Social network Graph Link Prediction - Facebook Challenge

In [ ]:

```
#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
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
```

In [ ]:

```
#reading
from pandas import read_hdf
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.describe
```

Out[ ]:

```
<bound method NDFrame.describe of         source_node  destination_node  indicator_link
jaccard_followers  \
0            273084           1505602               1                  0
1            832016           1543415               1                  0
2           1325247            760242               1                  0
3           1368400           1006992               1                  0
4            140165           1708748               1                  0
...             ...               ...             ...                ...
99997        139353            893843               0                  0
99998        910842            704068               0                  0
99999        794228           1172755               0                  0
100000       949992           1854931               0                  0
100001      1642037           1090977               0                  0

        jaccard_followees  cosine_followers  cosine_followees  \
```

```
0                 0.000000          0.000000          0.000000
1                 0.187135          0.028382          0.343828
2                 0.369565          0.156957          0.566038
3                 0.000000          0.000000          0.000000
4                 0.000000          0.000000          0.000000
...                    ...               ...               ...
99997             0.000000          0.000000          0.000000
99998             0.000000          0.000000          0.000000
99999             0.000000          0.000000          0.000000
100000            0.000000          0.000000          0.000000
100001            0.000000          0.000000          0.000000

        num_followers_s  num_followees_s  num_followees_d  ...     svd_v_s_3  \
0                     6               15                8  ...  1.983691e-06
1                    94               61              142  ... -6.236048e-11
2                    28               41               22  ... -2.380564e-19
3                    11                5                7  ...  6.058498e-11
4                     1               11                3  ...  1.197283e-07
...                 ...              ...              ...  ...           ...
99997                 7                1               10  ...  0.000000e+00
99998                 0                4                1  ...  1.336987e-12
99999                 0                5                1  ...  2.154438e-11
100000                1                2                0  ...  1.620187e-11
100001                1               14                0  ...  7.778089e-09

           svd_v_s_4     svd_v_s_5     svd_v_s_6     svd_v_d_1     svd_v_d_2  \
0       1.545075e-13  8.108434e-13  1.719702e-14 -1.355368e-12  4.675307e-13
1       1.345726e-02  3.703479e-12  2.251737e-10  1.245101e-12 -1.636948e-10
2      -7.021227e-19  1.940403e-19 -3.365389e-19 -1.238370e-18  1.438175e-19
3       1.514614e-11  1.513483e-12  4.498061e-13 -9.818087e-10  3.454672e-11
4       1.999809e-14  3.360247e-13  1.407670e-14  0.000000e+00  0.000000e+00
...              ...           ...           ...           ...           ...
99997   0.000000e+00  0.000000e+00  0.000000e+00 -3.303718e-12  1.538318e-13
99998   4.493330e-15  4.528679e-14  5.475207e-18  0.000000e+00  0.000000e+00
99999   1.566738e-12  2.294564e-13  3.493379e-14  0.000000e+00  0.000000e+00
100000  1.325874e-15  2.066643e-14  2.662102e-16 -1.142753e-17  5.200344e-17
100001  8.005625e-13  9.429577e-11  7.386157e-14 -1.657134e-14  2.085059e-14

           svd_v_d_3     svd_v_d_4     svd_v_d_5     svd_v_d_6
0       1.128591e-06  6.616550e-14  9.771077e-13  4.159752e-14
1      -3.112650e-10  6.738902e-02  2.607801e-11  2.372904e-09
2      -1.852863e-19 -5.901864e-19  1.629341e-19 -2.572452e-19
3       5.213635e-08  9.595823e-13  3.047045e-10  1.246592e-13
4       0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
...              ...           ...           ...           ...
99997   1.296745e-06  2.990887e-13  1.589668e-12  7.338551e-14
99998   0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
99999   0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
100000  3.858875e-15  2.173437e-17  2.241477e-16  3.528355e-20
100001  2.107704e-07  2.652994e-12  2.004727e-14  2.805020e-14

[100002 rows x 54 columns]>
```

In [ ]:

```
df_final_train.columns
```

Out[ ]:

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

In [ ]:

```
df_final_train.head()
```

Out[ ]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	nun
0	273084	1505602	1	0	0.000000	0.000000	0.000000	
1	832016	1543415	1	0	0.187135	0.028382	0.343828	
2	1325247	760242	1	0	0.369565	0.156957	0.566038	
3	1368400	1006992	1	0	0.000000	0.000000	0.000000	
4	140165	1708748	1	0	0.000000	0.000000	0.000000	

5 rows × 54 columns

In [ ]:

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

In [ ]:

```
df_final_train=df_final_train.drop('indicator_link',axis=1)
df_final_test=df_final_test.drop('indicator_link',axis=1)
```

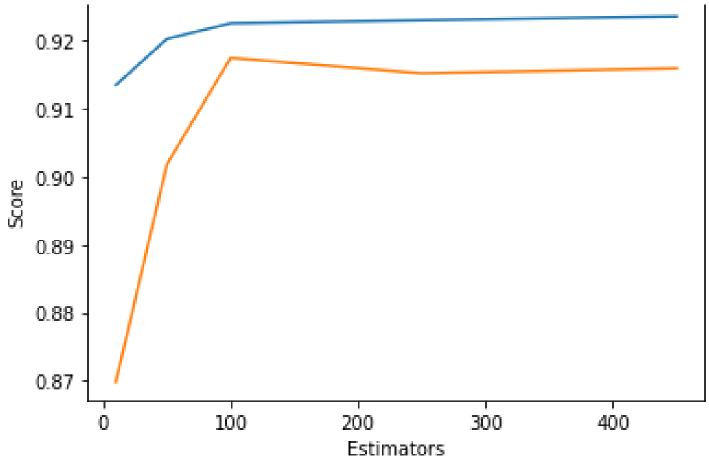
In [ ]:

```
estimators = [10,50,100,250,450]
train_scores = []
test_scores = []
for i in estimators:
    clf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=5, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0,
            min_samples_leaf=52, min_samples_split=120,
            min_weight_fraction_leaf=0.0, n_estimators=i, n_jobs=-1,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('Estimators = ',i,'Train Score',train_sc,'test Score',test_sc)
plt.plot(estimators,train_scores,label='Train Score')
plt.plot(estimators,test_scores,label='Test Score')
plt.xlabel('Estimators')
plt.ylabel('Score')
plt.title('Estimators vs score at depth of 5')
```

```
Estimators =  10 Train Score 0.9134734626246626 test Score 0.8698418194014671
Estimators =  50 Train Score 0.9202517042475092 test Score 0.9018085650912813
Estimators =  100 Train Score 0.9225489170241707 test Score 0.917456249080863
Estimators =  250 Train Score 0.9229916897506927 test Score 0.9151985855907053
Estimators =  450 Train Score 0.923543372120984 test Score 0.9159283654959068
```

Out[ ]:

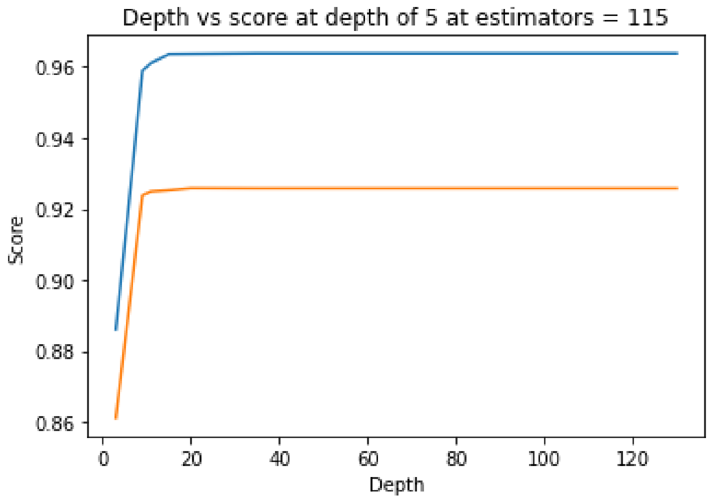
```
Text(0.5, 1.0, 'Estimators vs score at depth of 5')
```



In [ ]:

```
depths = [3,9,11,15,20,35,50,70,130]
train_scores = []
test_scores = []
for i in depths:
    clf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=i, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0,
            min_samples_leaf=52, min_samples_split=120,
            min_weight_fraction_leaf=0.0, n_estimators=115, n_jobs=-1,random_state=25,ve
rbose=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()
```

```
depth =  3 Train Score 0.8859874293561507 test Score 0.8611105111979785
depth =  9 Train Score 0.9588823655104346 test Score 0.9238249594813615
depth =  11 Train Score 0.961099653564426 test Score 0.9248900393543363
depth =  15 Train Score 0.963534859215579 test Score 0.9252507411846338
depth =  20 Train Score 0.9635935629198207 test Score 0.9258183960279391
depth =  35 Train Score 0.9637559104653287 test Score 0.9257676846336792
depth =  50 Train Score 0.9637559104653287 test Score 0.9257676846336792
depth =  70 Train Score 0.9637559104653287 test Score 0.9257676846336792
depth =  130 Train Score 0.9637559104653287 test Score 0.9257676846336792
```



In [ ]:

```
from sklearn.metrics import f1_score
```

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

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

clf = RandomForestClassifier(random_state=25,n_jobs=-1)

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

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

```
mean test scores [0.96213851 0.96183154 0.96037467 0.96207089 0.96285674]
```

In [ ]:

```
print('mean train scores',rf_random.cv_results_['mean_train_score'])
```

```
mean train scores [0.96314275 0.96254055 0.96088891 0.9627027  0.96395558]
```

In [ ]:

```
print(rf_random.best_estimator_)
```

```
RandomForestClassifier(max_depth=14, min_samples_leaf=28, min_samples_split=111,
                       n_estimators=121, n_jobs=-1, random_state=25)
```

In [ ]:

```
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)
```

In [ ]:

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

In [ ]:

```
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 0.9636780303798496
Test f1 score 0.9257234185733513
```

In [ ]:

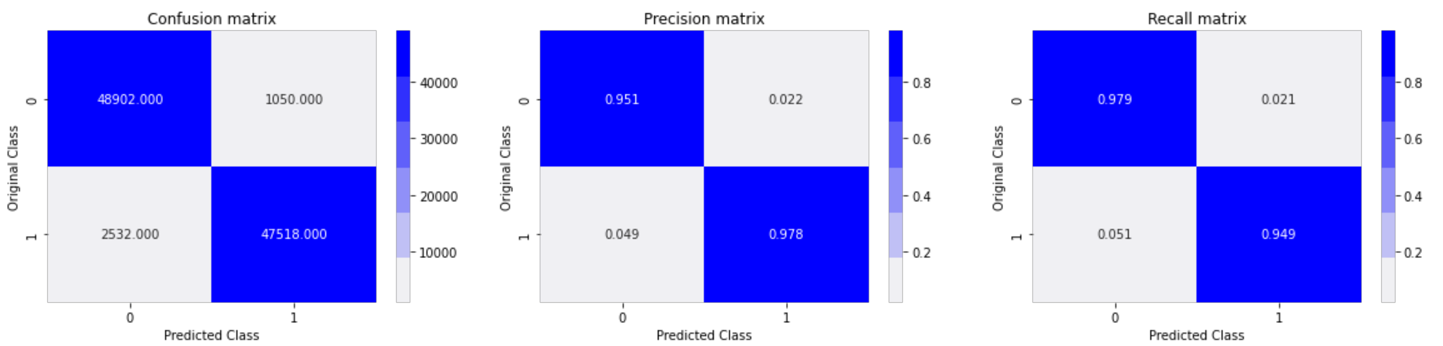
```
%matplotlib inline
from sklearn.metrics import confusion_matrix
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)

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

    plt.show()
```

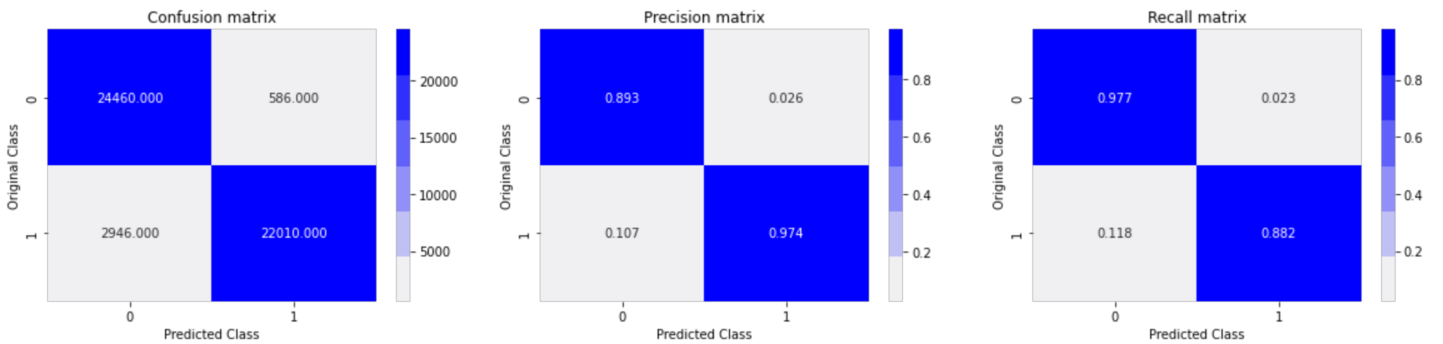
In [ ]:

```
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
```



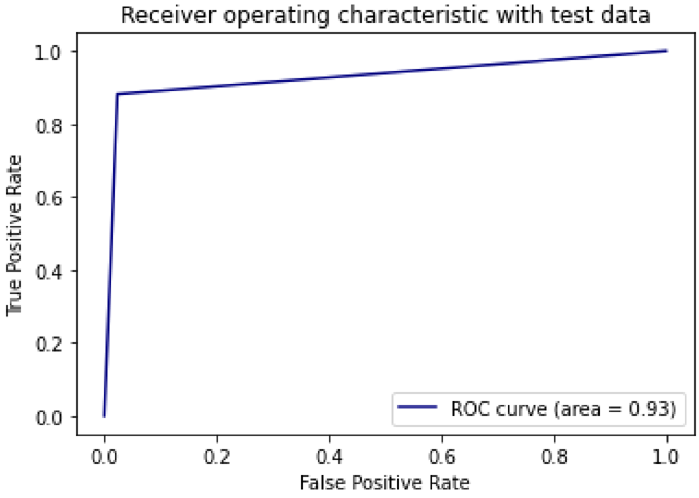

```
Test confusion_matrix
```



In [ ]:

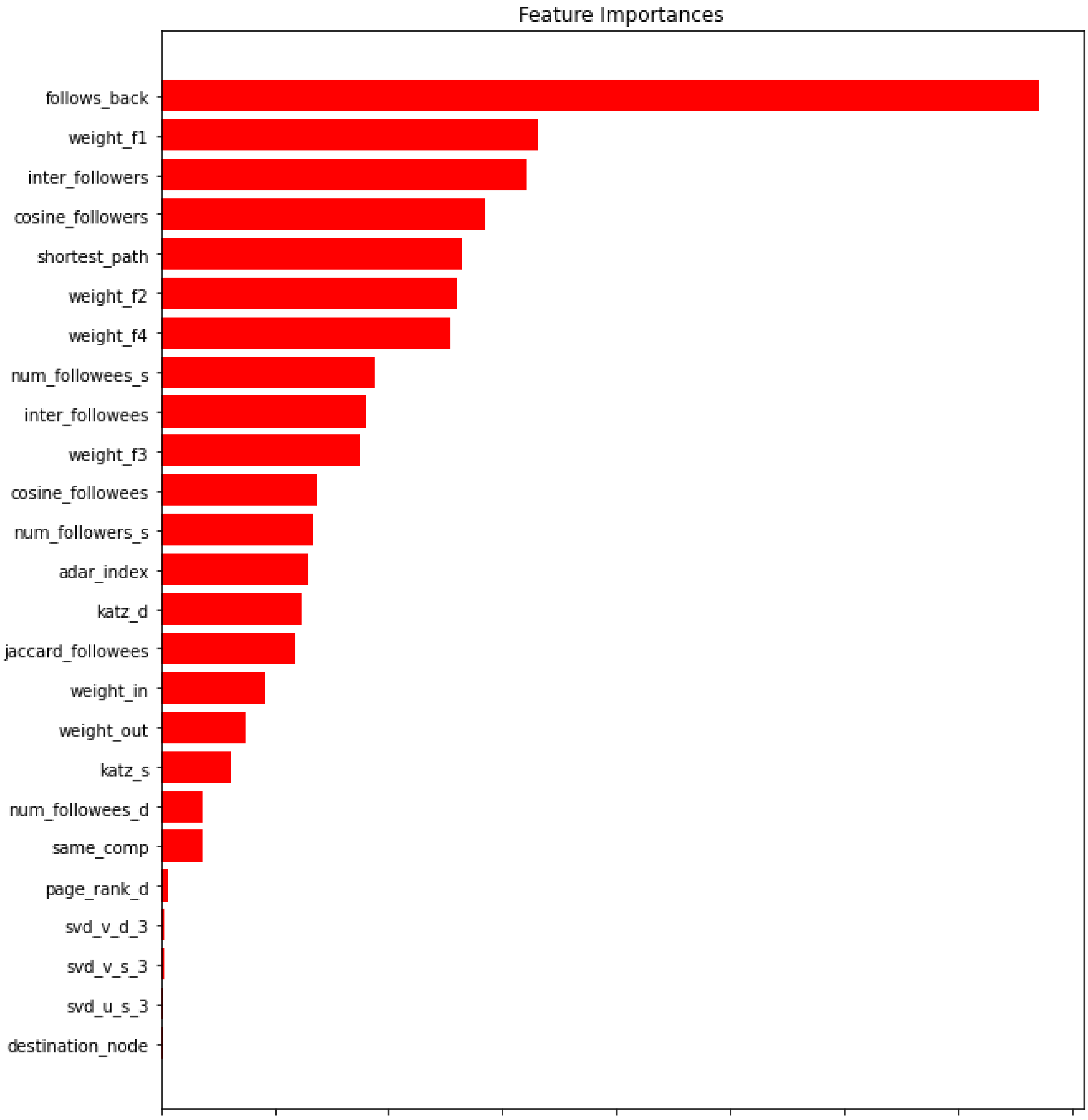
```
from sklearn.metrics import roc_curve, auc
fpr,tpr,ths = roc_curve(y_test,y_test_pred)
auc_sc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='navy',label='ROC curve (area = %0.2f)' % auc_sc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic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

```
plt.show()
```



In [ ]:

```
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()
```



0.000 0.025 0.050 0.075 0.100 0.125 0.150 0.175 0.200
Relative Importance

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.**

In [ ]:

```
train_graph = nx.read_edgelist('train_pos_after_eda.csv',delimiter=',',create_using=nx.DiGraph(),nodetype=int)
test_graph = nx.read_edgelist('test_pos_after_eda.csv',delimiter=',',create_using=nx.DiGraph(),nodetype=int)
```

In [ ]:

```
from tqdm import tqdm
###https://colab.research.google.com/drive/151MIcMDCjpYl8DKCEICnm4b4lkflLJRJ
def feat(df,viz):
  num_followers_s=[]
  num_followees_s=[]
  num_followers_d=[]
  num_followees_d=[]
  inter_followers=[]
  inter_followees=[]
  for q,ran in tqdm(df.iterrows()):
    try:
      s1=set(viz.predecessors(ran['source_node']))
      s2=set(viz.successors(ran['source_node']))
    except:
      s1=set()
      s2=set()
    try:
      d1=set(viz.predecessors(ran['destination_node']))
      d2=set(viz.successors(ran['destination_node']))
    except:
      d1=set()
      d2=set()
    num_followers_s.append(len(s1))
    num_followees_s.append(len(s2))
    num_followers_d.append(len(d1))
    num_followees_d.append(len(d2))
    inter_followers.append(len(s1.intersection(d1)))
    inter_followees.append(len(s2.intersection(d2)))
  return num_followers_s,num_followees_s,num_followers_d,num_followees_d,inter_followers,inter_followees
```

In [ ]:

```
tr_num_followers_s,tr_num_followees_s,tr_num_followers_d,tr_num_followees_d,tr_inter_followers,tr_inter_followees=feat(df_final_train,train_graph)
test_num_followers_s,test_num_followees_s,test_num_followers_d,test_num_followees_d,test_inter_followers,test_inter_followees=feat(df_final_test,test_graph)
```

```
100002it [00:10, 9809.40it/s]
50002it [00:03, 14064.44it/s]
```

In [ ]:

```
df_final_train['num_followers_d']=tr_num_followers_d
df_final_test['num_followers_d']=test_num_followers_d
```

In [ ]:

```
def preferential_attachment(df_fin):
    ###https://in.coursera.org/lecture/python-social-network-analysis/preferential-attachment-model-abipd
    link_pred_followees=[]
    link_pred_followers=[]
    link_pred_followees=df_fin['num_followees_s']*df_fin['num_followees_d']
    link_pred_followers=df_fin['num_followers_s']*df_fin['num_followers_d']
    return link_pred_followers,link_pred_followees
```

In [ ]:

```
df_final_train[' link_pred_followers'], df_final_train['link_pred_followees']= preferential_attachment(df_final_train)

df_final_test[' link_pred_followers'], df_final_test['link_pred_followees']= preferential_attachment(df_final_test)
```

In [ ]:

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

In [ ]:

```
sadj_col = sorted(train_graph.nodes())
sadj_dict = { val:idx for idx,val in enumerate(sadj_col)}
```

In [ ]:

```
U, s, V = svds(Adj,k=6)
print('U Shape',U.shape)
print('V Shape',V.shape)
print('s Shape',s.shape)
```

```
U Shape (1780722, 6)
V Shape (6, 1780722)
s Shape (6,)
```

In [ ]:

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

In [ ]:

```
df_final_train.columns
```

Out[ ]:

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

In [ ]:

```
def dot_prod(df):
```

```
    ###https://www.geeksforgeeks.org/numpy-dot-python/
    ### https://storage.googleapis.com/kaggle-forum-message-attachments/2594/supervised_l
ink_prediction.pdf
    dot_u=[]
    dot_v=[]
    for index,ran in df.iterrows():
        s1=svd(ran['source_node'],U)
        d1=svd(ran['destination_node'],U)
        s2=svd(ran['source_node'],V.T)
        d2=svd(ran['destination_node'],V.T)
        dot_u.append(np.dot(s1,d1))
        dot_v.append(np.dot(s2,d2))

    return dot_u,dot_v


(df_final_train['dot_u'],df_final_train['dot_v'])=dot_prod(df_final_train)
(df_final_test['dot_u'],df_final_test['dot_v'])=dot_prod(df_final_test)
```

In [ ]:

```
df_final_train.columns
```

Out[ ]:

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

In [ ]:

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

In [ ]:

```
df_final_train.columns
```

Out[ ]:

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

In [ ]:

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

```python
from scipy.stats import uniform
from xgboost import XGBClassifier

###https://www.projectpro.io/recipes/use-xgboost-classifier-and-regressor-in-python

param = {'min_child_weight': [1, 5, 10],
        'max_depth': [15,20,30,40],
        'learning_rate':[0.03,0.05,0.1,0.15,0.2],
        'n_estimators' :[20,50,100,150],
        'subsample': [0.6, 0.8, 1.0],
        'colsample_bytree': [0.6, 0.8, 1.0],
        'gamma': [1, 1.5, 2, 5]}

clf =  XGBClassifier(random_state=25,n_jobs=-1)

rnd_src = RandomizedSearchCV(clf, param_distributions=param,n_iter=3,
                             cv=3,scoring='f1',random_state=25,return_train_score=True)

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

```
mean test scores [0.97800923 0.97994223 0.97307308]
mean train scores [0.98788082 0.99413852 0.97810217]
```

In [ ]:

```python
clf1=rnd_src.best_estimator_
clf1.fit(df_final_train,y_train)

y_train_pred = clf1.predict(df_final_train)
y_test_pred = clf1.predict(df_final_test)

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 0.9950570342205324
Test f1 score 0.9311120543293718
```

In [ ]:

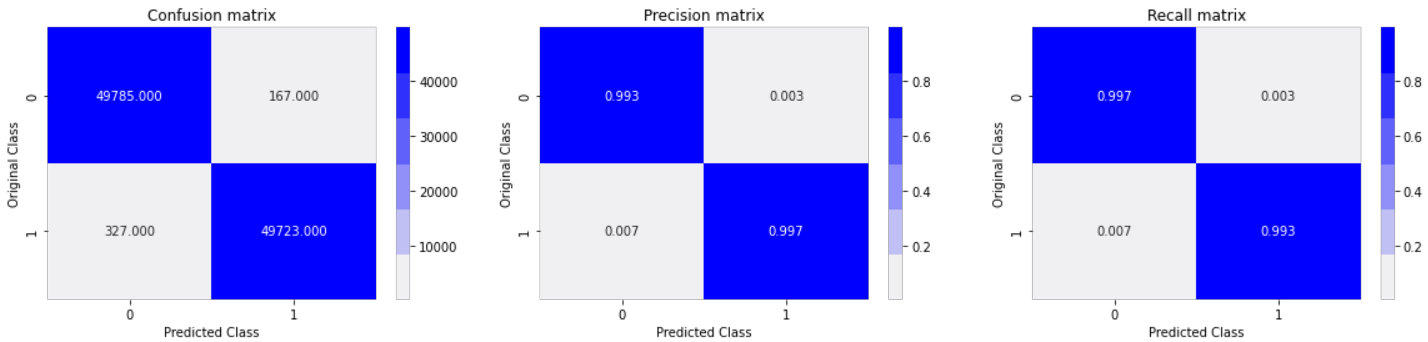
```python
from sklearn.metrics import confusion_matrix
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)

    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=lab
els)
    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=lab
els)
    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=lab
els)
    plt.xlabel('Predicted Class')
```

```
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()
```

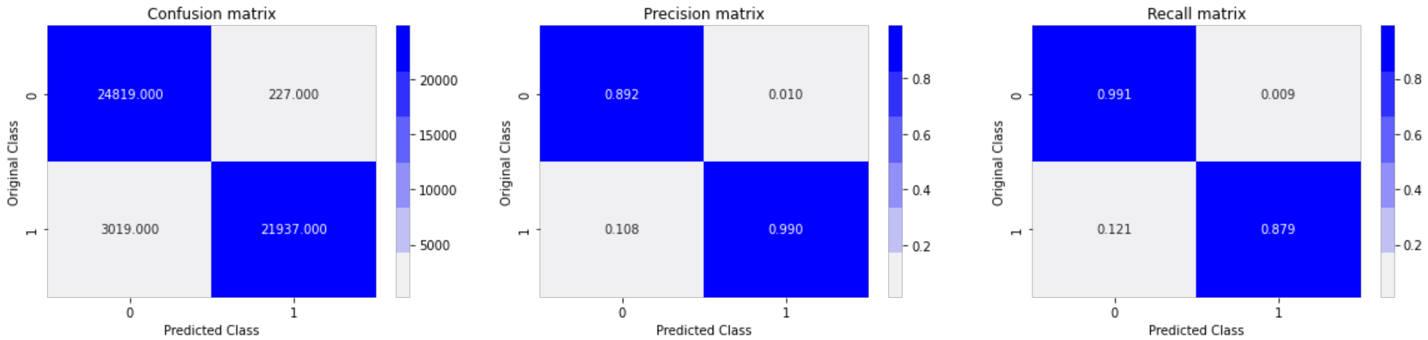
In [ ]:

```
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
```

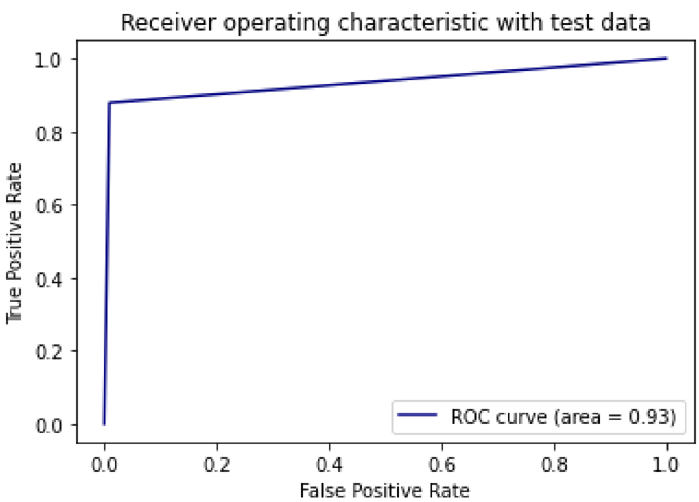



```
Test confusion_matrix
```



In [ ]:

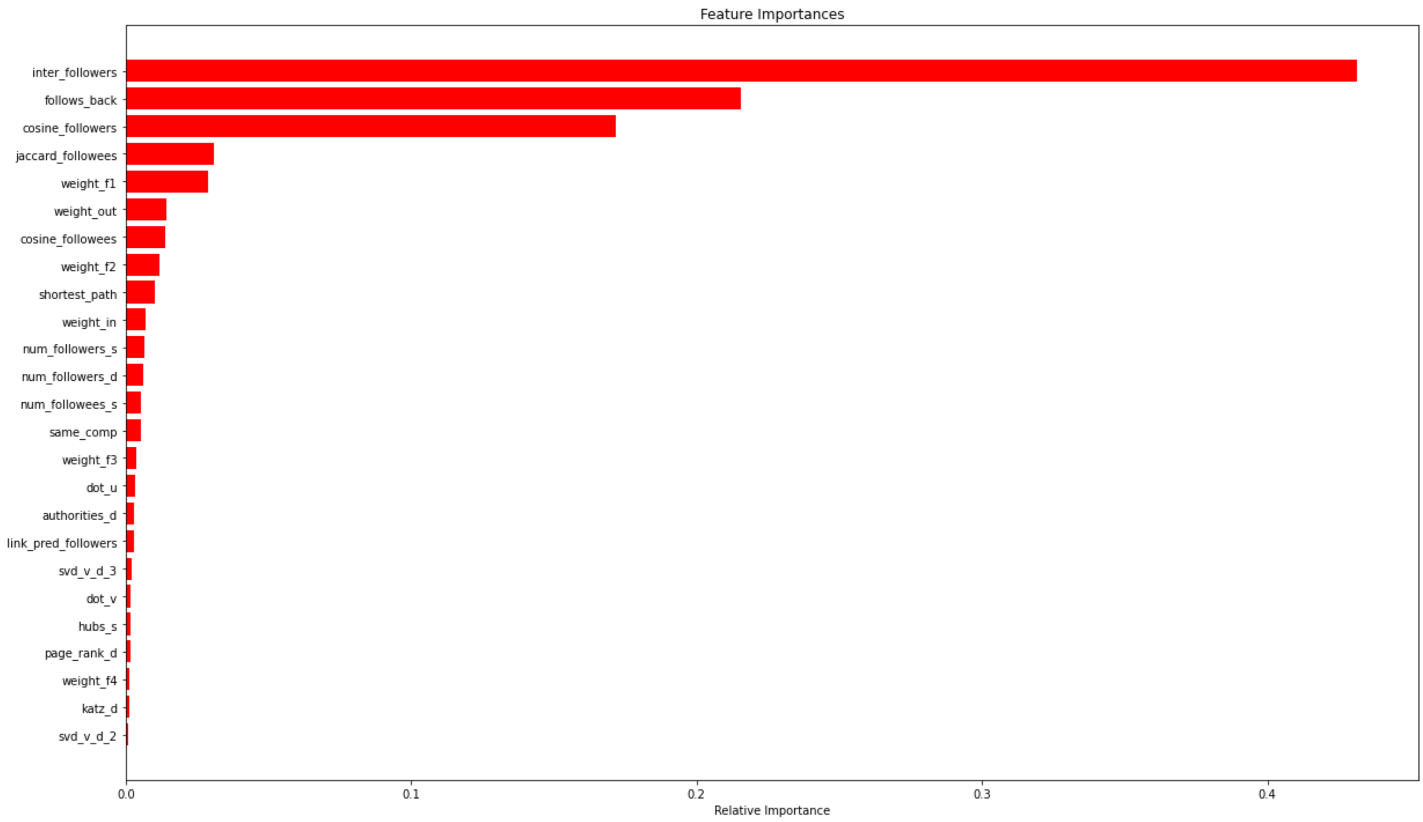
```
from sklearn.metrics import roc_curve, auc
fpr,tpr,ths = roc_curve(y_test,y_test_pred)
auc_sc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='navy',label='ROC curve (area = %0.2f)' % auc_sc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic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



In [ ]:

```
features = df_final_train.columns
```

```
features = df_final_train.columns
importances = clf1.feature_importances_
indices = (np.argsort(importances))[-25:]
plt.figure(figsize=(20,12))
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 [ ]:

```
from prettytable import PrettyTable
t = PrettyTable()
t.field_names=["model","train_f1_Score","test_f1_score"]
t.add_row(["XGBoost","0.995","0.931"])
print(t)
```

```
+---------+----------------+----------------+
|  model  | train_f1_Score | test_f1_score |
+---------+----------------+----------------+
| XGBoost |     0.995      |     0.931      |
+---------+----------------+----------------+
```

Observations:

1. **When actual prediction takes place in the xgboost classifier, the f1 score for test is 0.931.**
2. **The f1 score for train is 0.995.**
3. **Some of the Best features that were extracted are:**

- **inter_followers**
- **follows_back**
- **cosine_followers**
- **jaccard_followees**

1. **Area under ROC curve is coming out to be 0.93**