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
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV

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 [ ]:

```
#Getting after_eda data
!wget --header="Host: doc-0s-80-drive-data-export.googleusercontent.com" --header="Use
```

```
--2020-09-02 06:55:47--  https://doc-0s-80-drive-data-export.googleuserconte
nt.com/download/sghr8pnsjffiha0mq8imgl11o723ufn6/dk0plg57034lp2684omnrvf1ube
bje3k/1599025500000/51fbc4b6-29b4-4fdb-b0b9-717a0a8c3a63/100968300222125116 5
81/ADt3v-NWmVn6oYzxH1_stsqDLgfb8XWEyQ8q9lRpqBnNhva--067JxM8CHaiBvcHZx71smjTS
a4ItbtEmCkmFL3btGAcTiYDR8r2TSEHI2YVKN4XEWVeciepyGzQWRN_YYkwtMQiE2rm2dQqez1GY
TsYk4OShWVjtyVeNWKHmmNRBlbgbZv_9a33ayKvsevQeehaFuP3swBS7vtKjUohOFq1Y7_C0kY1_
C9TAebJpHUjFpsYLAjqwYY6UEzGxPn51HI3oS04vFvMRwQQ1SHG5GfI_0tGDYRk8EbMh8HZYUyQ1
9irmTZ-BvjRVzfQBwcawQnKfvdfWQrd?authuser=0&nonce=b34jo1tr99bjq&user=10096830
0222125116581&hash=4mvd5ln84an7rk81gn71nl3fp7biloin (https://doc-0s-80-drive
-data-export.googleusercontent.com/download/sghr8pnsjffiha0mq8imgl11o723ufn
6/dk0plg57034lp2684omnrvf1ubebje3k/1599025500000/51fbc4b6-29b4-4fdb-b0b9-717
a0a8c3a63/10096830022212511658 1/ADt3v-NWmVn6oYzxH1_stsqDLgfb8XWEyQ8q9lRpqBnN
hva--067JxM8CHaiBvcHZx71smjTSa4ItbtEmCkmFL3btGAcTiYDR8r2TSEHI2YVKN4XEWVeciep
yGzQWRN_YYkwtMQiE2rm2dQqez1GYTsYk4OShWVjtyVeNWKHmmNRBlbgbZv_9a33ayKvsevQeeha
FuP3swBS7vtKjUohOFq1Y7_C0kY1_C9TAebJpHUjFpsYLAjqwYY6UEzGxPn51HI3oS04vFvMRwQQ
1SHG5GfI_0tGDYRk8EbMh8HZYUyQ19irmTZ-BvjRVzfQBwcawQnKfvdfWQrd?authuser=0&nonc
e=b34jo1tr99bjq&user=10096830022212511658 1&hash=4mvd5ln84an7rk81gn71nl3fp7bi
loin)
Resolving doc-0s-80-drive-data-export.googleusercontent.com (doc-0s-80-drive
-data-export.googleusercontent.com)... 172.217.203.132, 2607:f8b0:400c:c07::
84
Connecting to doc-0s-80-drive-data-export.googleusercontent.com (doc-0s-80-d
rive-data-export.googleusercontent.com)|172.217.203.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 416199167 (397M) [application/octet-stream]
Saving to: ‘drive-download-20200902T064408Z-001.zip’

drive-download-2020 100%[===================>] 396.92M   125MB/s    in 3.2s

2020-09-02 06:55:52 (125 MB/s) - ‘drive-download-20200902T064408Z-001.zip’ s
aved [416199167/416199167]
```

In [ ]:

```
!unzip 'drive-download-20200902T064408Z-001.zip'
```

```
Archive:  drive-download-20200902T064408Z-001.zip
  inflating: test_neg_after_eda.csv
  inflating: test_pos_after_eda.csv
  inflating: train_pos_after_eda.csv
  inflating: train_woheader.csv
  inflating: train_neg_after_eda.csv
  inflating: missing_edges_final.p
  inflating: train_after_eda.csv
  inflating: test_after_eda.csv
```

In [ ]:

```
#Getting fea_sample data
!wget --header="Host: doc-3s-20-drive-data-export.googleusercontent.com" --header="Use
```

```
--2020-09-02 06:42:49--  https://doc-3s-20-drive-data-export.googleuserconte
nt.com/download/sghr8pnsjffiha0mq8imgl11o723ufn6/hrl7r51t9k3qdbe5h7gs8h09gp6
lep10/1599025500000/919e3811-e419-4176-824e-e545b09a3b53/100968300222125116 5
81/ADt3v-Pw9RoFjhn6TfXjSqSOQDZ0-j1Wz0g4RbiO-PNfsajPiPhuQ7g8apoM9UEVsVfuyW507
azeiJ_QflAP6OlBIqY-pHLL5vI2ato01eoBysWGUpFMbR0xtityAiPJ-rYDEmNDdDW0QlA4UPKw3
D_Z9WVDuYFmsuDObOYy7r-BjZu9QdWTNtrOLNFoXL04yC8cufxzkeg98RB0W8pNvweOhtKFNkW-T
14veyrN0v9fPnkXA0KhG2v6EOHI99qrPS054IlJcvOZZ4Z3gGUk1nmjtuNVp14p9bmlOrBDV5X6O
PzCG6bHFnOFE_3WqTQeRfH3eKNwaJ2h?authuser=0&nonce=p0gl2as7auk2e&user=10096830
0222125116581&hash=n4spur6l1sn86ghmi6tc96a8mc811h60 (https://doc-3s-20-drive
-data-export.googleusercontent.com/download/sghr8pnsjffiha0mq8imgl11o723ufn
6/hrl7r51t9k3qdbe5h7gs8h09gp6lep10/1599025500000/919e3811-e419-4176-824e-e54
5b09a3b53/10096830022212511658 1/ADt3v-Pw9RoFjhn6TfXjSqSOQDZ0-j1Wz0g4RbiO-PNf
sajPiPhuQ7g8apoM9UEVsVfuyW507azeiJ_QflAP6OlBIqY-pHLL5vI2ato01eoBysWGUpFMbR0x
tityAiPJ-rYDEmNDdDW0QlA4UPKw3D_Z9WVDuYFmsuDObOYy7r-BjZu9QdWTNtrOLNFoXL04yC8c
ufxzkeg98RB0W8pNvweOhtKFNkW-T14veyrN0v9fPnkXA0KhG2v6EOHI99qrPS054IlJcvOZZ4Z3
gGUk1nmjtuNVp14p9bmlOrBDV5X6OPzCG6bHFnOFE_3WqTQeRfH3eKNwaJ2h?authuser=0&nonc
e=p0gl2as7auk2e&user=10096830022212511658 1&hash=n4spur6l1sn86ghmi6tc96a8mc81
1h60)
Resolving doc-3s-20-drive-data-export.googleusercontent.com (doc-3s-20-drive
-data-export.googleusercontent.com)... 172.217.203.132, 2607:f8b0:400c:c07::
84
Connecting to doc-3s-20-drive-data-export.googleusercontent.com (doc-3s-20-d
rive-data-export.googleusercontent.com)|172.217.203.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 178254609 (170M) [application/octet-stream]
Saving to: ‘drive-download-20200902T063450Z-001.zip’

drive-download-2020 100%[===================>] 170.00M  90.6MB/s    in 1.9s

2020-09-02 06:42:51 (90.6 MB/s) - ‘drive-download-20200902T063450Z-001.zip’
 saved [178254609/178254609]
```

In [ ]:

```
!unzip 'drive-download-20200902T063450Z-001.zip'
```

```
Archive:  drive-download-20200902T063450Z-001.zip
  inflating: storage_sample_stage2.h5
  inflating: storage_sample_stage1.h5
  inflating: storage_sample_stage3.h5
  inflating: hits.p
  inflating: storage_sample_stage4.h5
  inflating: katz.p
  inflating: page_rank.p
```

In [ ]:

```
import pandas as pd
train_pos = pd.read_csv('train_pos_after_eda.csv', header=None)
train_pos.head()
```

Out[10]:

	0	1
0	273084	1505602
1	912810	1678443
2	365429	1523458
3	527014	1605979
4	1228116	471233

In [ ]:

```
len(train_pos)
```

Out[11]:

```
7550015
```

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 [ ]:

```
print(len(df_final_train))
len(df_final_test)
```

```
100002
```

Out[15]:

```
50002
```

In [ ]:

```
df_final_train.head()
```

Out[16]:

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

In [ ]:

```
#Creating Graph from data_new
train_graph=nx.read_edgelist('train_pos_after_eda.csv',delimiter=',',create_using=nx.G
print(nx.info(train_graph))
```

```
Name:
Type: Graph
Number of nodes: 1780722
Number of edges: 5457004
Average degree:   6.1290
```

###Preferential Attachment

In [ ]:

```
#Preferential Attachment Function
def preferential_attachment_(a,b):
  if train_graph.has_node(a) and train_graph.has_node(b):
    preds = nx.preferential_attachment(train_graph, [(a,b)])
  else:
    preds = [(a,b,-1)]
  for u, v, p in preds:
     sc = p
 #   print('(%d, %d) -> %d' % (u, v, p))
  return sc
```

In [ ]:

```
preferential_attachment_(273084,15)  #Node not present in Graph
```

Out[30]:

```
-1
```

In [ ]:

```
df_final_train['preferential_attachment'] = df_final_train.apply(lambda row:
                                            preferential_attachment_(row['source_node'
df_final_test['preferential_attachment'] = df_final_test.apply(lambda row:
                                            preferential_attachment_(row['source_node'
```

###Svd Dot between svd's of Source and Dest.

In [ ]:

```
#SVD_DOT Function
def svd_dot_(df_svd):
  svd_dot_u = 0
  pt = 0
  for i in range(6):
    u_s = df_svd[pt]  #Values of svd_u_s
    u_d = df_svd[pt+6] #Values of svd_u_d
    svd_dot_u = svd_dot_u + (u_s+u_d)  #Elementwise Mul and Sum : (svd_u_s_i*svd_u_d_i
    pt = pt+1
  #print(svd_dot_u)
  return svd_dot_u
```

In [ ]:

```
df_final_train['svd_dot_u'] = df_final_train.apply(lambda row:
                                            svd_dot_(row[['svd_u_s_1','svd_u_s_2','svd
                                                          'svd_u_d_1','svd_u_d_2','svd
```

In [ ]:

```
df_final_train['svd_dot_v'] = df_final_train.apply(lambda row:
                                            svd_dot_(row[['svd_v_s_1','svd_v_s_2','svd
                                                          'svd_v_d_1','svd_v_d_2','svd
```

In [ ]:

```
df_final_train.head()
```

Out[68]:

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

In [ ]:

```
df_final_test['svd_dot_u'] = tqdm(df_final_test.apply(lambda row:
                                                      svd_dot_(row[['svd_u_s_1','svd_u_s_2','svd
                                                                    'svd_u_d_1','svd_u_d_2','svd
df_final_test['svd_dot_v'] = tqdm(df_final_test.apply(lambda row:
                                                      svd_dot_(row[['svd_v_s_1','svd_v_s_2','svd
                                                                    'svd_v_d_1','svd_v_d_2','svd
```

```
100%|██████████| 50002/50002 [00:00<00:00, 2101291.38it/s]
100%|██████████| 50002/50002 [00:00<00:00, 2191308.77it/s]
```

In [ ]:

```
df_final_test.head()
```

Out[70]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_fo
0	848424	784690	1	0	0.0	0
1	483294	1255532	1	0	0.0	0
2	626190	1729265	1	0	0.0	0
3	947219	425228	1	0	0.0	0
4	991374	975044	1	0	0.2	0

###Implementation for XGBoost Starts Here

In [ ]:

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

In [ ]:

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

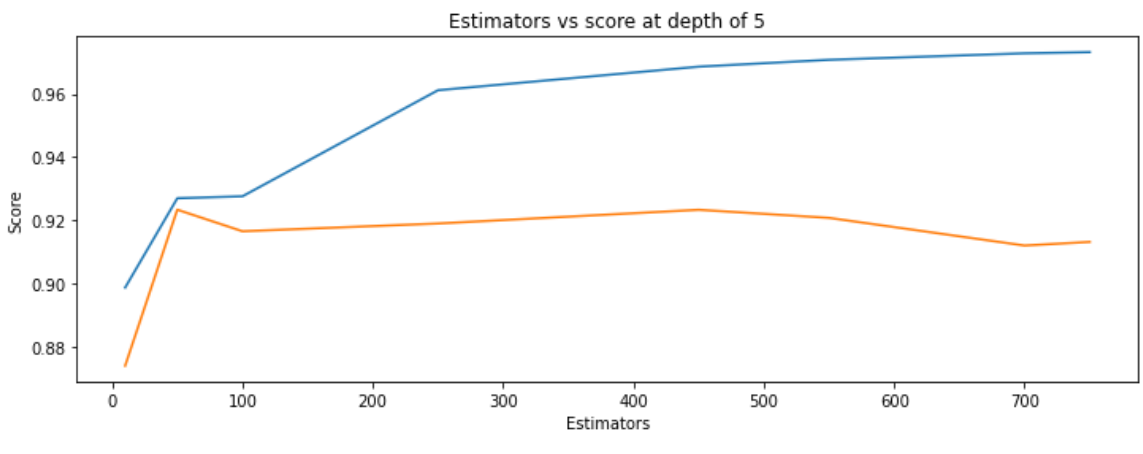
In [ ]:

```
#Implementing XGBoost Here
estimators = [10,50,100,250,450,550,700,750]
train_scores = []
test_scores = []
for i in estimators:
    clf = XGBClassifier(silent=False,
                        scale_pos_weight=1,
                        learning_rate=0.01,
                        colsample_bytree = 0.4,
                        subsample = 0.8,
                        objective='binary:logistic',
                        n_estimators=i,
                        reg_alpha = 0.3,
                        max_depth=4,
                        gamma=10)
    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 0.8987468187236235 test Score 0.874023694602896
Estimators =  50 Train Score 0.9269906841461872 test Score 0.923373351119016
Estimators =  100 Train Score 0.9276093214818552 test Score 0.91654627347667
22
Estimators =  250 Train Score 0.961156351791531 test Score 0.919004831736882
3
Estimators =  450 Train Score 0.9686075025839531 test Score 0.92333778427756
94
Estimators =  550 Train Score 0.9707796479792714 test Score 0.92078217758580
34
Estimators =  700 Train Score 0.9728346098200104 test Score 0.91203574420552
91
Estimators =  750 Train Score 0.973202720207254 test Score 0.913176930176651
1
```

Out[90]:

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


Estimators vs score at depth of 5
Score
0.96
0.94
0.92
0.90
0.88
0
100
200
300
400
500
600
700
Estimators

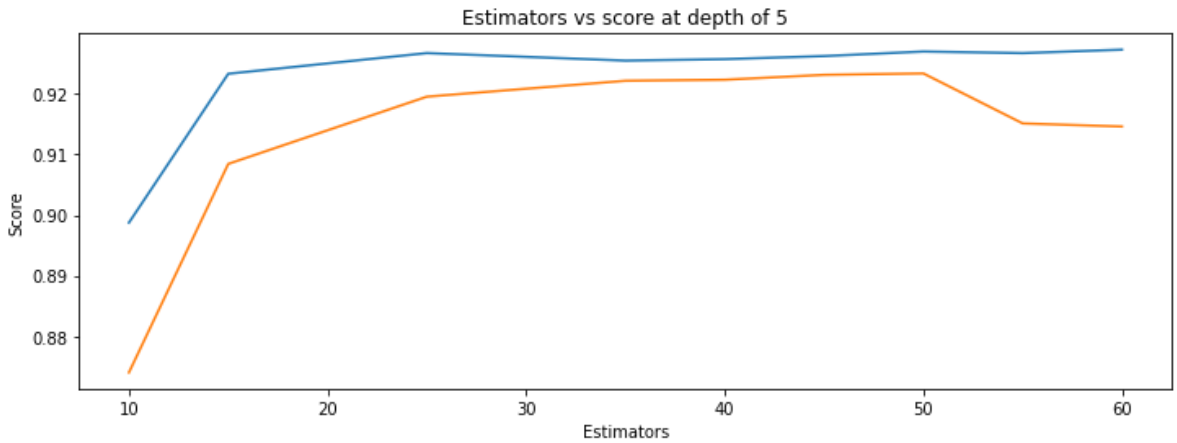
In [ ]:

```
estimators = [10,15,25,35,40,45,50,55,60]
train_scores = []
test_scores = []
for i in estimators:
    clf = XGBClassifier(silent=False,
                        scale_pos_weight=1,
                        learning_rate=0.01,
                        colsample_bytree = 0.4,
                        subsample = 0.8,
                        objective='binary:logistic',
                        n_estimators=i,
                        reg_alpha = 0.3,
                        max_depth=4,
                        gamma=10)
    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 0.8987468187236235 test Score 0.874023694602896
Estimators =  15 Train Score 0.9233213057153978 test Score 0.908443040945654
8
Estimators =  25 Train Score 0.9267309174205398 test Score 0.919538271499765
7
Estimators =  35 Train Score 0.9254898653990071 test Score 0.922161269141973
6
Estimators =  40 Train Score 0.925730673527055 test Score 0.922333799415279
Estimators =  45 Train Score 0.9262387565509896 test Score 0.923142142524057
9
Estimators =  50 Train Score 0.9269906841461872 test Score 0.923373351119016
Estimators =  55 Train Score 0.9267256786022161 test Score 0.915124441608168
5
Estimators =  60 Train Score 0.9272975814931651 test Score 0.914611813806907
7
```

Out[91]:

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



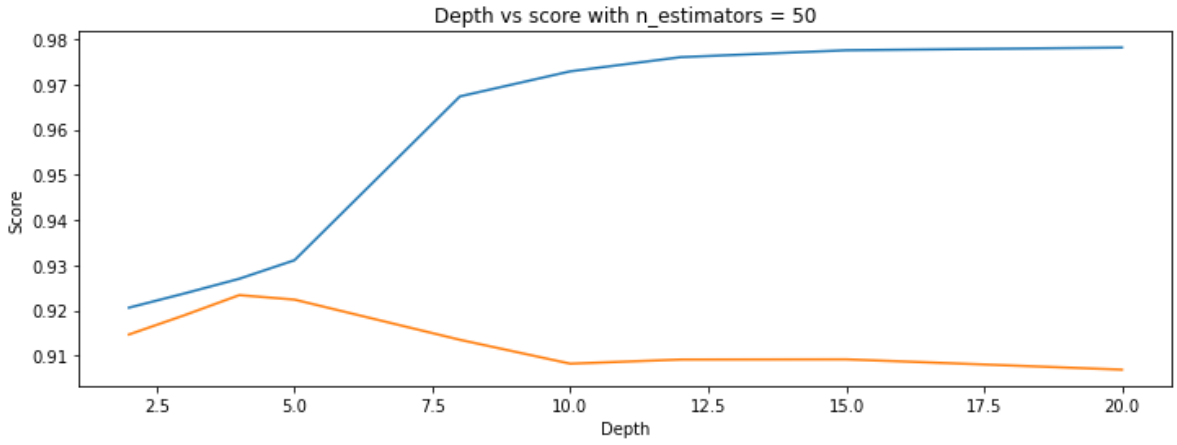
In [ ]:

```
depths = [2,3,4,5,8,10,12,15,20]
train_scores = []
test_scores = []
for i in depths:
    clf = XGBClassifier(silent=False,
                        scale_pos_weight=1,
                        learning_rate=0.01,
                        colsample_bytree = 0.4,
                        subsample = 0.8,
                        objective='binary:logistic',
                        n_estimators=50,
                        reg_alpha = 0.3,
                        max_depth=i,
                        gamma=10)
    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 with n_estimators = 50')
```

```
Depth =  2 Train Score 0.9205948018029403 test Score 0.9146623153701074
Depth =  3 Train Score 0.9237252687497386 test Score 0.9188973400008404
Depth =  4 Train Score 0.9269906841461872 test Score 0.923373351119016
Depth =  5 Train Score 0.9310883818515604 test Score 0.9223829390711366
Depth =  8 Train Score 0.9673295167196002 test Score 0.9134871434353763
Depth =  10 Train Score 0.972870167246966 test Score 0.908207805546046
Depth =  12 Train Score 0.976004016064257 test Score 0.9091258733787739
Depth =  15 Train Score 0.9775280898876405 test Score 0.9091686623332192
Depth =  20 Train Score 0.9781313323836848 test Score 0.9068895317689918
```

Out[94]:

```
Text(0.5, 1.0, 'Depth vs score with n_estimators = 50')
```



In [ ]:

```
from scipy.stats import randint as sp_randint
from scipy.stats import uniform
```

In [ ]:

```
param_dist = {"n_estimators":[40,45,50,55,60],
              "max_depth": [3,4,5,6,7],
              "learning_rate": [0.01,0.05,0.1,0.15,0.2],
              "subsample": [0.75,0.8,0.85,0.9,0.95],
              "gamma": [0,1,5,8,10],
              "colsample_bytree": [0.4,0.6,0.8,0.9,1],
              "reg_alpha": [0.2,0.3,0.4,0.5,0.6]}

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

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

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

```
mean test scores [0.9741902  0.92802887 0.96636358 0.97428761 0.9773855 ]
mean train scores [0.9749252  0.92821009 0.96688544 0.97487048 0.98057475]
```

In [ ]:

```
print(xgb_random.best_estimator_)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.9, gamma=8,
              learning_rate=0.2, max_delta_step=0, max_depth=7,
              min_child_weight=1, missing=None, n_estimators=50, n_jobs=-1,
              nthread=None, objective='binary:logistic', random_state=25,
              reg_alpha=0.4, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=0.95, verbosity=1)
```

In [ ]:

```
clf = XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.9, gamma=8,
              learning_rate=0.2, max_delta_step=0, max_depth=7,
              min_child_weight=1, missing=None, n_estimators=50, n_jobs=-1,
              nthread=None, objective='binary:logistic', random_state=25,
              reg_alpha=0.4, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=0.95, verbosity=1)
```

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 0.980836113995383
Test f1 score 0.9314090102273688
```

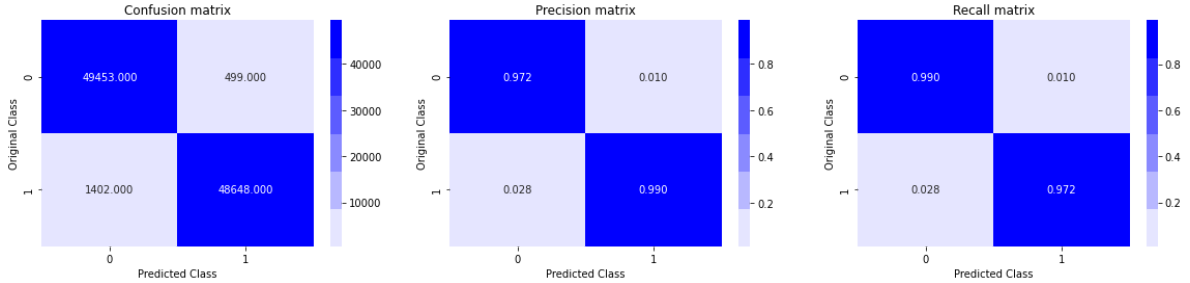
In [ ]:

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

    plt.show()
```

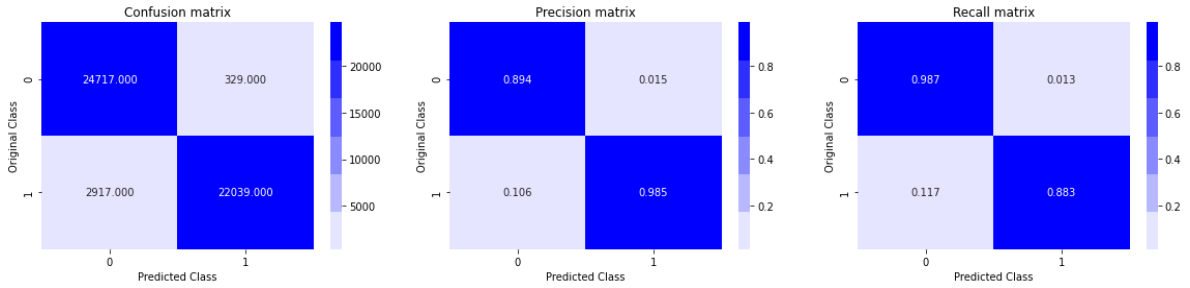
In [ ]:

```
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
```

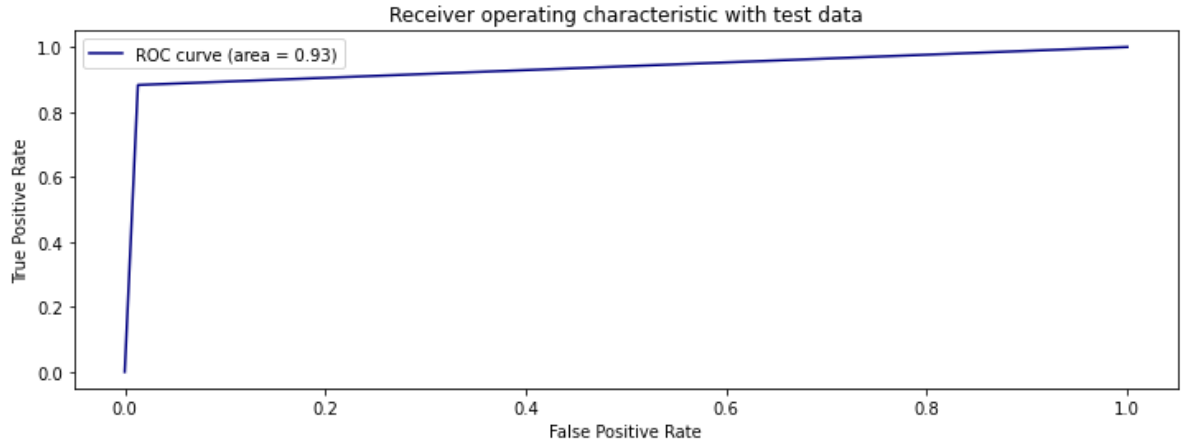



```
Test confusion_matrix
```



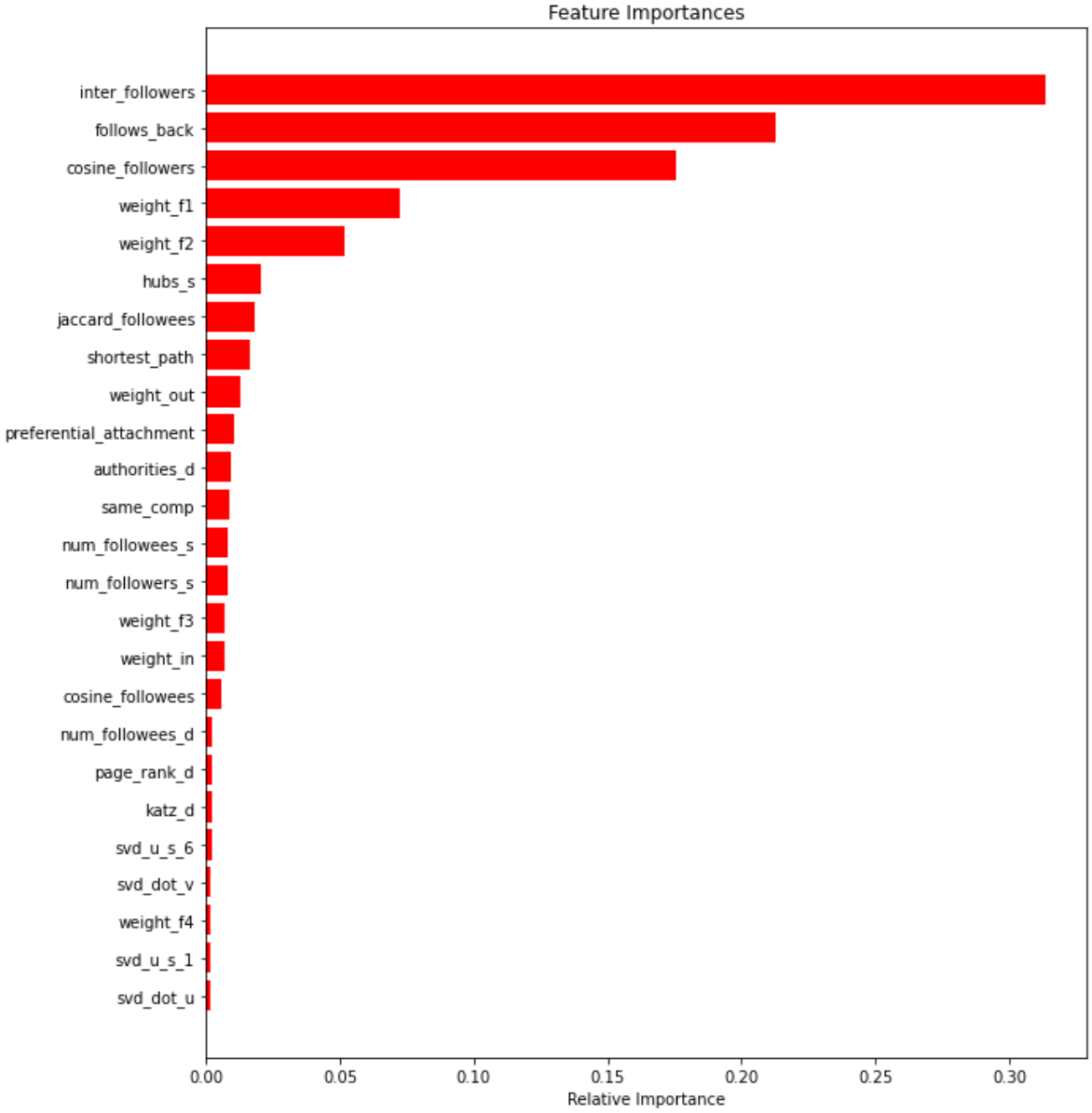
In [ ]:

```
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()
```



####1. For Detailed Explanation of Steps upto addition of Preferential Attachment as a Feature, Please refer to the "**Facebook Case Study - Notes**" pdf attached with this assignment submission

#####2. Preferential Attachment:

######2.1. One well-known concept in social networks is that users with many friends tend to create more connections in the future. This is due to the fact that in some social networks, like in finance, the rich get richer. We estimate how ”rich” our two vertices are by calculating the multiplication between the number of friends (|Γ(x)|) or followers each vertex has. It may be noted that the similarity index does not require any node neighbor information; therefore, this similarity index has the lowest computational complexity.

######3. SVD Dot Feature: 3.1. svd_dot is a Dot product between sourse node svd and destination node svd features.

######4. Apply XGBoostClassifier on all these Features to predict whether link is present or not

######5. Hyperparameter Tune the parameters of XGBoost Classifier to get the best posible metric value for this dataset and set of features

#####**6. Results:**

#####**Train f1 score 0.980836113995383**

#####**Test f1 score 0.9314090102273688**