h
5', 'train_df',mode='r')
    df_final_test = read_hdf('data/fea_sample/storage_sample_stage3.h5'
, 'test_df',mode='r')
```

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 [67]:
```
def svd(x, S):
    try:
        z = sadj_dict[x]
        return S[z]
    except:
        return [0,0,0,0,0,0]
```

In [68]:
```
#for svd features to get feature vector creating a dict node val and in
edx in svd vector
sadj_col = sorted(train_graph.nodes())
sadj_dict = { val:idx for idx,val in enumerate(sadj_col)}
```

In [69]:
```
Adj = nx.adjacency_matrix(train_graph,nodelist=sorted(train_graph.nodes
())).asfptype()
```

In [70]:
```
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 [71]:

```
if not os.path.isfile('data/fea_sample/storage_sample_stage4.h5'):
    #=================================================================
==================================

    df_final_train[['svd_u_s_1', 'svd_u_s_2','svd_u_s_3', 'svd_u_s_4',
'svd_u_s_5', 'svd_u_s_6']] = \
    df_final_train.source_node.apply(lambda x: svd(x, U)).apply(pd.Seri
es)

    df_final_train[['svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3', 'svd_u_d_4',
 'svd_u_d_5','svd_u_d_6']] = \
    df_final_train.destination_node.apply(lambda x: svd(x, U)).apply(pd
.Series)
    #=================================================================
==================================

    df_final_train[['svd_v_s_1','svd_v_s_2', 'svd_v_s_3', 'svd_v_s_4',
'svd_v_s_5', 'svd_v_s_6',]] = \
    df_final_train.source_node.apply(lambda x: svd(x, V.T)).apply(pd.Se
ries)

    df_final_train[['svd_v_d_1', 'svd_v_d_2', 'svd_v_d_3', 'svd_v_d_4',
 'svd_v_d_5','svd_v_d_6']] = \
    df_final_train.destination_node.apply(lambda x: svd(x, V.T)).apply(
pd.Series)
    #=================================================================
==================================

    df_final_test[['svd_u_s_1', 'svd_u_s_2','svd_u_s_3', 'svd_u_s_4',
'svd_u_s_5', 'svd_u_s_6']] = \
    df_final_test.source_node.apply(lambda x: svd(x, U)).apply(pd.Serie
```

```
s)

    df_final_test[['svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3', 'svd_u_d_4',
'svd_u_d_5','svd_u_d_6']] = \
    df_final_test.destination_node.apply(lambda x: svd(x, U)).apply(pd.
Series)

    #===================================================================
=================================

    df_final_test[['svd_v_s_1','svd_v_s_2', 'svd_v_s_3', 'svd_v_s_4',
'svd_v_s_5', 'svd_v_s_6',]] = \
    df_final_test.source_node.apply(lambda x: svd(x, V.T)).apply(pd.Ser
ies)

    df_final_test[['svd_v_d_1', 'svd_v_d_2', 'svd_v_d_3', 'svd_v_d_4',
'svd_v_d_5','svd_v_d_6']] = \
    df_final_test.destination_node.apply(lambda x: svd(x, V.T)).apply(p
d.Series)
    #===================================================================
=================================



    ###### SVD dot is the product between source node svd and destinati
on node svd
    #Train Dataset
    s1,s2,s3,s4,s5,s6=df_final_train['svd_u_s_1'],df_final_train['svd_u
_s_2'],df_final_train['svd_u_s_3'],df_final_train['svd_u_s_4'],df_final
_train['svd_u_s_5'],df_final_train['svd_u_s_6']
    s7,s8,s9,s10,s11,s12=df_final_train['svd_v_s_1'],df_final_train['sv
d_v_s_2'],df_final_train['svd_v_s_3'],df_final_train['svd_v_s_4'],df_fi
nal_train['svd_v_s_5'],df_final_train['svd_v_s_6']

    d1,d2,d3,d4,d5,d6=df_final_train['svd_u_d_1'],df_final_train['svd_u
_d_2'],df_final_train['svd_u_d_3'],df_final_train['svd_u_d_4'],df_final
_train['svd_u_d_5'],df_final_train['svd_u_d_6']
    d7,d8,d9,d10,d11,d12=df_final_train['svd_v_d_1'],df_final_train['sv
d_v_d_2'],df_final_train['svd_v_d_3'],df_final_train['svd_v_d_4'],df_fi
nal_train['svd_v_d_5'],df_final_train['svd_v_d_6']
```

```
    svd_dot=[]
    for i in range(len(np.array(s1))):
        a=[]
        b=[]
        a.append(np.array(s1[i]))
        a.append(np.array(s2[i]))
        a.append(np.array(s3[i]))
        a.append(np.array(s4[i]))
        a.append(np.array(s5[i]))
        a.append(np.array(s6[i]))
        a.append(np.array(s7[i]))
        a.append(np.array(s8[i]))
        a.append(np.array(s9[i]))
        a.append(np.array(s10[i]))
        a.append(np.array(s11[i]))
        a.append(np.array(s12[i]))
        b.append(np.array(d1[i]))
        b.append(np.array(d2[i]))
        b.append(np.array(d3[i]))
        b.append(np.array(d4[i]))
        b.append(np.array(d5[i]))
        b.append(np.array(d6[i]))
        b.append(np.array(d7[i]))
        b.append(np.array(d8[i]))
        b.append(np.array(d9[i]))
        b.append(np.array(d10[i]))
        b.append(np.array(d11[i]))
        b.append(np.array(d12[i]))
        svd_dot.append(np.dot(a,b))
    df_final_train['svd_dot']=svd_dot

    # Test dataset
    s1,s2,s3,s4,s5,s6=df_final_test['svd_u_s_1'],df_final_test['svd_u_s_2'],df_final_test['svd_u_s_3'],df_final_test['svd_u_s_4'],df_final_test['svd_u_s_5'],df_final_test['svd_u_s_6']
    s7,s8,s9,s10,s11,s12=df_final_test['svd_v_s_1'],df_final_test['svd_v_s_2'],df_final_test['svd_v_s_3'],df_final_test['svd_v_s_4'],df_final_test['svd_v_s_5'],df_final_test['svd_v_s_6']
```

```
    d1,d2,d3,d4,d5,d6=df_final_test['svd_u_d_1'],df_final_test['svd_u_d
_2'],df_final_test['svd_u_d_3'],df_final_test['svd_u_d_4'],df_final_tes
t['svd_u_d_5'],df_final_test['svd_u_d_6']
    d7,d8,d9,d10,d11,d12=df_final_test['svd_v_d_1'],df_final_test['svd_
v_d_2'],df_final_test['svd_v_d_3'],df_final_test['svd_v_d_4'],df_final_
test['svd_v_d_5'],df_final_test['svd_v_d_6']

    svd_dot=[]
    for i in range(len(np.array(s1))):
        a=[]
        b=[]
        a.append(np.array(s1[i]))
        a.append(np.array(s2[i]))
        a.append(np.array(s3[i]))
        a.append(np.array(s4[i]))
        a.append(np.array(s5[i]))
        a.append(np.array(s6[i]))
        a.append(np.array(s7[i]))
        a.append(np.array(s8[i]))
        a.append(np.array(s9[i]))
        a.append(np.array(s10[i]))
        a.append(np.array(s11[i]))
        a.append(np.array(s12[i]))
        b.append(np.array(d1[i]))
        b.append(np.array(d2[i]))
        b.append(np.array(d3[i]))
        b.append(np.array(d4[i]))
        b.append(np.array(d5[i]))
        b.append(np.array(d6[i]))
        b.append(np.array(d7[i]))
        b.append(np.array(d8[i]))
        b.append(np.array(d9[i]))
        b.append(np.array(d10[i]))
        b.append(np.array(d11[i]))
        b.append(np.array(d12[i]))
        svd_dot.append(np.dot(a,b))
    df_final_test['svd_dot']=svd_dot
```

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

In [72]:
```
df_final_train.head(2)
```

Out[72]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_foll
0	273084	1505602	1	0	0.0	0.0
1	1757093	912379	1	0	0.0	0.0

2 rows × 57 columns

In [73]:
```
df_final_test.head(2)
```

Out[73]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_foll
0	848424	784690	1	0	0.0	0.0
1	1227220	827479	1	0	0.0	0.0

2 rows × 57 columns

Modelling

In [2]:
```
from sklearn.metrics import confusion_matrix
def plot_confusion_matrix(test_y, predict_y):
```

```
    C = confusion_matrix(test_y, predict_y)

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

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

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

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

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

    plt.show()
```

In [3]:
```
global_result = pd.DataFrame(columns=['Model', 'Hyperparameters', 'Trai
n-F1-Score', 'Test-F1-Score', 'Train-AUC', 'Test-AUC'])
```

In [4]:
```
#reading
```

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

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

Out[5]:
```
Index(['source_node', 'destination_node', 'indicator_link',
       'jaccard_followers', 'jaccard_followees', 'cosine_followers',
       'cosine_followees', 'pref_attachment', 'num_followers_s',
       'num_followers_d', 'num_followees_s', 'num_followees_d',
       'inter_followers', 'inter_followees', 'adar_index', 'follows_back',
       'same_comp', 'shortest_path', 'weight_in', 'weight_out', 'weight_f1',
       'weight_f2', 'weight_f3', 'weight_f4', 'page_rank_s', 'page_rank_d',
       'katz_s', 'katz_d', 'hubs_s', 'hubs_d', 'authorities_s',
       'authorities_d', 'svd_u_s_1', 'svd_u_s_2', 'svd_u_s_3', 'svd_u_s_4',
       'svd_u_s_5', 'svd_u_s_6', 'svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3',
       'svd_u_d_4', 'svd_u_d_5', 'svd_u_d_6', 'svd_v_s_1', 'svd_v_s_2',
       'svd_v_s_3', 'svd_v_s_4', 'svd_v_s_5', 'svd_v_s_6', 'svd_v_d_1',
       'svd_v_d_2', 'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5', 'svd_v_d_6',
       'svd_dot'],
      dtype='object')
```

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

In [7]:
```
df_final_train.drop(['source_node', 'destination_node','indicator_link'],axis=1,inplace=True)
df_final_test.drop(['source_node', 'destination_node','indicator_link'],axis=1,inplace=True)
```

In [8]:
```
print("The shape of Train data :",df_final_train.shape)
print("The shape of Train data :",df_final_test.shape)
```

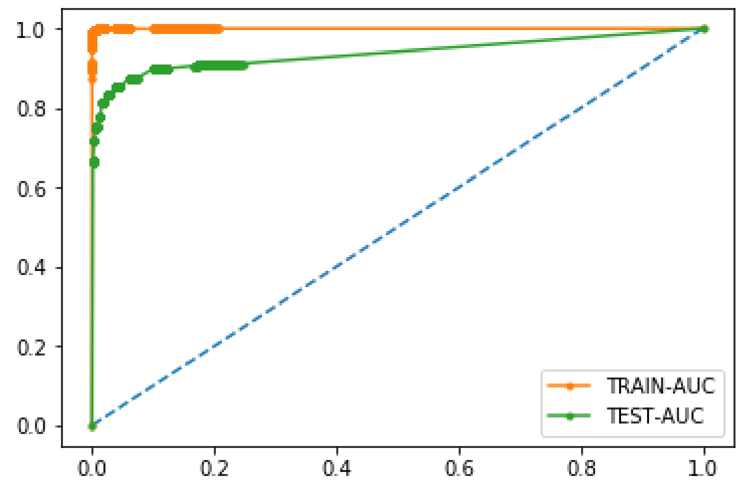
```
The shape of Train data : (100002, 54)
The shape of Train data : (50002, 54)
```

RandomForest Classifier with Default Parameters

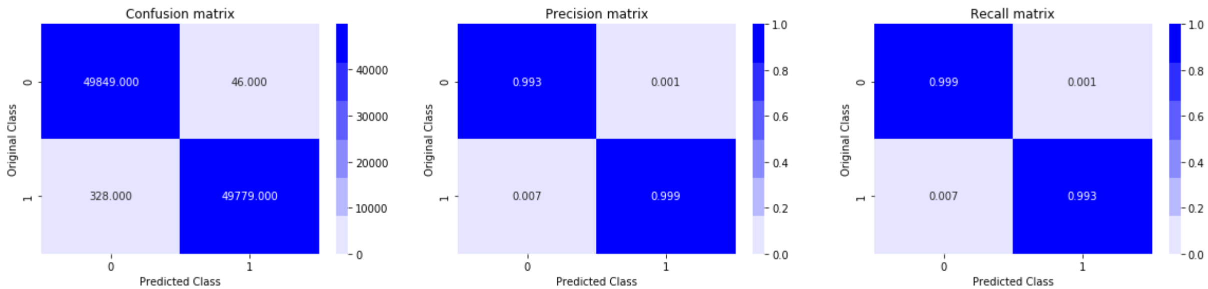
In [9]:
```
clf = RandomForestClassifier(n_jobs=-1, random_state=42)
clf.fit(df_final_train, y_train)

y_pred_train = clf.predict(df_final_train)
y_pred_test = clf.predict(df_final_test)

y_pred_train_proba = clf.predict_proba(df_final_train)[:,1]
y_pred_test_proba = clf.predict_proba(df_final_test)[:,1]

f1_score_val_train = f1_score(y_train, y_pred_train)
roc_auc_val_train = roc_auc_score(y_train, y_pred_train)
f1_score_val_test = f1_score(y_test, y_pred_test)
roc_auc_val_test = roc_auc_score(y_test, y_pred_test)

global_result = global_result.append({'Model': "RandomForest",
                      'Hyperparameters': "Default Parameters",
                      'Train-F1-Score': '{0:.4}'.format(f1_score_val_tr
ain),
                      'Test-F1-Score': '{0:.4}'.format(f1_score_val_tes
t),
                      'Train-AUC': '{0:.4}'.format(roc_auc_val_train),
                      'Test-AUC': '{0:.4}'.format(roc_auc_val_test)
                     }, ignore_index=True)

print("TRAIN Scores","*"*20)
print("\tF1-Score: ", f1_score_val_train)
print("\tROC-AUC : ", roc_auc_val_train)

print("TEST Scores","*"*20)
print("\tF1-Score: ", f1_score_val_test)
print("\tROC-AUC : ", roc_auc_val_test)
```

```
print("===================================== ROC CURVE =================
===================================")
# calculate roc curve
fpr_train, tpr_train, thresholds = roc_curve(y_train, y_pred_train_prob
a)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(fpr_train, tpr_train, marker='.', label='TRAIN-AUC')

fpr_test, tpr_test, thresholds = roc_curve(y_test, y_pred_test_proba)
plt.plot(fpr_test, tpr_test, marker='.', label='TEST-AUC')
# show the plot
plt.legend()
plt.show()
print("=====================================TRAIN CONFUSION MATRIX======
===================================")
plot_confusion_matrix(y_train, y_pred_train)
print("=====================================TEST CONFUSION MATRIX=======
===================================")
plot_confusion_matrix(y_test, y_pred_test)
```

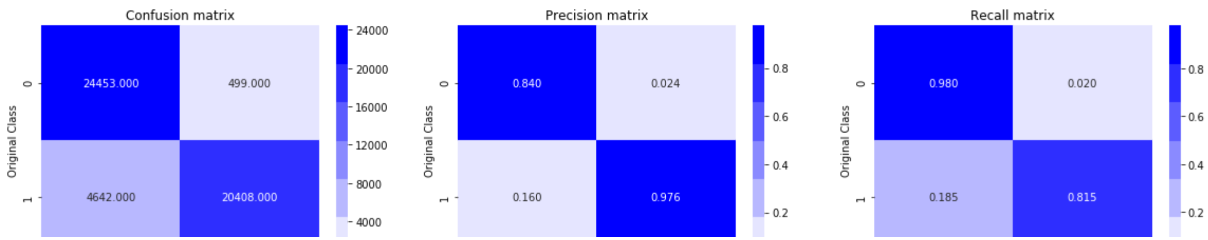
```
TRAIN Scores ********************
        F1-Score:  0.9962574550694472
        ROC-AUC :  0.9962660361781194
TEST Scores ********************
        F1-Score:  0.8881345605674871
        ROC-AUC :  0.8973461109201925
===================================== ROC CURVE ========================
============================
```

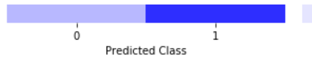


====================================TRAIN CONFUSION MATRIX=============
============================



====================================TEST CONFUSION MATRIX==============
============================





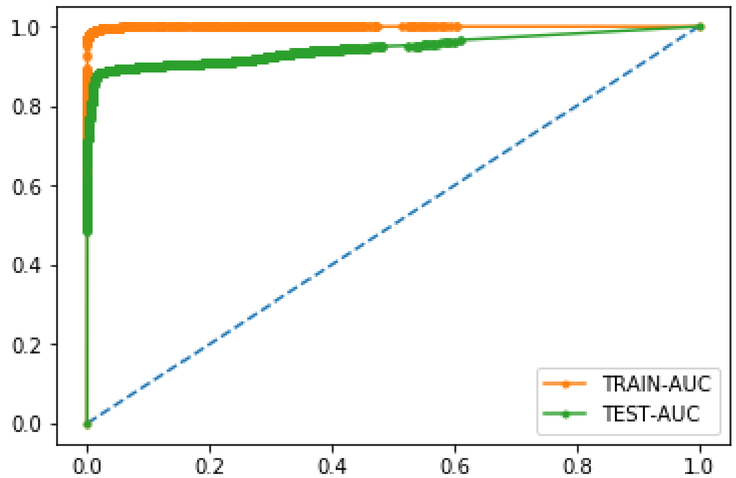
RandomForest Classifier with RandomSearchCV

In [10]:
```
parameters = {
                'n_estimators': [10, 50, 100, 125, 150, 200, 300, 500],

                'min_samples_leaf': [1, 2, 3, 5, 7, 9],
                'min_samples_split': [2, 5, 6, 7, 8, 9]
            }
clf = RandomForestClassifier(n_jobs=-1, random_state=42, oob_score=True
)
r_clf = RandomizedSearchCV(clf, param_distributions=parameters, scoring
='f1', cv=5, random_state=42, n_jobs=-1, return_train_score=True)
r_clf.fit(df_final_train, y_train)

y_pred_train = r_clf.predict(df_final_train)
y_pred_test = r_clf.predict(df_final_test)

y_pred_train_proba = r_clf.predict_proba(df_final_train)[:,1]
y_pred_test_proba = r_clf.predict_proba(df_final_test)[:,1]

f1_score_val_train = f1_score(y_train, y_pred_train)
roc_auc_val_train = roc_auc_score(y_train, y_pred_train)
f1_score_val_test = f1_score(y_test, y_pred_test)
roc_auc_val_test = roc_auc_score(y_test, y_pred_test)

global_result = global_result.append({'Model': "RandomForest",
                      'Hyperparameters': r_clf.best_params_,
                      'Train-F1-Score': '{0:.4}'.format(f1_score_val_tr
ain),
                      'Test-F1-Score': '{0:.4}'.format(f1_score_val_tes
t),
                      'Train-AUC': '{0:.4}'.format(roc_auc_val_train),
                      'Test-AUC': '{0:.4}'.format(roc_auc_val_test)
                     }, ignore_index=True)
```

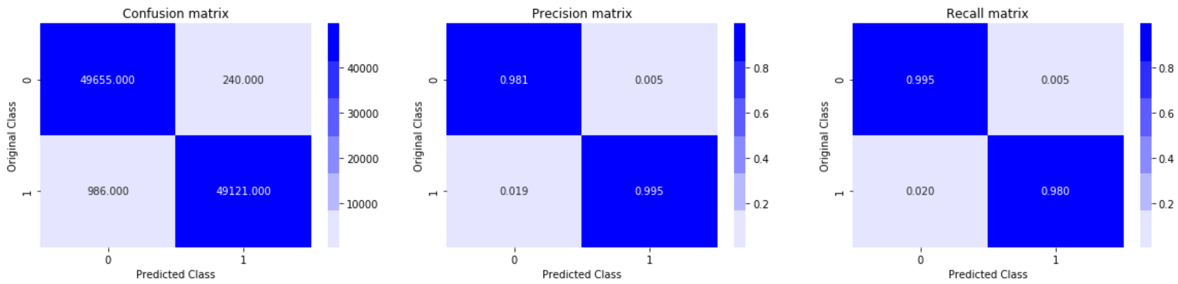
```
print("*"*100)
print("Best Parameters: ", r_clf.best_params_)
print("*"*100)

print("TRAIN Scores","*"*20)
print("\tF1-Score: ", f1_score_val_train)
print("\tROC-AUC : ", roc_auc_val_train)

print("TEST Scores","*"*20)
print("\tF1-Score: ", f1_score_val_test)
print("\tROC-AUC : ", roc_auc_val_test)

print("==================================== ROC CURVE ==================
===================================")
# calculate roc curve
fpr_train, tpr_train, thresholds = roc_curve(y_train, y_pred_train_prob
a)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(fpr_train, tpr_train, marker='.', label='TRAIN-AUC')

fpr_test, tpr_test, thresholds = roc_curve(y_test, y_pred_test_proba)
plt.plot(fpr_test, tpr_test, marker='.', label='TEST-AUC')
# show the plot
plt.legend()
plt.show()
print("====================================TRAIN CONFUSION MATRIX======
===================================")
plot_confusion_matrix(y_train, y_pred_train)
print("====================================TEST CONFUSION MATRIX=======
===================================")
plot_confusion_matrix(y_test, y_pred_test)
```

```
**********************************************************************
******************************
Best Parameters:  {'n_estimators': 200, 'min_samples_split': 9, 'min_
samples_leaf': 1}
**********************************************************************
******************************
TRAIN Scores ********************
	F1-Score:  0.9876744279567297
```

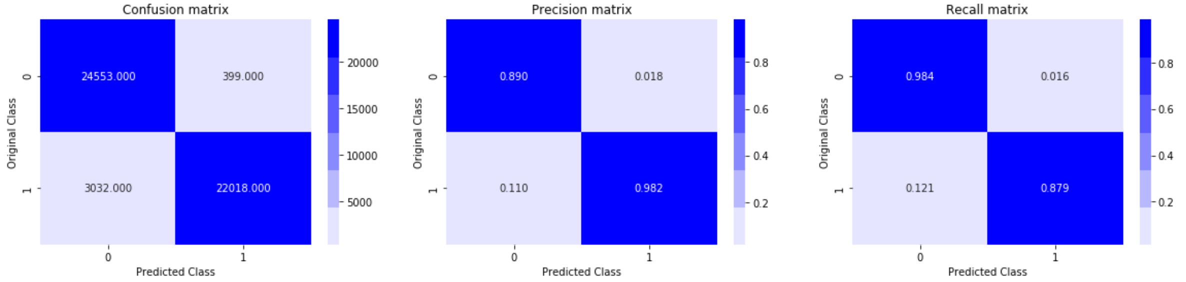
```
        ROC-AUC :  0.9877560047352959
TEST Scores *******************
        F1-Score:  0.927718204226094
        ROC-AUC :  0.9314856868500896
==================================== ROC CURVE =====================
==============================
```




```
====================================TRAIN CONFUSION MATRIX===========
==============================
```




```
====================================TEST CONFUSION MATRIX============
==============================
```

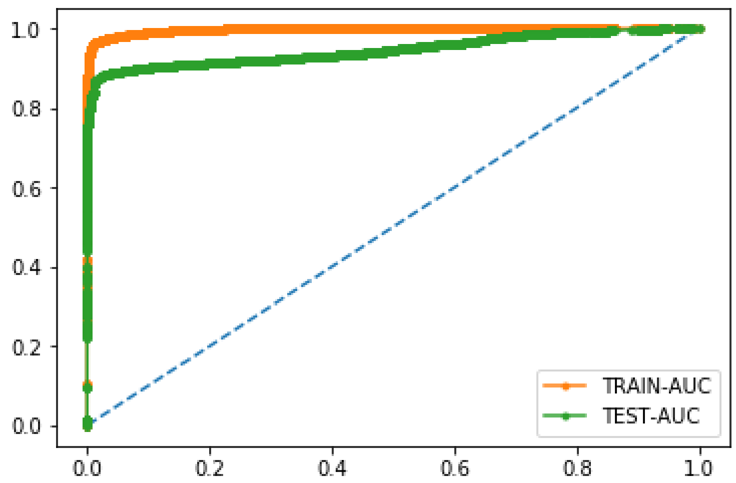


XGBoost Classifier with Default parameters

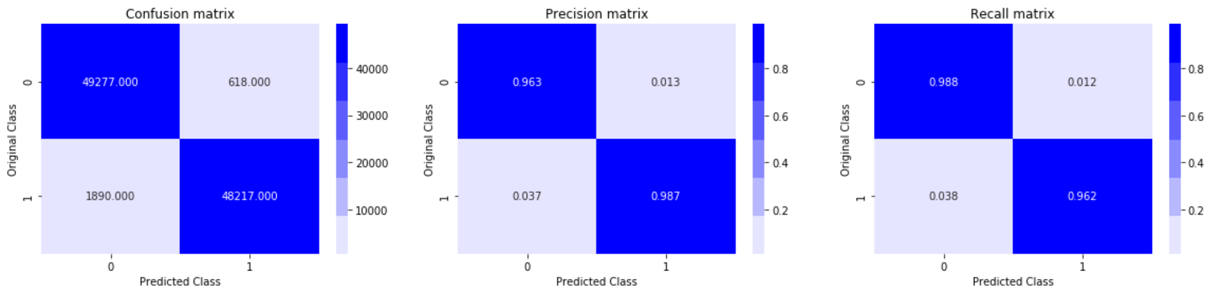
In [11]:
```
clf = XGBClassifier(n_jobs=-1, random_state=42)
clf.fit(df_final_train, y_train)

y_pred_train = clf.predict(df_final_train)
y_pred_test = clf.predict(df_final_test)

y_pred_train_proba = clf.predict_proba(df_final_train)[:,1]
y_pred_test_proba = clf.predict_proba(df_final_test)[:,1]

f1_score_val_train = f1_score(y_train, y_pred_train)
roc_auc_val_train = roc_auc_score(y_train, y_pred_train)
f1_score_val_test = f1_score(y_test, y_pred_test)
roc_auc_val_test = roc_auc_score(y_test, y_pred_test)

global_result = global_result.append({'Model': "XGBoost",
                      'Hyperparameters': "Default Parameters",
                      'Train-F1-Score': '{0:.4}'.format(f1_score_val_tr
ain),
                      'Test-F1-Score': '{0:.4}'.format(f1_score_val_tes
t),
                      'Train-AUC': '{0:.4}'.format(roc_auc_val_train),
                      'Test-AUC': '{0:.4}'.format(roc_auc_val_test)
```

```
                      }, ignore_index=True)

print("TRAIN Scores","*"*20)
print("\tF1-Score: ", f1_score_val_train)
print("\tROC-AUC : ", roc_auc_val_train)

print("TEST Scores","*"*20)
print("\tF1-Score: ", f1_score_val_test)
print("\tROC-AUC : ", roc_auc_val_test)

print("==================================== ROC CURVE =================
===================================")
# calculate roc curve
fpr_train, tpr_train, thresholds = roc_curve(y_train, y_pred_train_prob
a)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(fpr_train, tpr_train, marker='.', label='TRAIN-AUC')

fpr_test, tpr_test, thresholds = roc_curve(y_test, y_pred_test_proba)
plt.plot(fpr_test, tpr_test, marker='.', label='TEST-AUC')
# show the plot
plt.legend()
plt.show()
print("====================================TRAIN CONFUSION MATRIX======
===================================")
plot_confusion_matrix(y_train, y_pred_train)
print("====================================TEST CONFUSION MATRIX=======
===================================")
plot_confusion_matrix(y_test, y_pred_test)
```

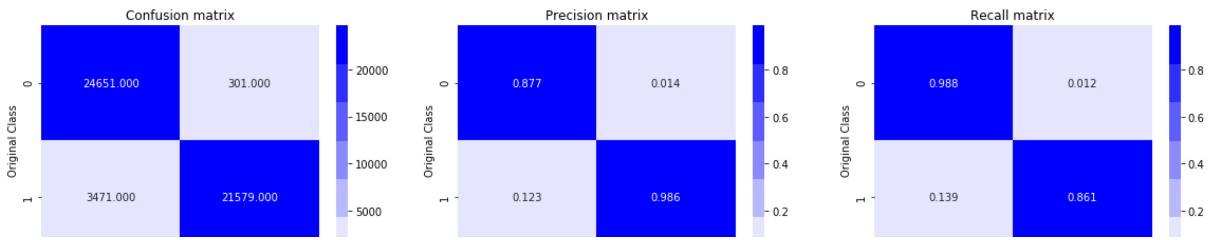
```
TRAIN Scores ********************
        F1-Score:  0.9746518162155606
        ROC-AUC :  0.9749473543192375
TEST Scores ********************
        F1-Score:  0.9196249733645856
        ROC-AUC :  0.9246869822394327
==================================== ROC CURVE ========================
============================
```



====================================TRAIN CONFUSION MATRIX=============
============================



====================================TEST CONFUSION MATRIX==============
============================



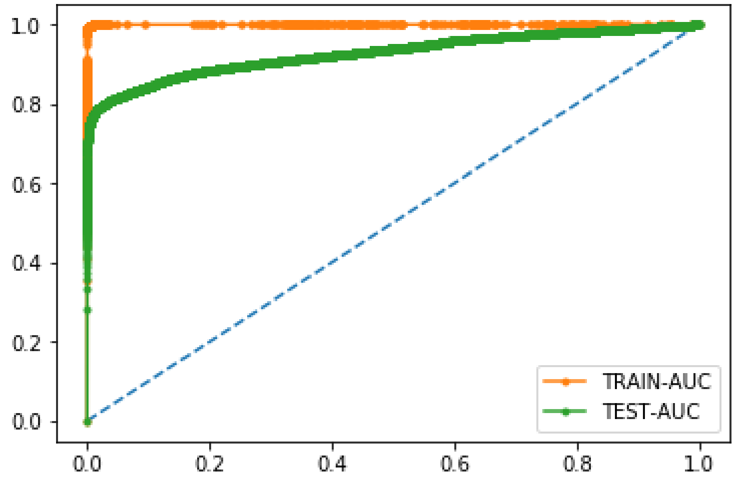
XGBoost Classifier with RandomSearchCV

In [12]:
```
parameters = {
                'n_estimators': [10, 50, 100, 125, 150, 175, 300],
                'learning_rate': [0.1, 0.001, 0.5, 1, 0.005],
                'max_depth': [2, 3, 4, 5, 7, 9],
                'gamma': [0, 0.01, 0.005, 0.5, 0.001],
                'min_child_weight':range(1,6,2)
            }
clf = XGBClassifier(n_jobs=-1, random_state=42, oob_score=True)
r_clf = RandomizedSearchCV(clf, param_distributions=parameters, scoring
='f1', cv=5, random_state=42, n_jobs=-1, return_train_score=True)
r_clf.fit(df_final_train, y_train)

y_pred_train = r_clf.predict(df_final_train)
y_pred_test = r_clf.predict(df_final_test)

y_pred_train_proba = r_clf.predict_proba(df_final_train)[:,1]
y_pred_test_proba = r_clf.predict_proba(df_final_test)[:,1]

f1_score_val_train = f1_score(y_train, y_pred_train)
roc_auc_val_train = roc_auc_score(y_train, y_pred_train)
f1_score_val_test = f1_score(y_test, y_pred_test)
roc_auc_val_test = roc_auc_score(y_test, y_pred_test)

global_result = global_result.append({'Model': "XGBoost",
                      'Hyperparameters': r_clf.best_params_,
                      'Train-F1-Score': '{0:.4}'.format(f1_score_val_tr
ain),
```

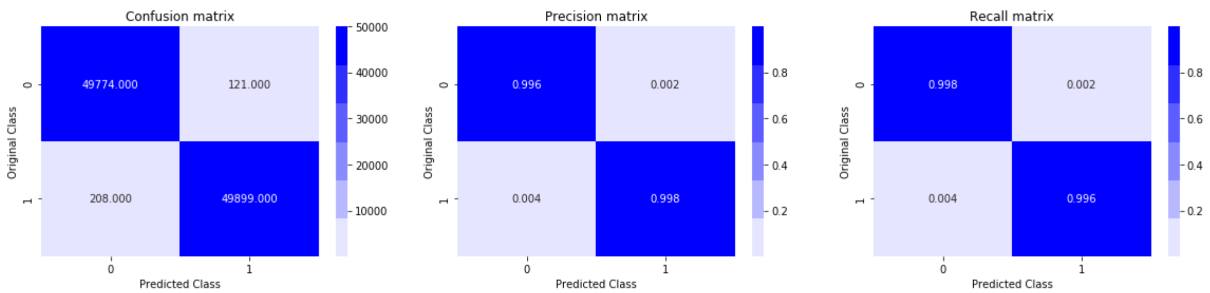
```
                        'Test-F1-Score': '{0:.4}'.format(f1_score_val_tes
t),
                        'Train-AUC': '{0:.4}'.format(roc_auc_val_train),
                        'Test-AUC': '{0:.4}'.format(roc_auc_val_test)
                       }, ignore_index=True)

print("*"*100)
print("Best Parameters: ", r_clf.best_params_)
print("*"*100)

print("TRAIN Scores","*"*20)
print("\tF1-Score: ", f1_score_val_train)
print("\tROC-AUC : ", roc_auc_val_train)

print("TEST Scores","*"*20)
print("\tF1-Score: ", f1_score_val_test)
print("\tROC-AUC : ", roc_auc_val_test)

print("==================================== ROC CURVE =================
===================================")
# calculate roc curve
fpr_train, tpr_train, thresholds = roc_curve(y_train, y_pred_train_prob
a)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(fpr_train, tpr_train, marker='.', label='TRAIN-AUC')

fpr_test, tpr_test, thresholds = roc_curve(y_test, y_pred_test_proba)
plt.plot(fpr_test, tpr_test, marker='.', label='TEST-AUC')
# show the plot
plt.legend()
plt.show()
print("====================================TRAIN CONFUSION MATRIX======
===================================")
plot_confusion_matrix(y_train, y_pred_train)
print("====================================TEST CONFUSION MATRIX=======
===================================")
plot_confusion_matrix(y_test, y_pred_test)
```

```
**********************************************************************
*****************************
```

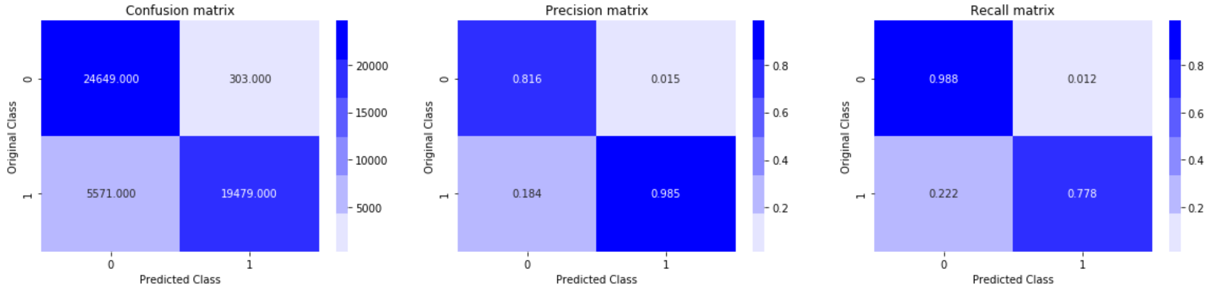
```
Best Parameters:  {'n_estimators': 300, 'learning_rate': 0.5, 'min_chil
d_weight': 1, 'gamma': 0.5, 'max_depth': 3}
***********************************************************************

*****************************
TRAIN Scores ********************
        F1-Score:  0.9967141730002895
        ROC-AUC :  0.9967118953474439
TEST Scores ********************
        F1-Score:  0.8689775160599571
        ROC-AUC :  0.8827307376270224
==================================== ROC CURVE =======================
============================
```




```
====================================TRAIN CONFUSION MATRIX============
============================
```



```
====================================TEST CONFUSION MATRIX==============
============================
```



Conclusion

In [13]:

```
global_result
```

Out[13]:

	Model	Hyperparameters	Train-F1-Score	Test-F1-Score	Train-AUC	Test-AUC
0	RandomForest	Default Parameters	0.9963	0.8881	0.9963	0.8973
1	RandomForest	{'n_estimators': 200, 'min_samples_split': 9, 'min_samples_leaf': 1}	0.9877	0.9277	0.9878	0.9315
2	XGBoost	Default Parameters	0.9747	0.9196	0.9749	0.9247
3	XGBoost	{'n_estimators': 300, 'learning_rate': 0.5, 'min_child_weight': 1, 'gamma': 0.5, 'max_depth': 3}	0.9967	0.869	0.9967	0.8827

Summary

Given a directed social graph, we had to predict missing links to recommend users.

The Dataset contains 2 columns - Source Node and Destination Node.

The graph based dataset was converted to supervised classification learning problem. I generated training samples of good and bad links from given directed graph and for each link got some features like no of followers, has he followed back, page rank, katz score, adar index, some svd fetures of adj matrix, some weight features, preferential attachment, svd dot etc. and trained ml model based on these features to predict link.

I deployed two Models - **RandomForest and XGBoost with hyperparameter tuning using RandomizedSearchCV**. Each model's performance was assessed using **F1-score and AUC score**.

XGBoost with extreme hypertuning was overfitting to a little extent but RandomForest with hyperparameter tuning was performing the best with test f1 score of 0.92 and auc score of 0.93.