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

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

import warnings
import networkx as nx
import pdb
import pickle
```

In [2]:
```python
from google.colab import drive

# This will prompt for authorization.
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [3]:
```python
!ls
```

```
drive  sample_data
```

Reading data

```
In [4]: if os.path.isfile('drive/My Drive/train_pos_after_eda.csv'):
            train_graph=nx.read_edgelist('drive/My Drive/train_pos_after_eda.csv',delimiter=',',create_using=nx.DiGraph(),noc
            print(nx.info(train_graph))
        else:
            print("please run the FB_EDA.ipynb or download the files from drive")
```

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

2. Similarity measures

2.1 Jaccard Distance:

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

$$j = \frac{|X \cap Y|}{|X \cup Y|} \tag{1}$$

```
In [5]: #for followees
        def jaccard_for_followees(a,b):
            try:
                if len(set(train_graph.successors(a))) == 0  | len(set(train_graph.successors(b))) == 0:
                    return 0
                sim = (len(set(train_graph.successors(a)).intersection(set(train_graph.successors(b)))))/\
                                            (len(set(train_graph.successors(a)).union(set(train_graph.successors(b)))))
            except:
                return 0
            return sim
```

```
In [6]: #one test case
        print(jaccard_for_followees(273084,1505602))
```

```
0.0
```

In [7]:
```
#node 1635354 not in graph
print(jaccard_for_followees(273084,1505602))
```

0.0

In [8]:
```
#for followers
def jaccard_for_followers(a,b):
    try:
        if len(set(train_graph.predecessors(a))) == 0  | len(set(g.predecessors(b))) == 0:
            return 0
        sim = (len(set(train_graph.predecessors(a)).intersection(set(train_graph.predecessors(b)))))/\
                                 (len(set(train_graph.predecessors(a)).union(set(train_graph.predecessors(b)))))
        return sim
    except:
        return 0
```

In [9]:
```
print(jaccard_for_followers(273084,470294))
```

0

In [10]:
```
#node 1635354 not in graph
print(jaccard_for_followees(669354,1635354))
```

0

2.2 Cosine distance

$$CosineDistance = \frac{|X \cap Y|}{|X| \cdot |Y|} \tag{2}$$

In [11]:
```
#for followees
def cosine_for_followees(a,b):
    try:
        if len(set(train_graph.successors(a))) == 0  | len(set(train_graph.successors(b))) == 0:
            return 0
        sim = (len(set(train_graph.successors(a)).intersection(set(train_graph.successors(b)))))/\
                                    (math.sqrt(len(set(train_graph.successors(a)))*len((set(train_graph.successors(b)
        return sim
    except:
        return 0
```

In [12]:
```
print(cosine_for_followees(273084,1505602))
```

```
0.0
```

In [13]:
```
print(cosine_for_followees(273084,1635354))
```

```
0
```

In [14]:
```
def cosine_for_followers(a,b):
    try:

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

In [15]:
```
print(cosine_for_followers(2,470294))
```

```
0.02886751345948129
```

In [16]:
```
print(cosine_for_followers(669354,1635354))
```

```
0
```

3. Ranking Measures

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

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

Mathematical PageRanks for a simple network, expressed as percentages. (Google uses a logarithmic scale.) Page C has a higher PageRank than Page E, even though there are fewer links to C; the one link to C comes from an important page and hence is of high value. If web surfers who start on a random page have an 85% likelihood of choosing a random link from the page they are currently visiting, and a 15% likelihood of jumping to a page chosen at random from the entire web, they will reach Page E 8.1% of the time. **(The 15% likelihood of jumping to an**

arbitrary page corresponds to a damping factor of 85%.) Without damping, all web surfers would eventually end up on Pages A, B, or C, and all other pages would have PageRank zero. In the presence of damping, Page A effectively links to all pages in the web, even though it has no outgoing links of its own.

3.1 Page Ranking

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

In [17]:
```
if not os.path.isfile('drive/My Drive/page_rank.p'):
    pr = nx.pagerank(train_graph, alpha=0.85)
    pickle.dump(pr,open('drive/My Drive/page_rank.p','wb'))
else:
    pr = pickle.load(open('drive/My Drive/page_rank.p','rb'))
```

In [18]:
```
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 [19]:
```
#for imputing to nodes which are not there in Train data
mean_pr = float(sum(pr.values())) / len(pr)
print(mean_pr)
```

```
5.615699699389075e-07
```

4. Other Graph Features

4.1 Shortest path:

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

In [20]:
```
#if has direct edge then deleting that edge and calculating shortest path
def compute_shortest_path_length(a,b):
    p=-1
```

```
    try:
        if train_graph.has_edge(a,b):
            train_graph.remove_edge(a,b)
            p= nx.shortest_path_length(train_graph,source=a,target=b)
            train_graph.add_edge(a,b)
        else:
            p= nx.shortest_path_length(train_graph,source=a,target=b)
        return p
    except:
        return -1
```

In [21]:
```
#testing
compute_shortest_path_length(77697, 826021)
```

Out[21]: 10

In [22]:
```
#testing
compute_shortest_path_length(669354,1635354)
```

Out[22]: -1

4.2 Checking for same community

In [23]:
```
#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 [24]:
```
belongs_to_same_wcc(861, 1659750)
```

Out[24]: 0

In [25]:
```
belongs_to_same_wcc(669354,1635354)
```

Out[25]: 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 [26]:
```
#adar index
def calc_adar_in(a,b):
    sum=0
    try:
        n=list(set(train_graph.successors(a)).intersection(set(train_graph.successors(b))))
        if len(n)!=0:
            for i in n:
                sum=sum+(1/np.log10(len(list(train_graph.predecessors(i)))))
```

```
                return sum
            else:
                return 0
    except:
        return 0
```

In [27]:
```
calc_adar_in(1,189226)
```

Out[27]: 0

In [28]:
```
calc_adar_in(669354,1635354)
```

Out[28]: 0

4.4 Is persion was following back:

In [29]:
```
def follows_back(a,b):
    if train_graph.has_edge(b,a):
        return 1
    else:
        return 0
```

In [30]:
```
follows_back(1,189226)
```

Out[30]: 1

In [31]:
```
follows_back(669354,1635354)
```

Out[31]: 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 [32]:
```
if not os.path.isfile('katz.p'):
    katz = nx.katz.katz_centrality(train_graph,alpha=0.005,beta=1)
    pickle.dump(katz,open('katz.p','wb'))
else:
    katz = pickle.load(open('katz.p','rb'))
```

In [33]:
```
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 [34]:
```
mean_katz = float(sum(katz.values())) / len(katz)
print(mean_katz)
```

```
0.0007483800935562018
```

4.6 Hits Score

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

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

In [35]:
```
if not os.path.isfile('drive/My Drive/hits.p'):
    hits = nx.hits(train_graph, max_iter=100, tol=1e-08, nstart=None, normalized=True)
    pickle.dump(hits,open('drive/My Drive/hits.p','wb'))
else:
    hits = pickle.load(open('drive/My Drive/hits.p','rb'))
```

In [36]:
```
print('min',hits[0][min(hits[0], key=hits[0].get)])
print('max',hits[0][max(hits[0], key=hits[0].get)])
print('mean',float(sum(hits[0].values())) / len(hits[0]))
```

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

5. Featurization

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

In [37]:
```
import random
if os.path.isfile('drive/My Drive/train_after_eda.csv'):
    filename = "drive/My Drive/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

else:
  print("verify")
```

In [38]:
```
if os.path.isfile('drive/My Drive/train_after_eda.csv'):
```

```
    filename = "drive/My Drive/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
else:
  print("verify")
```

In [39]:

```
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 [40]:

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

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

Out[40]:

	source_node	destination_node	indicator_link
0	273084	1505602	1
1	99661	62265	1

In [41]:

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

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

Out[41]:

	source_node	destination_node	indicator_link
0	848424	784690	1

	source_node	destination_node	indicator_link
1	593083	1131838	1

5.2 Adding a set of features

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

1. jaccard_followers
2. jaccard_followees
3. cosine_followers
4. cosine_followees
5. num_followers_s
6. num_followees_s
7. num_followers_d
8. num_followees_d
9. inter_followers
10. inter_followees

In [42]:
```
if not os.path.isfile('My Drive/storage_sample_stage1.h5'):
    #mapping jaccrd followers to train and test data
    df_final_train['jaccard_followers'] = df_final_train.apply(lambda row:
                                        jaccard_for_followers(row['source_node'],row['destination_node']),axis=1)
    df_final_test['jaccard_followers'] = df_final_test.apply(lambda row:
                                        jaccard_for_followers(row['source_node'],row['destination_node']),axis=1)

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


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

```
    df_final_test['cosine_followers'] = df_final_test.apply(lambda row:
                                        cosine_for_followers(row['source_node'],row['destination_node']),axis=1)

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

In [43]:

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

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

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

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

In [44]:
```
import h5py
from pandas import HDFStore
```

In [45]:
```
if not os.path.isfile('My Drive/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('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('My Drive/storage_sample_stage1.h5', 'train_df',mode='r')
    df_final_test = read_hdf('My Drive/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('My Drive/storage_sample_stage2.h5'):
    #mapping adar index on train
    df_final_train['adar_index'] = df_final_train.apply(lambda row: calc_adar_in(row['source_node'],row['destination_
    #mapping adar index on test
    df_final_test['adar_index'] = df_final_test.apply(lambda row: calc_adar_in(row['source_node'],row['destination_no
```

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

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

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

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

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

    hdf = HDFStore('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('My Drive/storage_sample_stage2.h5', 'train_df',mode='r')
    df_final_test = read_hdf('My Drive/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

$$W = \frac{1}{\sqrt{1 + |X|}} \tag{3}$$

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

In [47]:
```
#weight for source and destination of each link
from tqdm import tqdm
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 [00:20<00:00, 88723.77it/s]
```

In [48]:

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

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


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

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

else:
  print("verify")
```

In [49]:

```
if not os.path.isfile('My Drive/fea_sample/storage_sample_stage3.h5'):

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

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

    #Katz centrality score for source and destination in Train and test
```

```
    #if anything not there in train graph then adding mean katz score
    df_final_train['katz_s'] = df_final_train.source_node.apply(lambda x: katz.get(x,mean_katz))
    df_final_train['katz_d'] = df_final_train.destination_node.apply(lambda x: katz.get(x,mean_katz))

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

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

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

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

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

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

5.5 Adding new set of features

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

1. SVD features for both source and destination

In [50]:
```
def svd(x, S):
    try:
        z = sadj_dict[x]
        return S[z]
    except:
        return [0,0,0,0,0,0]
```

In [51]:
```
#for svd features to get feature vector creating a dict node val and inedx in svd vector
sadj_col = sorted(train_graph.nodes())
sadj_dict = { val:idx for idx,val in enumerate(sadj_col)}
```

In [52]:
```
Adj = nx.adjacency_matrix(train_graph,nodelist=sorted(train_graph.nodes())).asfptype()
```

In [53]:
```
import numpy as np
from scipy.sparse.linalg import svds
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 [54]:
```
if not os.path.isfile('My Drive/storage_sample_stage4.h5'):
    #====================================================================================================

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

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

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

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

```
    #===================================================================================================

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

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

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

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

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

    hdf = HDFStore('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()
```

Assignments:

1. Add another feature called Preferential Attachment with followers and followees data of vertex. you can check about Preferential Attachment in below link http://be.amazd.com/link-prediction/
2. Add feature called svd_dot. you can calculate svd_dot as Dot product between sourse node svd and destination node svd features. you can read about this in below pdf https://storage.googleapis.com/kaggle-forum-message-attachments/2594/supervised_link_prediction.pdf
3. Tune hyperparameters for XG boost with all these features and check the error metric.

Preferential attachment

In [55]:
```
def Preferential_Attachment_follower(a, b):
    try:
        if len(set(train_graph.successors(a))) == 0  | len(set(train_graph.successors(b))) == 0:
            return 0
        pref_score = len(set(train_graph.successors(a)))*len(set(train_graph.successors(b)))
```

```
        return pref_score
    except:
        return 0
```

In [56]:
```
def Preferential_Attachment_followee(a, b):
    try:
        if len(set(train_graph.predecessors(a))) == 0  | len(set(train_graph.predecessors(b))) == 0:
            return 0
        pref_score_pre = len(set(train_graph.predecessors(a)))*len(set(train_graph.predecessors(b)))
        return pref_score_pre
    except:
        return 0
```

In [57]:
```
if not os.path.isfile('My Drive/storage_sample_stage3.h5'):
    #mapping Preferential_Attachment_follower
    df_final_train['Preferential_Attachment_follower'] = df_final_train.apply(lambda row: Preferential_Attachment_fol
    df_final_test['Preferential_Attachment_follower'] = df_final_train.apply(lambda row: Preferential_Attachment_foll

    #mapping Preferential_Attachment_followee
    df_final_train['Preferential_Attachment_followee'] = df_final_test.apply(lambda row: Preferential_Attachment_foll
    df_final_test['Preferential_Attachment_followee'] = df_final_test.apply(lambda row: Preferential_Attachment_follo
```

In [58]:
```
df_final_train.head(1)
```

Out[58]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	num_followers_s	num_fol
0	273084	1505602	1	0	0.0	0.0	0.0	11	

Add feature called svd_dot Dot product between source node and destination node

In [59]:
```
svd_dot = []

lst_1 = ['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_v_s_1', 'svd_v_s_2', 'svd_
lst_2 = ['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_d_1', 'svd_v_d_2', 'svd_


for x in range(df_final_train.shape[0]):
  src = []
```

```
    dest = []

    for y in lst_1:
      src.append(df_final_train[y][x])

    for z in lst_2:
      dest.append(df_final_train[z][x])

    svd_dot.append(np.dot(src,dest))
df_final_train['svd_dot_prod'] = svd_dot
```

In [60]:
```
svd_dot = []

for x in range(df_final_test.shape[0]):
    src = []
    dest = []

    for y in lst_1:
      src.append(df_final_test[y][x])

    for z in lst_2:
      dest.append(df_final_test[z][x])

    svd_dot.append(np.dot(src,dest))

df_final_test['svd_dot_prod'] = svd_dot
```

In [61]:
```
df_final_test.head(1)
```

Out[61]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	num_followers_s	num_fol
0	848424	784690	1	0	0.0	0.029161	0.0	6	

In [62]:
```
df_final_test.columns
```

Out[62]:
```
Index(['source_node', 'destination_node', 'indicator_link',
       'jaccard_followers', 'jaccard_followees', 'cosine_followers',
       'cosine_followees', 'num_followers_s', 'num_followers_d',
       'num_followees_s', 'num_followees_d', 'inter_followers',
```

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

Training Model

In [63]:

```
y_train = df_final_train['indicator_link']
y_test = df_final_test['indicator_link']
```

In [64]:

```
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 [65]:

```
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier as XGBC
model = XGBC(random_state=40, learning_rate=0.001, reg_lambda=0.85, reg_alpha=0.9)

parameters = {'n_estimators': [100,150,200], 'max_depth':[9,10,11]}

classifier =  GridSearchCV(model, parameters, cv=3, scoring='f1', return_train_score=True, n_jobs=-1)
classifier.fit(df_final_train, y_train)

ans = pd.DataFrame.from_dict(classifier.cv_results_)
ans.head(2)
```

Out[65]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_n_estimators	params	split0_test_score	split1_test_
0	49.087363	17.526458	0.126537	0.010939	9	100	{'max_depth': 9, 'n_estimators': 100}	0.996764	0.!

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_n_estimators	params	split0_test_score	split1_test
1	77.674475	28.652536	0.147727	0.022101	9	150	{'max_depth': 9, 'n_estimators': 150}	0.996764	0.9

In [66]:
```
#dir(ans)
```

In [67]:
```
best_para = classifier.best_params_
print(best_para)
print(classifier.best_estimator_)
```

```
{'max_depth': 10, 'n_estimators': 200}
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.001, max_delta_step=0, max_depth=10,
              min_child_weight=1, missing=None, n_estimators=200, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=40,
              reg_alpha=0.9, reg_lambda=0.85, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

In [68]:
```
xgbc_clf = XGBC(max_depth = best_para['max_depth'], random_state=42, n_estimators = best_para['n_estimators'], learn
xgbc_clf.fit(df_final_train, y_train)
y_train_pred = xgbc_clf.predict(df_final_train)
y_test_pred = xgbc_clf.predict(df_final_test)
```

In [69]:
```
from sklearn.metrics import f1_score
print('f1 score train',f1_score(y_train,y_train_pred))
print('f1 score test',f1_score(y_test,y_test_pred))
```

```
f1 score train 0.9994191054223505
f1 score test 0.71506663804066
```

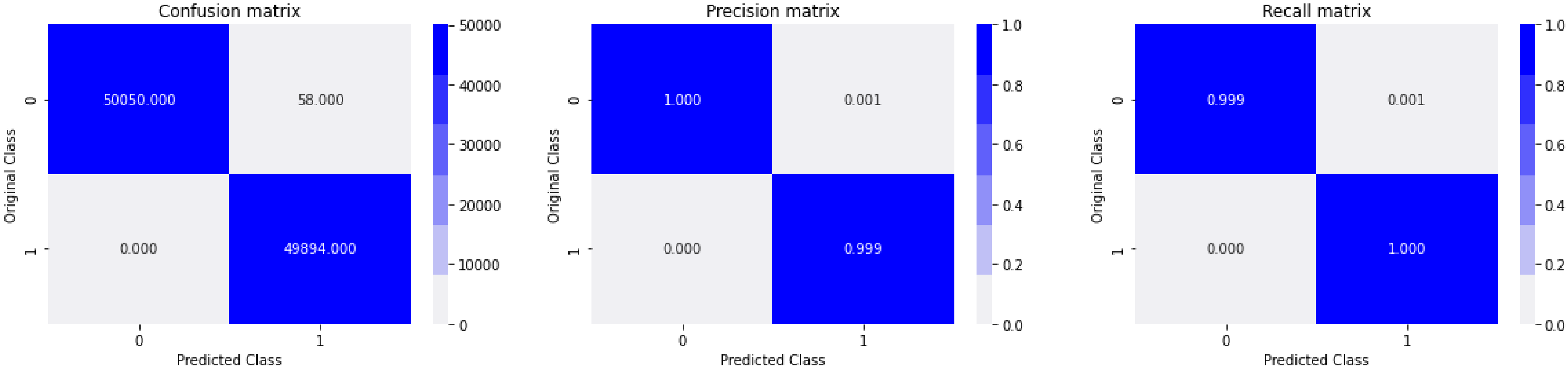
In [70]:
```
from sklearn.metrics import confusion_matrix
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)

    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()
```

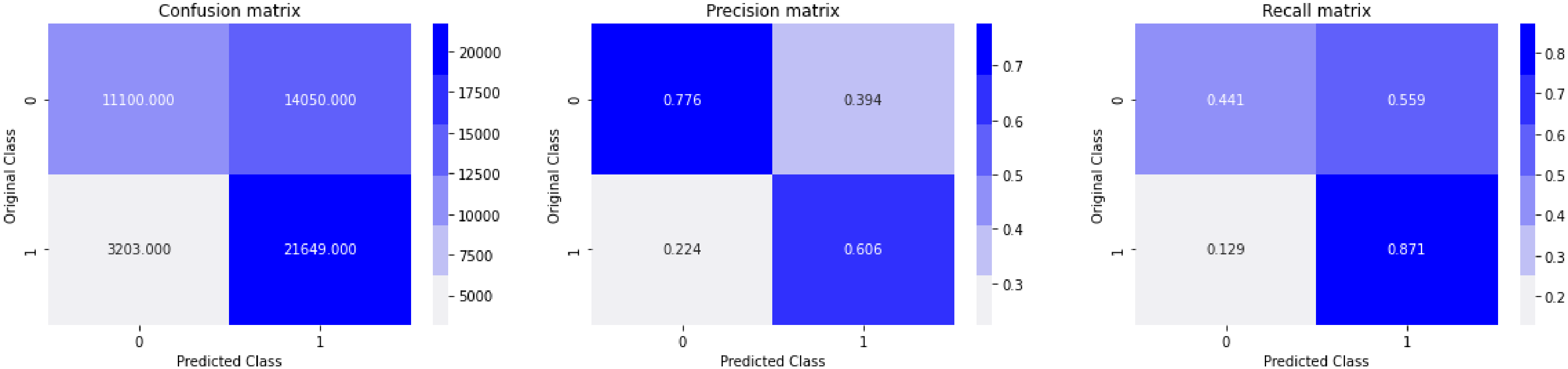
In [71]:

```
print('Train confusion_matrix')
plot_confusion_matrix(y_train,y_train_pred)
print('Test confusion_matrix')
plot_confusion_matrix(y_test,y_test_pred)
```

```
Train confusion_matrix
```

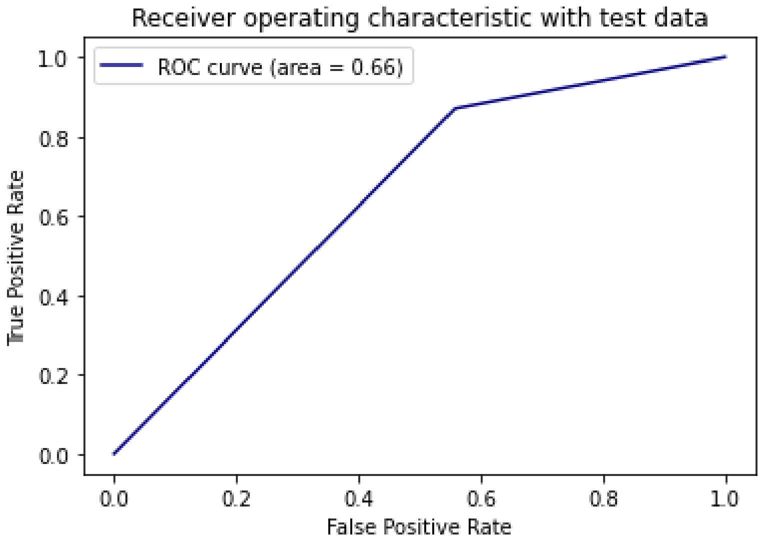


Test confusion_matrix



In [72]:

```
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()
```



OBSERVATION:

1. Firstly we perform eda for better vistualization

2.Then we apply Jaccard Distance,page ranking,cosine distance for node node similarity

3.Then apply periferial attachment on both follower and followee

4.SVD dot product

5.Tranning a model

6.f1 score on train and test

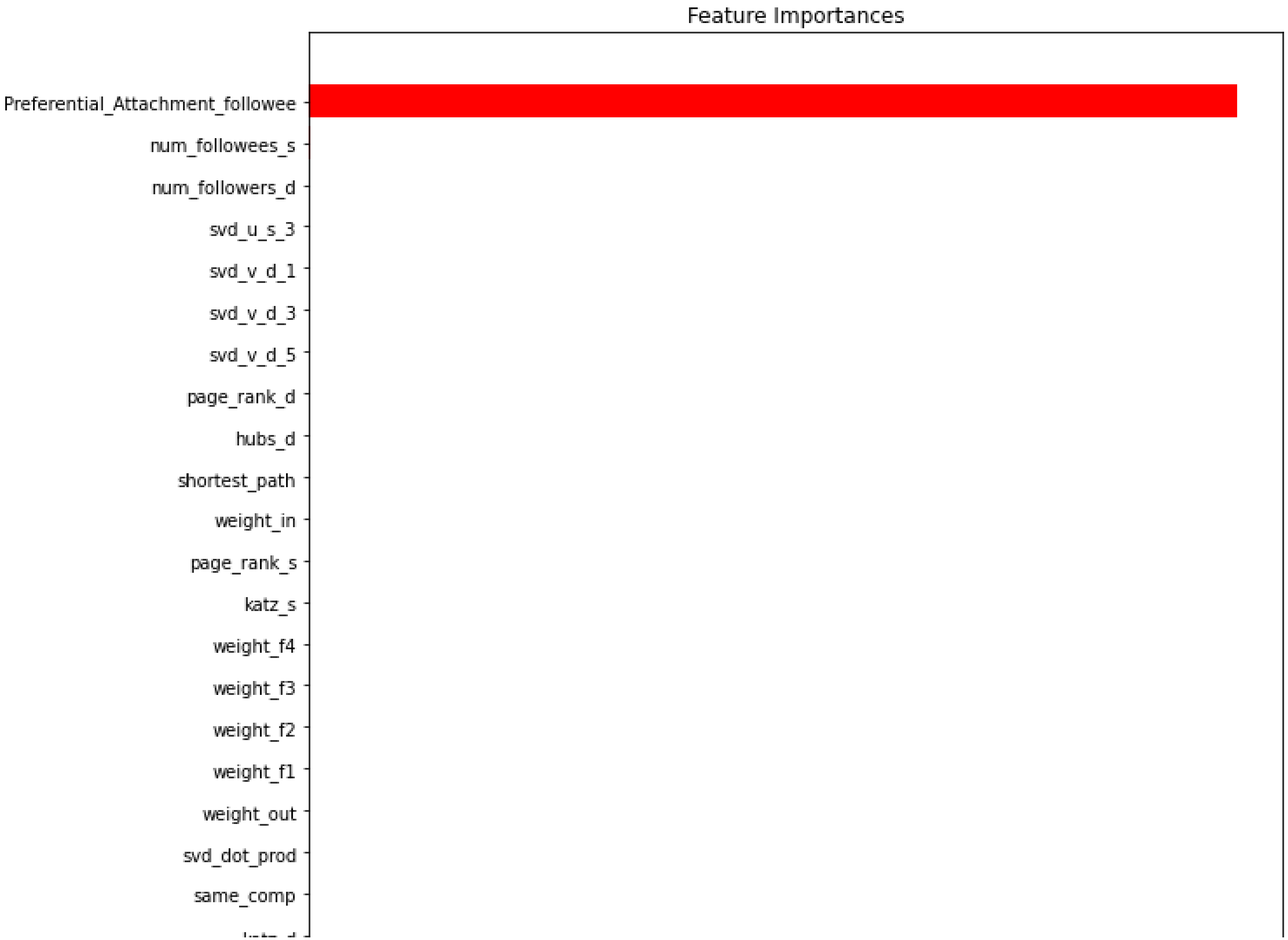
7.confusion matrix

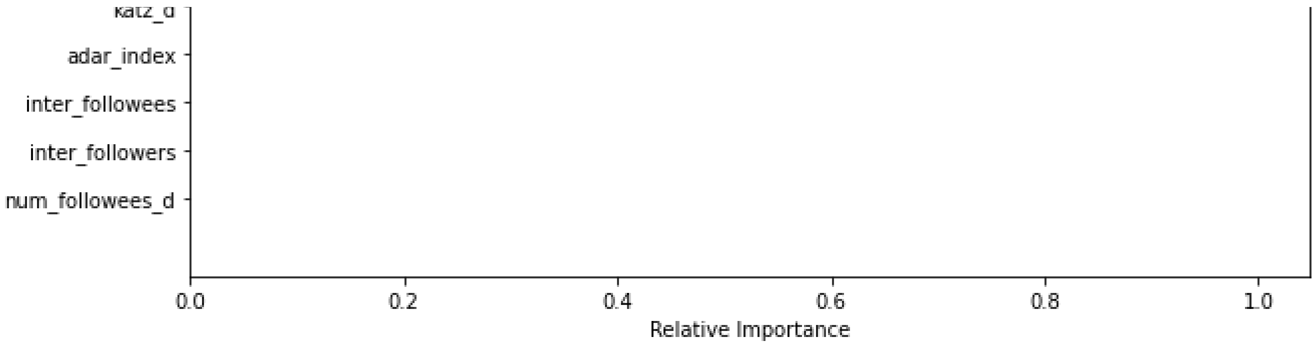
8.roc and auc curve

9.feature engineering

In [77]:
```
features = df_final_train.columns
```

```
importances = xgbc_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()
```




katz_d
adar_index
inter_followees
inter_followers
num_followees_d
0.0
0.2
0.4
0.6
0.8
1.0
Relative Importance

In [ ]: