Social network Graph Link Prediction - Facebook Challenge

In [1]:

```
from google.colab import drive
drive.mount('/content/drive')
```

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

In [2]:

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

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

import warnings
import networkx as nx

import pdb
import pickle
from pandas import HDFStore,DataFrame
from pandas import read_hdf
from scipy.sparse.linalg import svds, eigs
import gc
from tqdm import tqdm
```

1. Reading Data

In [3]:

```
if os.path.isfile('/content/drive/MyDrive/Facebook/data/after_eda/train_pos_after_eda.csv'):
    train_graph=nx.read_edgelist('/content/drive/MyDrive/Facebook/data/after_eda/train_pos_after_eda.csv',delimiter=',',create_using=nx.DiGraph(),nodetype=int)
    print(nx.info(train_graph))
else:
    print("please run the FB_EDA.ipynb or download the files from drive")
```

```
DiGraph with 1780722 nodes and 7550015 edges
```

2. Similarity measures

2.1 Jaccard Distance:

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

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

In [4]:

```
#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 [5]:

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

```
0.0
```

In [6]:

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

```
0.0
```

In [7]:

```
#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 [8]:

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

0

In [9]:

```
#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 [10]:

```
#for followees
def cosine_for_followees(a,b):
    try:
        if len(set(train_graph.successors(a))) == 0  | len(set(train_graph.succe
ssors(b))) == 0:
            return 0
        sim = (len(set(train_graph.successors(a)).intersection(set(train_graph.s
uccessors(b)))))/\
                                    (math.sqrt(len(set(train_graph.successors(a
)))*len((set(train_graph.successors(b))))))
        return sim
    except:
        return 0
```

In [11]:

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

0.0

In [12]:

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

```
0
```

In [13]:

```
def cosine_for_followers(a,b):
    try:

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

In [14]:

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

```
0.02886751345948129
```

In [15]:

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

```
0
```

3. Ranking Measures

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

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

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

3.1 Page Ranking

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

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 [14]:

```
#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 [19]:

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

Out[19]:

```
10
```

In [18]:

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

Out[18]:

```
-1
```

4.2 Checking for same community

In [15]:

```
#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 [21]:

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

Out[21]:

0

In [22]:

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

Out[22]:

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 [16]:

```
#adar index
def calc_adar_in(a,b):
    sum=0
    try:
        n=list(set(train_graph.successors(a)).intersection(set(train_graph.succe
ssors(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 [24]:

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

Out[24]:

0

In [25]:

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

Out[25]:

0

4.4 Is persion was following back:

In [17]:

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

In [27]:

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

Out[27]:

1

In [28]:

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

Out[28]:

0

4.5 Katz Centrality:

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

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

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

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

$$\lambda$$

.

The parameter

$$\beta$$

controls the initial centrality and

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

In [ ]:

```
if not os.path.isfile('/content/drive/MyDrive/Facebook/data/fea_sample/katz.p'):
    katz = nx.katz.katz_centrality(train_graph,alpha=0.005,beta=1)
    pickle.dump(katz,open('/content/drive/MyDrive/Facebook/data/fea_sample/katz.p','wb'))
else:
    katz = pickle.load(open('/content/drive/MyDrive/Facebook/data/fea_sample/katz.p','rb'))
```

In [27]:

```
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 [28]:

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

```
0.0007483800935562018
```

4.6 Hits Score

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

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

In [24]:

```
if not os.path.isfile('/content/drive/MyDrive/Facebook/data/fea_sample/hits.p'):
    hits = nx.hits(train_graph, max_iter=100, tol=1e-08, nstart=None, normalized
=True)
    pickle.dump(hits,open('/content/drive/MyDrive/Facebook/data/fea_sample/hits.
p','wb'))
else:
    hits = pickle.load(open('/content/drive/MyDrive/Facebook/data/fea_sample/hit
s.p','rb'))
```

In [25]:

```
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 [29]:

```
! gdown --id 1lcxzVZ0-MkPmoH3lS35Q8rRfrecKSXb1
! gdown --id 1_KN7S8zfHdrkRjRYOEtBxBVq8JrGxPXD
```

```
Downloading...
From: https://drive.google.com/uc?id=1lcxzVZ0-MkPmoH3lS35Q8rRfrecKSX
b1
To: /content/train_after_eda.csv
239MB [00:01, 149MB/s]
Downloading...
From: https://drive.google.com/uc?id=1_KN7S8zfHdrkRjRYOEtBxBVq8JrGxP
XD
To: /content/test_after_eda.csv
59.7MB [00:00, 115MB/s]
```

In [30]:

```
import random
if os.path.isfile('train_after_eda.csv'):
    filename = "train_after_eda.csv"
    # you uncomment this line, if you dont know the lentgh of the file name
    # here we have hardcoded the number of lines as 15100030
    # n_train = sum(1 for line in open(filename)) #number of records in file (ex
cludes 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 [31]:

```
if os.path.isfile('train_after_eda.csv'):
    filename = "test_after_eda.csv"
    # you uncomment this line, if you dont know the lentgh of the file name
    # here we have hardcoded the number of lines as 3775008
    # n_test = sum(1 for line in open(filename)) #number of records in file (exc
ludes 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 [32]:

```
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 [33]:

```
#https://drive.google.com/file/d/19mviN_yeJIfakb4kU5NfKdQlOQtaQ-kH/view?usp=shar
ing
!gdown --id 19mviN_yeJIfakb4kU5NfKdQlOQtaQ-kH
```

```
Downloading...
From: https://drive.google.com/uc?id=19mviN_yeJIfakb4kU5NfKdQlOQtaQ-
kH
To: /content/train_y.csv
45.3MB [00:00, 124MB/s]
```

In [34]:

```
#https://drive.google.com/file/d/1H6qybuXr8i_USWu3k3ulXEOurc-SElUh/view?usp=shar
ing
!gdown --id 1H6qybuXr8i_USWu3k3ulXEOurc-SElUh
```

```
Downloading...
From: https://drive.google.com/uc?id=1H6qybuXr8i_USWu3k3ulXEOurc-SEl
Uh
To: /content/test_y.csv
11.3MB [00:00, 99.1MB/s]
```

In [35]:

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

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

Out[35]:

	source_node	destination_node	indicator_link
0	273084	1505602	1
1	1452769	154251	1

In [36]:

```
df_final_test = pd.read_csv('test_after_eda.csv', skiprows=skip_train, names=['s
ource_node', 'destination_node'])
df_final_test['indicator_link'] = pd.read_csv('test_y.csv', skiprows=skip_train,
names=['indicator_link'])
print("Our train matrix size ",df_final_test.shape)
df_final_test.head(2)
```

```
Our train matrix size  (25073, 3)
```

Out[36]:

	source_node	destination_node	indicator_link
0	848424	784690	1
1	1441135	977791	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 [37]:

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

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

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

    return  num_followers_s,num_followees_s,num_followers_d,num_followees_d,inte
r_followers,inter_followees
```

In [ ]:

```
if not os.path.isfile('/content/drive/MyDrive/Facebook/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('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('/content/drive/MyDrive/Facebook/data/fea_sample/storage_sample_stage1.h5', 'train_df',mode='r')
    df_final_test = read_hdf('/content/drive/MyDrive/Facebook/data/fea_sample/storage_sample_stage1.h5', 'test_df',mode='r')
```

In [41]:

```
df_final_train = read_hdf('/content/drive/MyDrive/Facebook/data/fea_sample/storage_sample_stage1.h5', 'train_df',mode='r')
df_final_test = read_hdf('/content/drive/MyDrive/Facebook/data/fea_sample/storage_sample_stage1.h5', 'test_df',mode='r')
df_final_train.head()
```

Out[41]:

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

In [ ]:

```
a=df_final_train['num_followers_s'].values
b=df_final_train['num_followers_d'].values
for x,y in (zip(a,b)):
  if x==0:
    if y!=0:
      print('i')
```

In [ ]:

```
np.count_nonzero(a)
```

Out[ ]:

0

In [ ]:

```
np.count_nonzero(b)
```

Out[ ]:

0

In [43]:

```
# ! gdown --id 1fDJptlCFEWNV5UNGPc4geTykgFI3PDCV
```

In [ ]:

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

In [ ]:

```
# df_final_train.tail()
```

Out[ ]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cc
99997	139353	893843	0	0	0.0	
99998	910842	704068	0	0	0.0	
99999	794228	1172755	0	0	0.0	
100000	949992	1854931	0	0	0.0	
100001	1642037	1090977	0	0	0.0	

In [ ]:

```
# df_final_train_new=df_final_train.drop(['num_followers_s',    'num_followees_
s',     'num_followees_d'       ,'inter_followers',     'inter_followees'],axis=
1)
```

In [ ]:

```
# df_final_train['num_followers_d']= compute_features_stage1(df_final_train)
```

In [ ]:

```
# df_final_train.tail()
```

Out[ ]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cc
99997	139353	893843	0	0	0.0	
99998	910842	704068	0	0	0.0	
99999	794228	1172755	0	0	0.0	
100000	949992	1854931	0	0	0.0	
100001	1642037	1090977	0	0	0.0	

In [ ]:

```
# for val in df_final_train_new['num_followers_s'].values:
#   if(val>0):
#     print(val)
```

In [44]:

```
# https://drive.google.com/file/d/10qJ04GRcaDxc16gmJXb8rpGPmlyys7E2/view?usp=sharing
! gdown --id 10qJ04GRcaDxc16gmJXb8rpGPmlyys7E2
```

```
Downloading...
From: https://drive.google.com/uc?id=10qJ04GRcaDxc16gmJXb8rpGPmlyys7E2
To: /content/storage_sample_stage2.h5
22.9MB [00:00, 105MB/s]
```

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 [ ]:

```
if not os.path.isfile('storage_sample_stage2.h5'):
    #mapping adar index on train
    df_final_train['adar_index'] = df_final_train.apply(lambda row: calc_adar_in
(row['source_node'],row['destination_node']),axis=1)
    #mapping adar index on test
    df_final_test['adar_index'] = df_final_test.apply(lambda row: calc_adar_in(r
ow['source_node'],row['destination_node']),axis=1)

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

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

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

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

    #------------------------------------------------------------------------
----------------------------
    #mapping shortest path on train
    df_final_train['shortest_path'] = df_final_train.apply(lambda row: compute_s
hortest_path_length(row['source_node'],row['destination_node']),axis=1)
    #mapping shortest path on test
    df_final_test['shortest_path'] = df_final_test.apply(lambda row: compute_sho
rtest_path_length(row['source_node'],row['destination_node']),axis=1)

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

In [45]:

```
df_final_train = read_hdf('/content/drive/MyDrive/Facebook/data/fea_sample/storage_sample_stage2.h5', 'train_df',mode='r')
df_final_test = read_hdf('/content/drive/MyDrive/Facebook/data/fea_sample/storage_sample_stage2.h5', 'test_df',mode='r')
df_final_train.head()
```

Out[45]:

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

5.4 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 [46]:

```
#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 [00:21<00:00, 84207.15it/s]
```

In [ ]:

```
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(lambda x: We
ight_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: Weig
ht_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.weig
ht_out
    df_final_train['weight_f2'] = df_final_train.weight_in * df_final_train.weig
ht_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.we
ight_out)
    df_final_test['weight_f4'] = (1*df_final_test.weight_in + 2*df_final_test.we
ight_out)
```

In [ ]:

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

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

    df_final_test['page_rank_s'] = df_final_test.source_node.apply(lambda x:pr.g
et(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.g
et(x,mean_katz))
    df_final_train['katz_d'] = df_final_train.destination_node.apply(lambda x: k
atz.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: kat
z.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: h
its[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: hit
s[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(lamb
da x: hits[1].get(x,0))

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

    hdf = HDFStore('data/fea_sample/storage_sample_stage3.h5')
```

```
    hdf.put('train_df',df_final_train, format='table', data_columns=True)
    hdf.put('test_df',df_final_test, format='table', data_columns=True)
    hdf.close()
else:
    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_d
f',mode='r')
```

In [48]:

```
df_final_train = read_hdf('/content/drive/MyDrive/Facebook/data/fea_sample/stora
ge_sample_stage3.h5', 'train_df',mode='r')
df_final_test = read_hdf('/content/drive/MyDrive/Facebook/data/fea_sample/storag
e_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 [49]:

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

In [50]:

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

In [51]:

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

In [52]:

```
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 [ ]:

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

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

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

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

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

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

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

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

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

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

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

In [ ]:

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

In [59]:

```
df_final_train.shape
```

Out[59]:

```
(100002, 54)
```

In [58]:

```
df_final_train = read_hdf('/content/drive/MyDrive/Facebook/data/fea_sample/storage_sample_stage4.h5', 'train_df',mode='r')
df_final_test = read_hdf('/content/drive/MyDrive/Facebook/data/fea_sample/storage_sample_stage4.h5', 'test_df',mode='r')
```

Preferential Attachment

In [60]:

```
def followee_preferential_attachment(user1,user2):
    try:
        user_1 = len(set(train_graph.successors(user1)))
        user_2 = len(set(train_graph.successors(user2)))
        return(user_1*user_2)
    except:
        return(0)

def follower_preferential_attachment(user1,user2):
    try:
        user_1 = len(set(train_graph.predecessors(user1)))
        user_2 = len(set(train_graph.predecessors(user2)))
        return(user_1*user_2)
    except:
        return(0)
```

In [61]:

```
startTime = datetime.datetime.now()
print("Current Time = ",startTime)

if not os.path.isfile('/content/drive/MyDrive/Facebook/data/fea_sample/storage_sample_stage5.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)
    #================================================================================

    df_final_train['followee_preferential_attachment'] = df_final_train.apply(lambda row: followee_preferential_attachment(row['source_node'],row['destination_node']),axis=1)
    df_final_test['followee_preferential_attachment'] = df_final_test.apply(lambda row: followee_preferential_attachment(row['source_node'],row['destination_node']),axis=1)

    df_final_train['follower_preferential_attachment'] = df_final_train.apply(lambda row: follower_preferential_attachment(row['source_node'],row['destination_n
```

```
ode']),axis=1)
    df_final_test['follower_preferential_attachment'] = df_final_test.apply(lamb
da row: follower_preferential_attachment(row['source_node'],row['destination_nod
e']),axis=1)

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

    hdf = HDFStore('/content/drive/MyDrive/Facebook/data/fea_sample/storage_samp
le_stage5.h5')
    hdf.put('train_df',df_final_train, format='table', data_columns=True)
    hdf.put('test_df',df_final_test, format='table', data_columns=True)
    hdf.close()
else:
    df_final_train = read_hdf('/content/drive/MyDrive/Facebook/data/fea_sample/s
torage_sample_stage5.h5', 'train_df',mode='r')
    df_final_test = read_hdf('/content/drive/MyDrive/Facebook/data/fea_sample/st
orage_sample_stage5.h5', 'test_df',mode='r')

print("Time taken for creation of dataframe is {}".format(datetime.datetime.now
() - startTime))
```

```
Current Time =  2021-09-19 10:36:08.749403
Time taken for creation of dataframe is 0:03:11.476205
```

In [62]:

```
# for Train data

x1 = list(df_final_train['svd_u_s_1'])
x2 = list(df_final_train['svd_u_s_2'])
x3 = list(df_final_train['svd_u_s_3'])
x4 = list(df_final_train['svd_u_s_4'])
x5 = list(df_final_train['svd_u_s_5'])
x6 = list(df_final_train['svd_u_s_6'])

x7 = list(df_final_train['svd_u_d_1'])
x8 = list(df_final_train['svd_u_d_2'])
x9 = list(df_final_train['svd_u_d_3'])
x10 = list(df_final_train['svd_u_d_4'])
x11 = list(df_final_train['svd_u_d_5'])
x12 = list(df_final_train['svd_u_d_6'])

y1 = list(df_final_train['svd_v_s_1'])
y2 = list(df_final_train['svd_v_s_2'])
y3 = list(df_final_train['svd_v_s_3'])
y4 = list(df_final_train['svd_v_s_4'])
y5 = list(df_final_train['svd_v_s_5'])
y6 = list(df_final_train['svd_v_s_6'])

y7 = list(df_final_train['svd_v_d_1'])
y8 = list(df_final_train['svd_v_d_2'])
y9 = list(df_final_train['svd_v_d_3'])
y10 = list(df_final_train['svd_v_d_4'])
y11 = list(df_final_train['svd_v_d_5'])
y12 = list(df_final_train['svd_v_d_6'])

print(np.shape(x1))
print(np.shape(x2))
print(np.shape(x3))
print(np.shape(x4))
print(np.shape(x5))
print(np.shape(x6))
print(np.shape(x7))
print(np.shape(x8))
print(np.shape(x9))
print(np.shape(x10))
print(np.shape(x11))
print(np.shape(x12))

print(np.shape(y1))
print(np.shape(y2))
print(np.shape(y3))
print(np.shape(y4))
print(np.shape(y5))
print(np.shape(y6))
print(np.shape(y7))
print(np.shape(y8))
print(np.shape(y9))
print(np.shape(y10))
print(np.shape(y11))
print(np.shape(y12))

train_u_source = []
train_u_destination = []
train_v_source = []
```

```
train_v_destination = []
train_u_s_dot = []
train_u_d_dot = []

for loop1 in range(0,len(x1)):
    train_u_source.append(x1[loop1])
    train_u_source.append(x2[loop1])
    train_u_source.append(x3[loop1])
    train_u_source.append(x4[loop1])
    train_u_source.append(x5[loop1])
    train_u_source.append(x6[loop1])

    train_u_destination.append(x7[loop1])
    train_u_destination.append(x8[loop1])
    train_u_destination.append(x9[loop1])
    train_u_destination.append(x10[loop1])
    train_u_destination.append(x11[loop1])
    train_u_destination.append(x12[loop1])

    dot_product = np.dot(train_u_source[loop1],train_u_destination[loop1])
    train_u_s_dot.append(dot_product)


for loop2 in range(0,len(y1)):
    train_v_source.append(y1[loop2])
    train_v_source.append(y2[loop2])
    train_v_source.append(y3[loop2])
    train_v_source.append(y4[loop2])
    train_v_source.append(y5[loop2])
    train_v_source.append(y6[loop2])

    train_v_destination.append(y7[loop2])
    train_v_destination.append(y8[loop2])
    train_v_destination.append(y9[loop2])
    train_v_destination.append(y10[loop2])
    train_v_destination.append(y11[loop2])
    train_v_destination.append(y12[loop2])

    dot_product = np.dot(train_v_source[loop2],train_v_destination[loop2])
    train_u_d_dot.append(dot_product)

print("****************************************")
print(np.shape(train_u_s_dot))
print(np.shape(train_u_d_dot))
```

```
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
********************************************
(100002,)
(100002,)
```

In [63]:

```
# for Test data

x1 = list(df_final_test['svd_u_s_1'])
x2 = list(df_final_test['svd_u_s_2'])
x3 = list(df_final_test['svd_u_s_3'])
x4 = list(df_final_test['svd_u_s_4'])
x5 = list(df_final_test['svd_u_s_5'])
x6 = list(df_final_test['svd_u_s_6'])

x7 = list(df_final_test['svd_u_d_1'])
x8 = list(df_final_test['svd_u_d_2'])
x9 = list(df_final_test['svd_u_d_3'])
x10 = list(df_final_test['svd_u_d_4'])
x11 = list(df_final_test['svd_u_d_5'])
x12 = list(df_final_test['svd_u_d_6'])

y1 = list(df_final_test['svd_v_s_1'])
y2 = list(df_final_test['svd_v_s_2'])
y3 = list(df_final_test['svd_v_s_3'])
y4 = list(df_final_test['svd_v_s_4'])
y5 = list(df_final_test['svd_v_s_5'])
y6 = list(df_final_test['svd_v_s_6'])

y7 = list(df_final_test['svd_v_d_1'])
y8 = list(df_final_test['svd_v_d_2'])
y9 = list(df_final_test['svd_v_d_3'])
y10 = list(df_final_test['svd_v_d_4'])
y11 = list(df_final_test['svd_v_d_5'])
y12 = list(df_final_test['svd_v_d_6'])

print(np.shape(x1))
print(np.shape(x2))
print(np.shape(x3))
print(np.shape(x4))
print(np.shape(x5))
print(np.shape(x6))
print(np.shape(x7))
print(np.shape(x8))
print(np.shape(x9))
print(np.shape(x10))
print(np.shape(x11))
print(np.shape(x12))

print(np.shape(y1))
print(np.shape(y2))
print(np.shape(y3))
print(np.shape(y4))
print(np.shape(y5))
print(np.shape(y6))
print(np.shape(y7))
print(np.shape(y8))
print(np.shape(y9))
print(np.shape(y10))
print(np.shape(y11))
print(np.shape(y12))


test_u_source = []
test_u_destination = []
```

```
test_v_source = []
test_v_destination = []
test_v_s_dot = []
test_v_d_dot = []

for loop3 in range(0,len(x1)):
    test_u_source.append(x1[loop3])
    test_u_source.append(x2[loop3])
    test_u_source.append(x3[loop3])
    test_u_source.append(x4[loop3])
    test_u_source.append(x5[loop3])
    test_u_source.append(x6[loop3])

    test_u_destination.append(x7[loop3])
    test_u_destination.append(x8[loop3])
    test_u_destination.append(x9[loop3])
    test_u_destination.append(x10[loop3])
    test_u_destination.append(x11[loop3])
    test_u_destination.append(x12[loop3])

    dot_product = np.dot(test_u_source[loop3],test_u_destination[loop3])
    test_v_s_dot.append(dot_product)

for loop4 in range(0,len(y1)):
    test_v_source.append(y1[loop4])
    test_v_source.append(y2[loop4])
    test_v_source.append(y3[loop4])
    test_v_source.append(y4[loop4])
    test_v_source.append(y5[loop4])
    test_v_source.append(y6[loop4])

    test_v_destination.append(y7[loop4])
    test_v_destination.append(y8[loop4])
    test_v_destination.append(y9[loop4])
    test_v_destination.append(y10[loop4])
    test_v_destination.append(y11[loop4])
    test_v_destination.append(y12[loop4])

    dot_product = np.dot(test_v_source[loop4],test_v_destination[loop4])
    test_v_d_dot.append(dot_product)

print("****************************************")
print(np.shape(test_v_s_dot))
print(np.shape(test_v_d_dot))
```

```
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
*******************************************
(50002,)
(50002,)
```

In [64]:

```
startTime = datetime.datetime.now()
print("Current Time = ",startTime)

if not os.path.isfile('/content/drive/MyDrive/Facebook/data/fea_sample/storage_s
ample_stage6.h5'):
    #===============================================================================
========================
    df_final_train['s_dot'] = np.array(train_u_s_dot)
    df_final_train['d_dot'] = np.array(train_u_d_dot)

    df_final_test['s_dot'] = np.array(test_v_s_dot)
    df_final_test['d_dot'] = np.array(test_v_d_dot)

    hdf = HDFStore('/content/drive/MyDrive/Facebook/data/fea_sample/storage_samp
le_stage6.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('/content/drive/MyDrive/Facebook/data/fea_sample/s
torage_sample_stage6.h5', 'train_df',mode='r')
    df_final_test = read_hdf('/content/drive/MyDrive/Facebook/data/fea_sample/st
orage_sample_stage6.h5', 'test_df',mode='r')

print("Time taken for creation of dataframe is {}".format(datetime.datetime.now
() - startTime))
```

```
Current Time =  2021-09-19 10:41:16.927772
Time taken for creation of dataframe is 0:00:08.322808
```

In [5]:

```
#reading
from pandas import read_hdf
df_final_train = read_hdf('/content/drive/MyDrive/Facebook/data/fea_sample/storage_sample_stage6.h5', 'train_df',mode='r')
df_final_test = read_hdf('/content/drive/MyDrive/Facebook/data/fea_sample/storage_sample_stage6.h5', 'test_df',mode='r')
```

In [6]:

```
df_final_test.loc[:,'adar_index':][:10]
```

Out[6]:

	adar_index	follows_back	same_comp	shortest_path	weight_in	weight_out	weight_f1	we
0	0.000000	1	1	2	0.258199	0.377964	0.636163	0
1	0.000000	1	1	7	0.235702	0.707107	0.942809	0
2	0.000000	0	1	5	0.301511	0.242536	0.544047	0
3	0.000000	0	1	3	0.162221	0.301511	0.463733	0
4	6.136433	0	1	2	0.188982	0.250000	0.438982	0
5	0.000000	0	0	-1	0.588969	0.301511	0.890481	0
6	0.000000	1	1	-1	1.000000	0.353553	1.353553	0
7	3.095903	1	1	2	0.250000	0.288675	0.538675	0
8	0.000000	0	0	-1	0.588969	0.301511	0.890481	0
9	0.000000	1	1	-1	0.377964	1.000000	1.377964	0

In [7]:

```
df_final_train.columns
```

Out[7]:

```
Index(['source_node', 'destination_node', 'indicator_link',
       'jaccard_followers', 'jaccard_followees', 'cosine_followers',
       'cosine_followees', 'num_followers_s', 'num_followees_s',
       'num_followees_d', 'inter_followers', 'inter_followees', 'ada
r_index',
       'follows_back', 'same_comp', 'shortest_path', 'weight_in', 'w
eight_out',
       'weight_f1', 'weight_f2', 'weight_f3', 'weight_f4', 'page_ran
k_s',
       'page_rank_d', 'katz_s', 'katz_d', 'hubs_s', 'hubs_d', 'autho
rities_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',
       'followee_preferential_attachment', 'follower_preferential_at
tachment',
       's_dot', 'd_dot'],
      dtype='object')
```

In [8]:

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

In [9]:

```
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,i
nplace=True)
```

In [11]:

```
df_final_test.columns, df_final_test.shape
```

Out[11]:

```
(Index(['jaccard_followers', 'jaccard_followees', 'cosine_follower
s',
        'cosine_followees', 'num_followers_s', 'num_followees_s',
        'num_followees_d', 'inter_followers', 'inter_followees', 'ad
ar_index',
        'follows_back', 'same_comp', 'shortest_path', 'weight_in',
'weight_out',
        'weight_f1', 'weight_f2', 'weight_f3', 'weight_f4', 'page_ra
nk_s',
        'page_rank_d', 'katz_s', 'katz_d', 'hubs_s', 'hubs_d', 'auth
orities_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',
        'followee_preferential_attachment', 'follower_preferential_a
ttachment',
        's_dot', 'd_dot'],
       dtype='object'), (50002, 55))
```

In [12]:

```
df_final_train.columns, df_final_train.shape
```

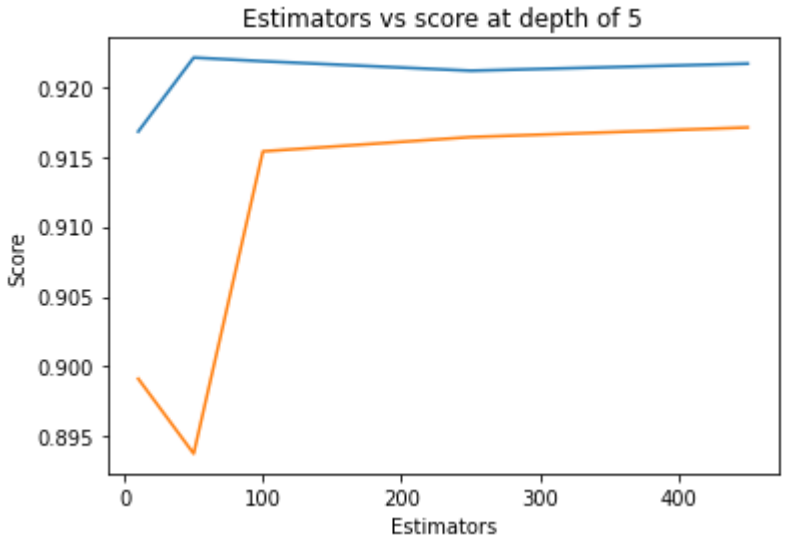
Out[12]:

```
(Index(['jaccard_followers', 'jaccard_followees', 'cosine_follower
s',
        'cosine_followees', 'num_followers_s', 'num_followees_s',
        'num_followees_d', 'inter_followers', 'inter_followees', 'ad
ar_index',
        'follows_back', 'same_comp', 'shortest_path', 'weight_in',
'weight_out',
        'weight_f1', 'weight_f2', 'weight_f3', 'weight_f4', 'page_ra
nk_s',
        'page_rank_d', 'katz_s', 'katz_d', 'hubs_s', 'hubs_d', 'auth
orities_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',
        'followee_preferential_attachment', 'follower_preferential_a
ttachment',
        's_dot', 'd_dot'],
       dtype='object'), (100002, 55))
```

In [24]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
startTime = datetime.datetime.now()
print("Current Time = ",startTime)

estimators = [10,50,100,250,450]
train_scores = []
test_scores = []
for i in estimators:
    clf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='g
ini',
            max_depth=5, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=52, min_samples_split=120,
            min_weight_fraction_leaf=0.0, n_estimators=i,random_state=25,verbose
=0,warm_start=False)
    clf.fit(df_final_train,y_train)
    train_sc = f1_score(y_train,clf.predict(df_final_train))
    test_sc = f1_score(y_test,clf.predict(df_final_test))
    test_scores.append(test_sc)
    train_scores.append(train_sc)
    print('Estimators = ',i,'Train Score',train_sc,'test Score',test_sc)
plt.plot(estimators,train_scores,label='Train Score')
plt.plot(estimators,test_scores,label='Test Score')
plt.xlabel('Estimators')
plt.ylabel('Score')
plt.title('Estimators vs score at depth of 5')

print("Time taken for creation of dataframe is {}".format(datetime.datetime.now
() - startTime))
```

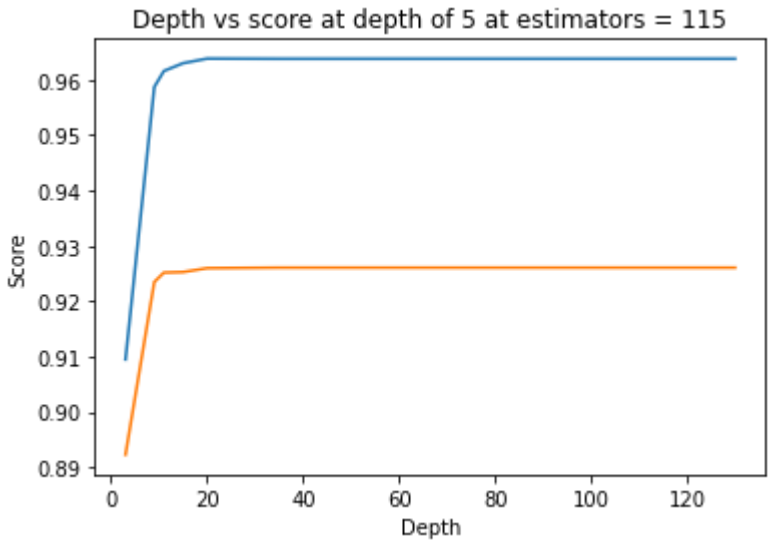
```
Current Time =  2021-09-20 07:41:10.553767
Estimators =  10 Train Score 0.9168054647804179 test Score 0.8991258
141926637
Estimators =  50 Train Score 0.9220895584588777 test Score 0.8937688
248990665
Estimators =  100 Train Score 0.9218287697728152 test Score 0.915386
0707603692
Estimators =  250 Train Score 0.9211508895233912 test Score 0.916405
9383071205
Estimators =  450 Train Score 0.9216563839799227 test Score 0.917095
2910797137
Time taken for creation of dataframe is 0:02:29.644470
```



In [25]:

```
startTime = datetime.datetime.now()
print("Current Time = ",startTime)

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

print("Time taken for creation of dataframe is {}".format(datetime.datetime.now
() - startTime))
```

```
Current Time =  2021-09-20 07:43:42.154849
depth =  3 Train Score 0.9094969294708766 test Score 0.8923076923076
922
depth =  9 Train Score 0.9586727982787979 test Score 0.9234631450149
783
depth =  11 Train Score 0.9615533269594579 test Score 0.925167488307
42
depth =  15 Train Score 0.9629599545122248 test Score 0.925264355226
0185
depth =  20 Train Score 0.963790564217155 test Score 0.9259657330720
111
depth =  35 Train Score 0.963760461525961 test Score 0.9260593800703
796
depth =  50 Train Score 0.963760461525961 test Score 0.9260593800703
796
depth =  70 Train Score 0.963760461525961 test Score 0.9260593800703
796
depth =  130 Train Score 0.963760461525961 test Score 0.926059380070
3796
```




```
Time taken for creation of dataframe is 0:04:21.631803
```

In [26]:

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

startTime = datetime.datetime.now()
print("Current Time = ",startTime)

param_dist = {"n_estimators":sp_randint(105,125),
              "max_depth": sp_randint(10,15),
              "min_samples_split": sp_randint(110,190),
              "min_samples_leaf": sp_randint(25,65)}

clf = RandomForestClassifier(random_state=25)

rf_random = RandomizedSearchCV(clf, param_distributions=param_dist,
                                    n_iter=5,cv=10,scoring='f1',random_state=25)

rf_random.fit(df_final_train,y_train)
```

```
Current Time =  2021-09-20 07:48:03.804317
```

Out[26]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=RandomForestClassifier(bootstrap=True,
                                                    ccp_alpha=0.0,
                                                    class_weight=Non
e,
                                                    criterion='gin
i',
                                                    max_depth=None,
                                                    max_features='au
to',
                                                    max_leaf_nodes=N
one,
                                                    max_samples=Non
e,
                                                    min_impurity_dec
rease=0.0,
                                                    min_impurity_spl
it=None,
                                                    min_samples_leaf
=1,
                                                    min_samples_spli
t=2,
                                                    min_weight_fract
ion_leaf=0.0,
                                                    n_estimators=10
0,
                                                    n_job...
                                            'min_samples_leaf': <scipy.s
tats._distn_infrastructure.rv_frozen object at 0x7f4eff3dc650>,
                                            'min_samples_split': <scipy.
stats._distn_infrastructure.rv_frozen object at 0x7f4eff30ee10>,
                                            'n_estimators': <scipy.stat
s._distn_infrastructure.rv_frozen object at 0x7f4eff30e5d0>},
                   pre_dispatch='2*n_jobs', random_state=25, refit=T
rue,
                   return_train_score=False, scoring='f1', verbose=
0)
```

In [27]:

```
#print('mean test scores',rf_random.cv_results_['mean_test_score'])
#print('mean train scores',rf_random.cv_results_['mean_train_score'])
print("*****************************************************")
print(rf_random.best_estimator_)

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

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

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

print("Time taken for creation of dataframe is {}".format(datetime.datetime.now
() - startTime))
```

```
*****************************************************
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=N
one,
                       criterion='gini', max_depth=14, max_features
='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split
=None,
                       min_samples_leaf=28, min_samples_split=111,
                       min_weight_fraction_leaf=0.0, n_estimators=12
1,
                       n_jobs=None, oob_score=False, random_state=2
5, verbose=0,
                       warm_start=False)

Train f1 score 0.9643266955735856
Test f1 score 0.9263264402706634
Time taken for creation of dataframe is 0:22:50.695773
```

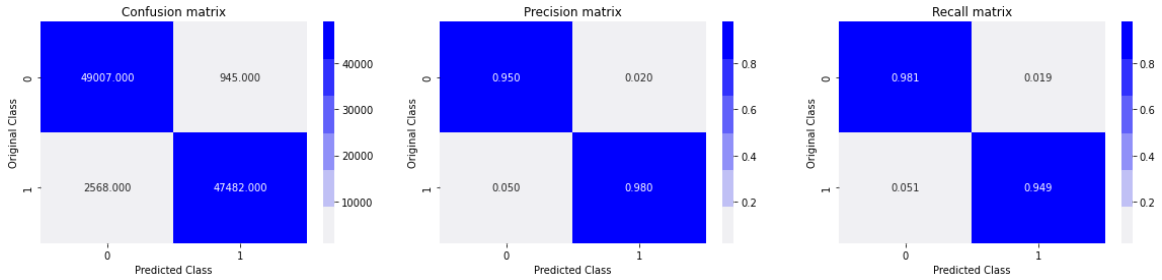
In [16]:

```
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()
```

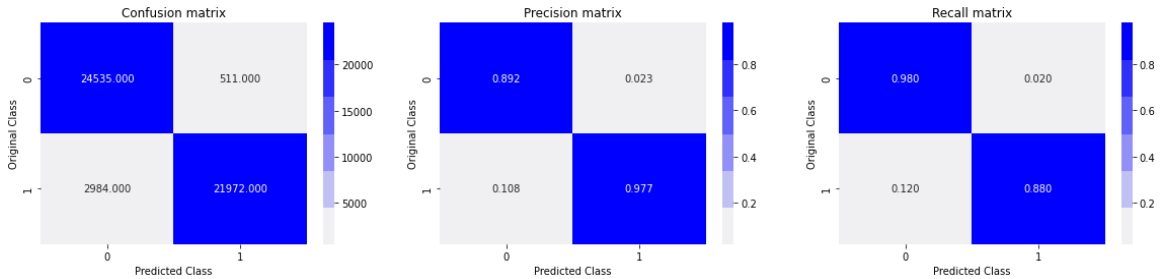
In [28]:

```
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
```

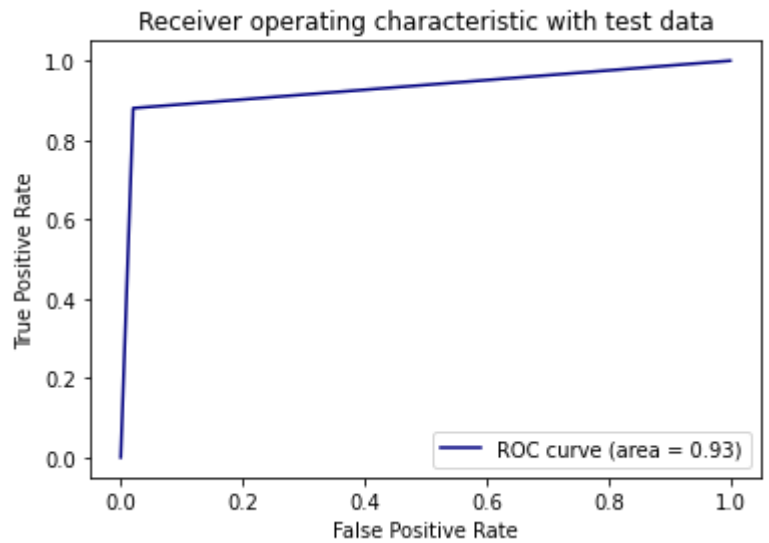



```
Test confusion_matrix
```



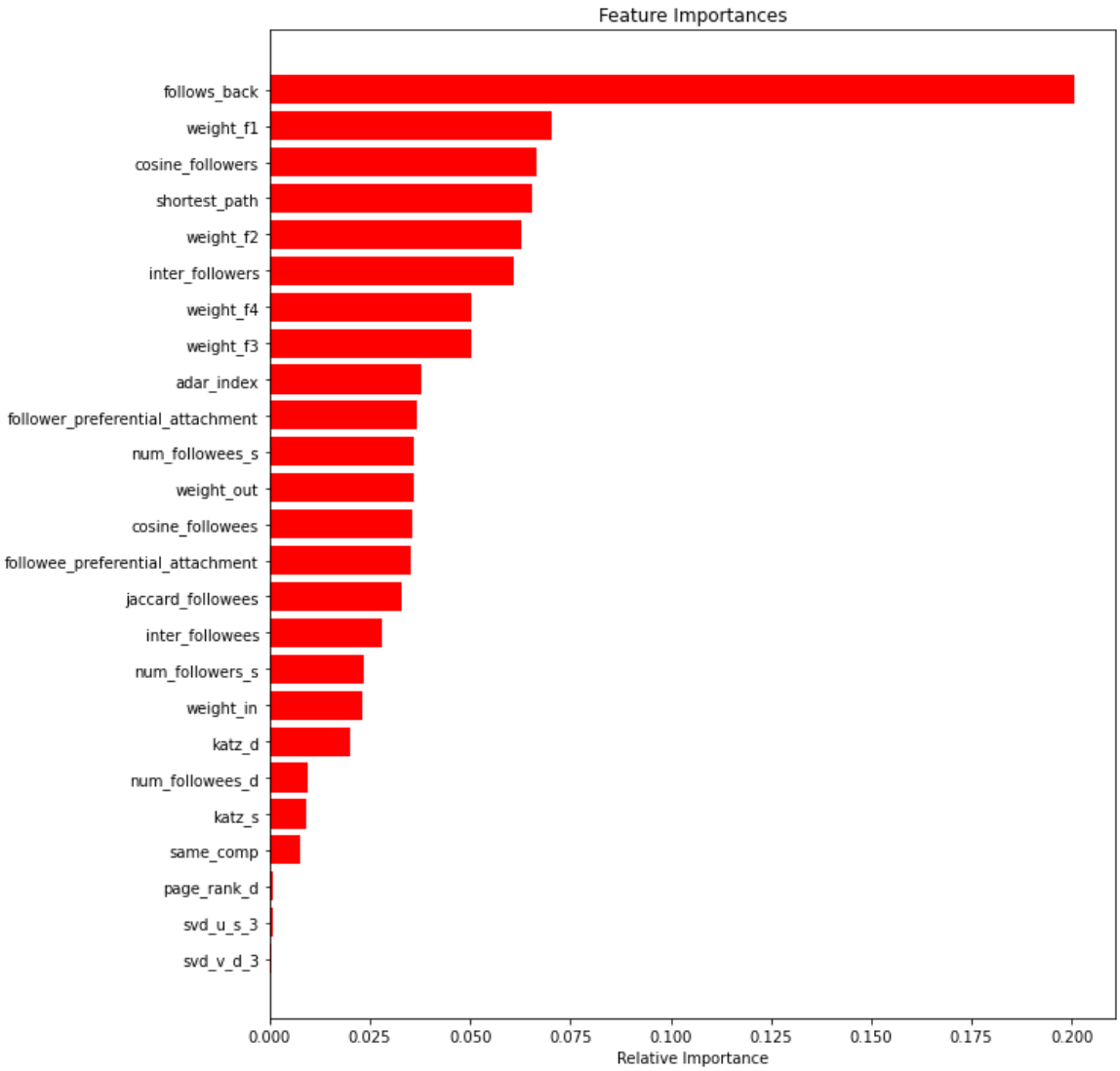
In [29]:

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



In [30]:

```
features = df_final_train.columns
importances = clf.feature_importances_
indices = (np.argsort(importances))[-25:]
plt.figure(figsize=(10,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='r', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



Hyper-parameter tunning using XGBoost

In [34]:

```
import xgboost as xgb
clf = xgb.XGBClassifier()
param_dist = {"n_estimators":sp_randint(105,125),
              "max_depth": sp_randint(10,15)
              }
model = RandomizedSearchCV(clf, param_distributions=param_dist,
                                   n_iter=5,cv=3,scoring='f1',random_state=25,re
turn_train_score=True)


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

```
mean test scores [0.97965255 0.97967356 0.97923936 0.97979006 0.9797
2997]
mean train scores [0.99998501 1.         0.99452855 0.99711555 0.997
42171]
```

In [35]:

```
print(model.best_estimator_)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=11,
              min_child_weight=1, missing=None, n_estimators=110, n_
jobs=1,
              nthread=None, objective='binary:logistic', random_stat
e=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=No
ne,
              silent=None, subsample=1, verbosity=1)
```

In [36]:

```
clf=xgb.XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=11, min_child_weight=1, missing=None, n_estimators=110,
       n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1)
```

In [37]:

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

In [38]:

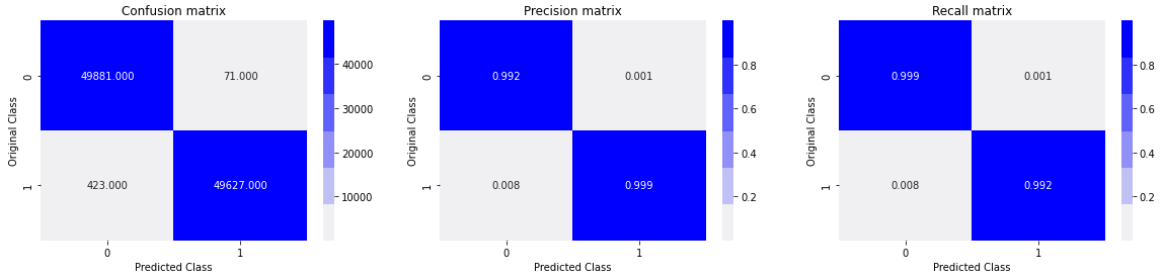
```
from sklearn.metrics import f1_score
print('Train f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))
```

```
Train f1 score 0.9950475197497693
Test f1 score 0.9268075167215203
```

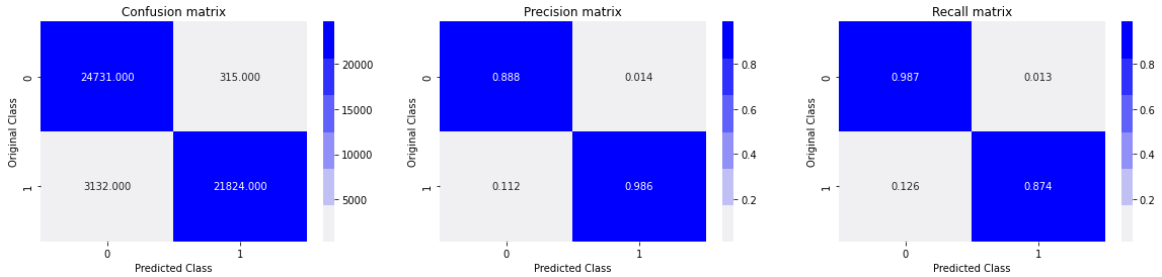
In [39]:

```
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
```

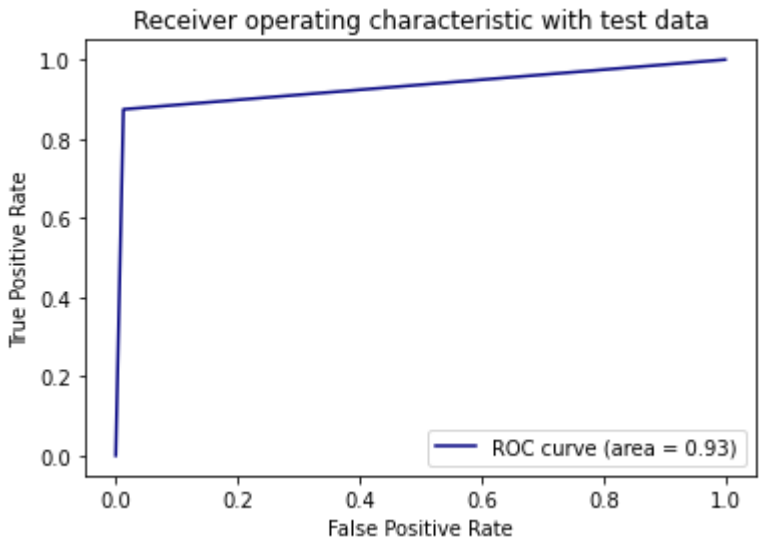



```
Test confusion_matrix
```



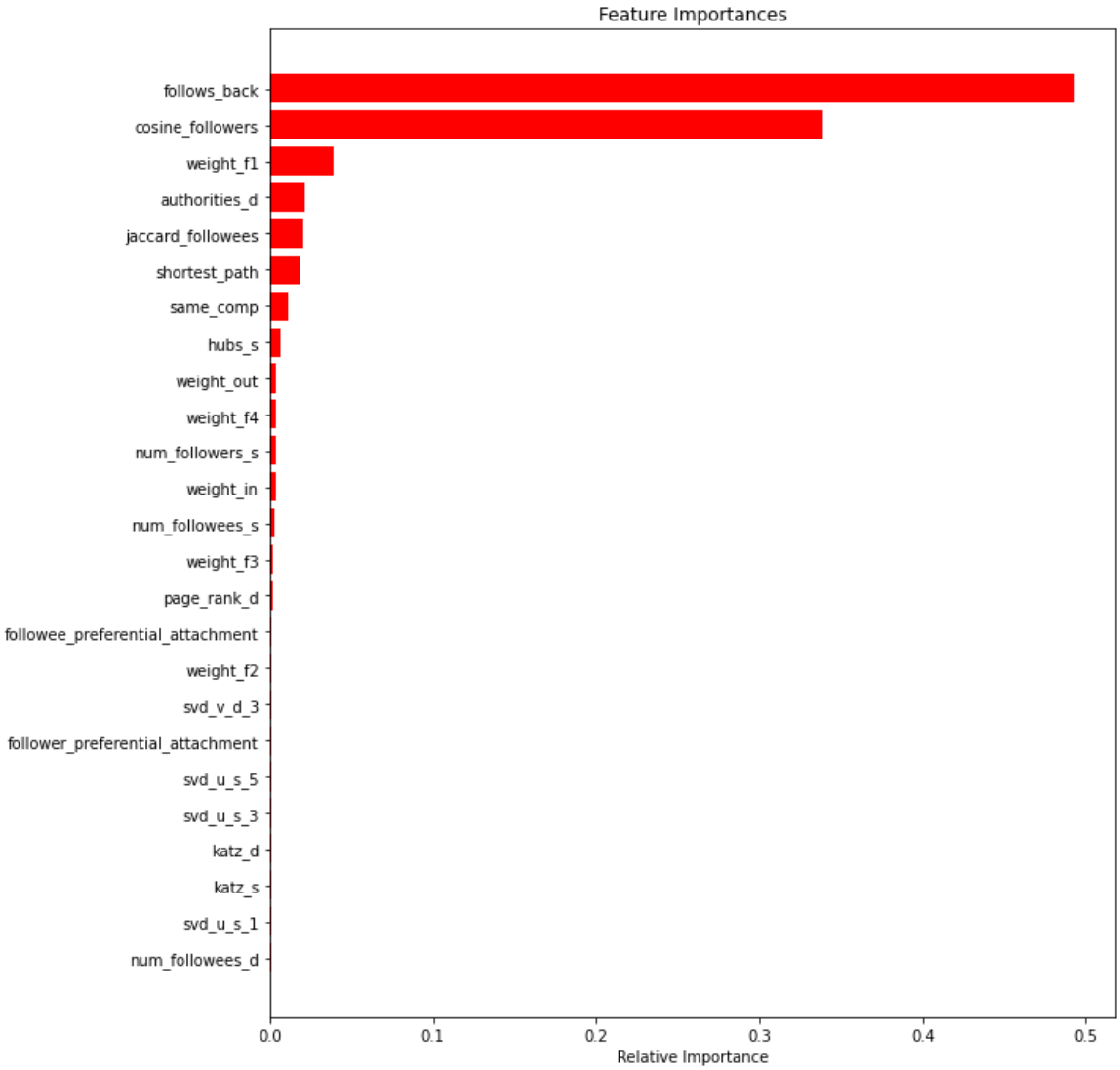
In [40]:

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



In [41]:

```
features = df_final_train.columns
importances = clf.feature_importances_
indices = (np.argsort(importances))[-25:]
plt.figure(figsize=(10,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='r', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



In [42]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model ","Train f1_score","Test f1_score"]
x.add_row(["RandomForest ",0.9643266955735856,0.9263264402706634])
x.add_row(["XGBClassifier ",0.9950475197497693,0.9268075167215203])
print(x)
```

```
+----------------+--------------------+--------------------+
|     Model      |   Train f1_score   |   Test f1_score    |
+----------------+--------------------+--------------------+
| RandomForest   | 0.9643266955735856 | 0.9263264402706634 |
| XGBClassifier  | 0.9950475197497693 | 0.9268075167215203 |
+----------------+--------------------+--------------------+
```