Social network Graph Link Prediction - Facebook Challenge

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

```
#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
```

Section 1

```
#reading
from pandas import read_hdf
df_final_train = read_hdf('/content/drive/MyDrive/AI-ML-Assignments/Facebook Friend Recomm
df_final_test = read_hdf('/content/drive/MyDrive/AI-ML-Assignments/Facebook Friend Recomme
df_final_train.columns
```

```
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')
```

```
df_final_test.columns
```

```
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')
```

```
df_final_train.indicator_link.value_counts()
```

```
1    50050
0    49952
Name: indicator_link, dtype: int64
```

```
df_final_test.indicator_link.value_counts()
```

```
0    25046
1    24956
Name: indicator_link, dtype: int64
```

```
df_final_train.head()
```

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followe
0	273084	1505602	1	0	0.0000
1	832016	1543415	1	0	0.1871
2	1325247	760242	1	0	0.3695

```
df_final_test.head()
```

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followe
0	848424	784690	1	0	(
1	483294	1255532	1	0	(
2	626190	1729265	1	0	(
3	947219	425228	1	0	(
4	991374	975044	1	0	(

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

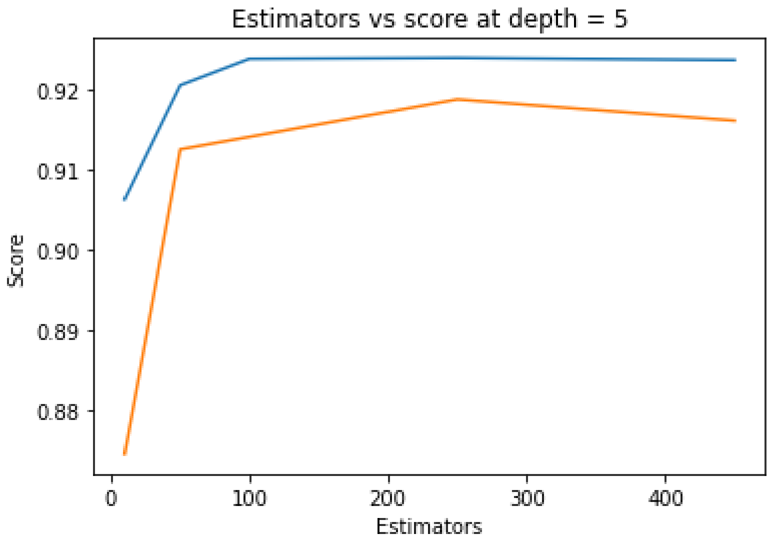
```
df_final_train.drop(['source_node', 'destination_node','indicator_link'],axis=1,inplace=Tr
df_final_test.drop(['source_node', 'destination_node','indicator_link'],axis=1,inplace=Tru
```

```
df_final_test.head()
```

	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	num_fol
0	0	0.0	0.029161	0.000000	

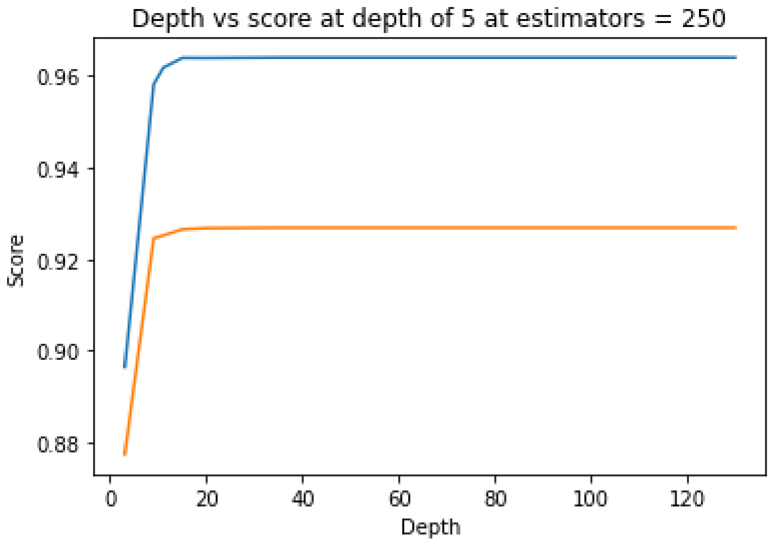
```
estimators = [10,50,100,250,450]
train_scores = []
test_scores = []
for i in estimators:
    clf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=5, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0,
            min_samples_leaf=52, min_samples_split=120,
            min_weight_fraction_leaf=0.0, n_estimators=i, n_jobs=-1,random_state=25,verbos
    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 = 5')
```

```
Estimators =  10 Train Score 0.9063252121775113 test Score 0.8745605278006858
Estimators =  50 Train Score 0.9205725512208812 test Score 0.9125653355634538
Estimators =  100 Train Score 0.9238690848446947 test Score 0.9141199714153599
Estimators =  250 Train Score 0.9239789348046863 test Score 0.9188007232664732
Estimators =  450 Train Score 0.9237190618658074 test Score 0.9161507685828595
Text(0.5, 1.0, 'Estimators vs score at depth = 5')
```



```
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=250, n_jobs=-1,random_state=25,verb
    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 = 250')
plt.show()
```

```
depth =  3 Train Score 0.8964289081404971 test Score 0.8774484755578266
depth =  9 Train Score 0.9581477820464481 test Score 0.9245867115931348
depth =  11 Train Score 0.9617793659333854 test Score 0.9251935375294513
depth =  15 Train Score 0.9639261063743403 test Score 0.9265346492981754
depth =  20 Train Score 0.9638992101683215 test Score 0.926801541020189
depth =  35 Train Score 0.964 test Score 0.926885749773689
depth =  50 Train Score 0.964 test Score 0.926885749773689
depth =  70 Train Score 0.964 test Score 0.926885749773689
depth =  130 Train Score 0.964 test Score 0.926885749773689
```



```
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
```

```

param_dist = {"n_estimators":sp_randint(200,215),
              "max_depth": sp_randint(15,20),
              "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_trai

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 [0.9630276  0.96358452 0.96250125 0.96275968 0.96432123]
mean train scores [0.9635756  0.96473346 0.96316174 0.96347044 0.96559538]
```

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

```
RandomForestClassifier(max_depth=19, min_samples_leaf=28, min_samples_split=111,
                       n_estimators=206, n_jobs=-1, random_state=25)
```

```
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_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)
```

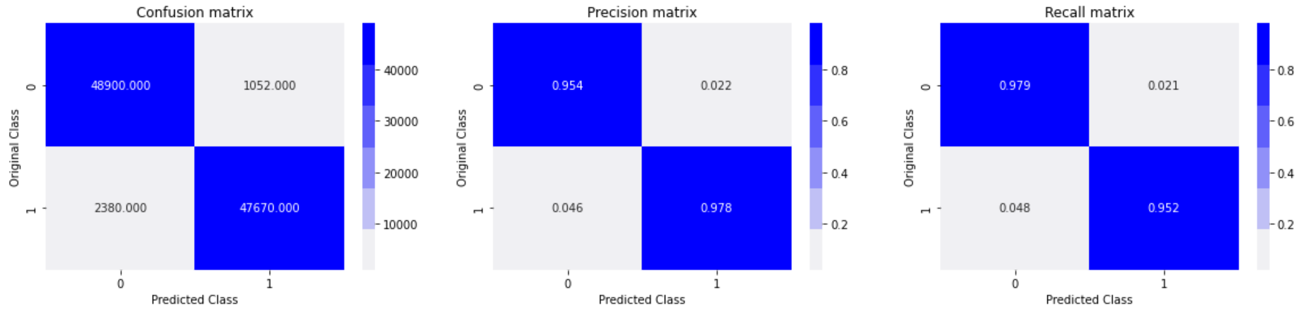
```
from sklearn.metrics import f1_score
print('Train f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))
```

```
Train f1 score 0.9652533106548414
Test f1 score 0.9241678239279553
```

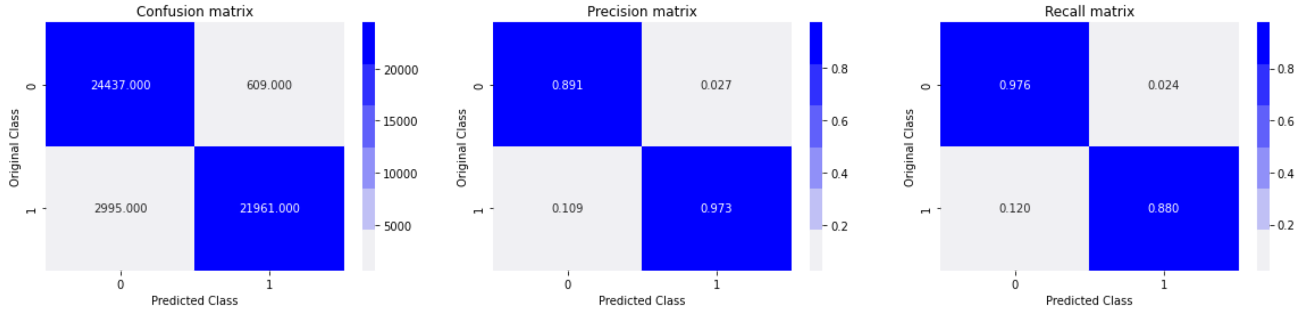
```
from sklearn.metrics import confusion_matrix
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)

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

    plt.show()
```

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

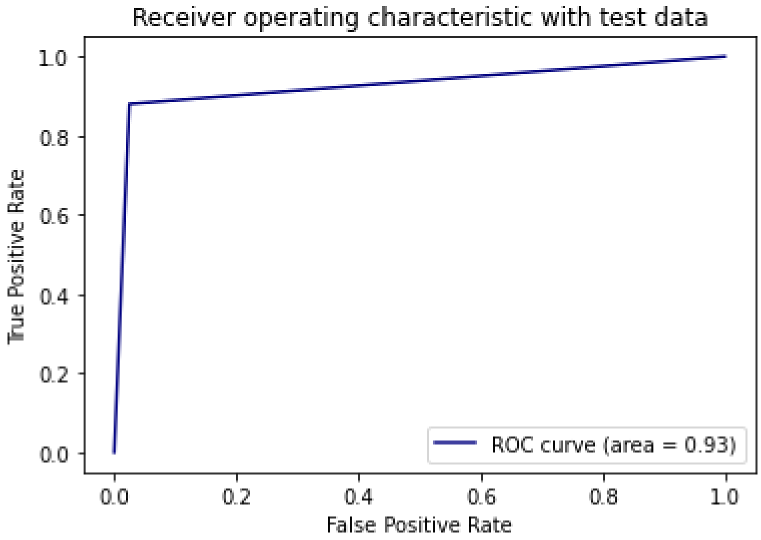
```
Train confusion_matrix
```



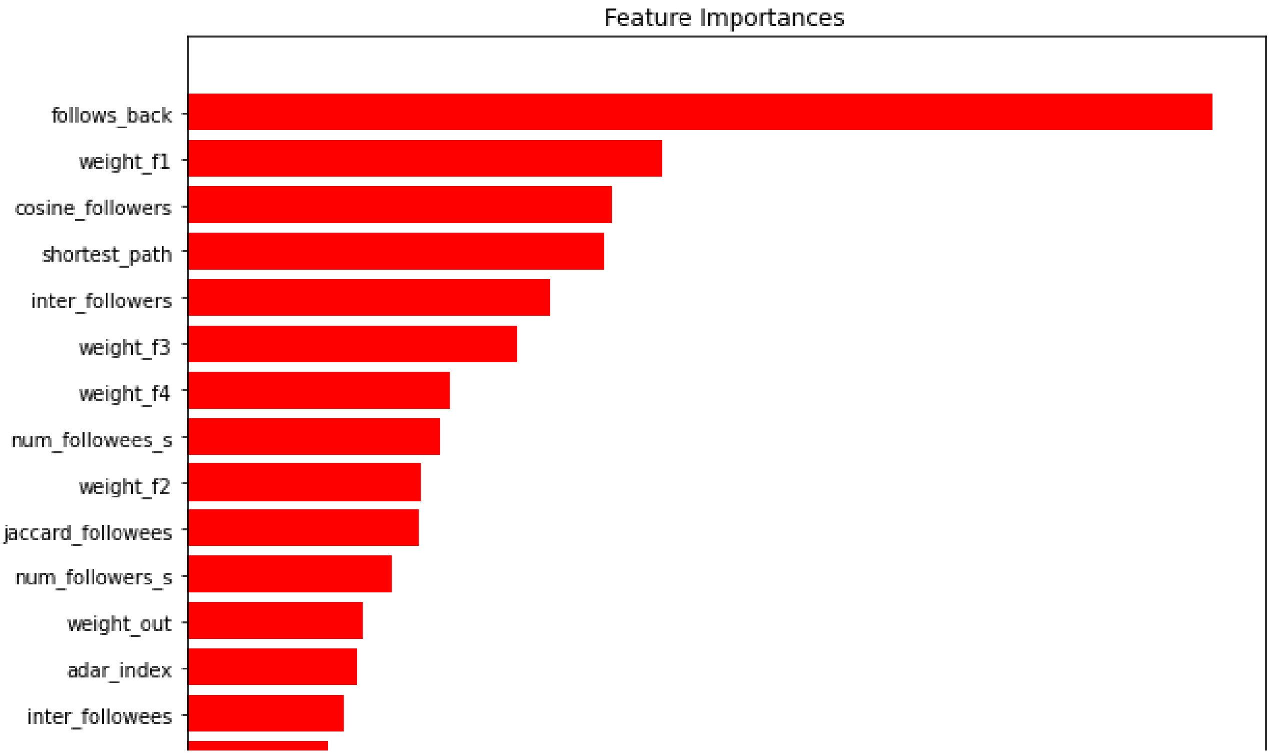
```
Test confusion_matrix
```



```
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()
```



```
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()
```



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.

```
#reading
from pandas import read_hdf
df_final_train = read_hdf('/content/drive/MyDrive/AI-ML-Assignments/Facebook Friend Recomm
df_final_test = read_hdf('/content/drive/MyDrive/AI-ML-Assignments/Facebook Friend Recomme
```

```
graph=nx.read_edgelist('/content/drive/MyDrive/AI-ML-Assignments/Facebook Friend Recommend
```

```
print(nx.info(graph))
```

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

```
def preferential_attachment_followers(a,b):
  try:
    return len(set(graph.successors(a)))*len(set(graph.successors(b)))
  except:
    return 0
```

```
def preferential_attachment_followee(a,b):
  try:
    return len(set(graph.predecessors(a)))*len(set(graph.predecessors(b)))
  except:
    return 0
```

```
l=list(graph.nodes())
print(l[0:10])
del l
```

```
[273084, 1505602, 912810, 1678443, 365429, 1523458, 527014, 1605979, 1228116, 471233]
```

```
l = list(sorted(graph.nodes()))
print(l[0:10])
del l
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 11]
```

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

```
#some of the nodes were to be missing from graph. so creating a dict where for every node
sadj_col = sorted(graph.nodes())
sadj_dict = { val:idx for idx,val in enumerate(sadj_col)}
```

```
def svd_dot(a,b,M):
  try:
    z1 = sadj_dict[a]
    z2 = sadj_dict[b]
    return np.dot(M[z1],M[z2])
  except:
    m1 = [0,0,0,0,0,0]
    m2 = [0,0,0,0,0,0]
    return np.dot(m1,m2)
```

```
#getting id value for the first source and destination node in train df
print(sadj_dict[273084])
print(sadj_dict[1505602])
```

```
261109
1439822
```

```
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,)
```

```
del Adj
```

```
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', 'svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3',
       'svd_u_d_4', 'svd_u_d_5', 'svd_u_d_6']].head()
```

	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
0	-1.666226e-13	4.613397e-13	1.043044e-05	6.676960e-13	2.451081e-13	3.584580e-12	-2.038017e-11
1	7.051088e-13	-8.250564e-11	-1.717702e-10	3.705016e-02	1.032392e-11	7.207497e-10	1.644399e-12
2	-4.900734e-18	1.096831e-18	-6.816555e-19	-2.226453e-18	6.710556e-19	-8.161336e-19	-2.606312e-18
	-9.965436e-	4.077137e-	5.083778e-	1.985267e-	2.471968e-	1.004354e-	-2.629029e-

```
U[261109]
```

```
array([ 1.66633547e-13, -4.61382017e-13,  1.04304053e-05, -6.67802938e-13,
        2.45110805e-13,  3.58494471e-12])
```

```
U[1439822]
```

```
array([ 2.03801812e-11, -5.39791190e-13,  1.06894289e-06, -1.19246167e-12,
        2.27948749e-12,  3.58120187e-12])
```

```
np.dot(U[261109],U[1439822])
```

```
1.1149507627529884e-11
```

```
(-1.666226e-13 * -2.038017e-11) + (4.613397e-13  * 5.397495e-13) + (1.043044e-05 * 1.06894
 + (6.676960e-13 * 1.192357e-12) + (2.451081e-13 * 2.279485e-12) + (3.584580e-12 * 3.58083
```

```
1.1149577116257837e-11
```

```
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', 'svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3',
       'svd_u_d_4', 'svd_u_d_5', 'svd_u_d_6']].head()
```

	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	sv
0	-9.987979e-12	2.283676e-13	1.439968e-10	6.136162e-12	4.188171e-12	5.983361e-15	-1.026186e-11	5.3

```
#getting id value for the first node in test df
print(sadj_dict[848424])
print(sadj_dict[784690])
```

```
811112
750092
```

	-4.491246e-	9.917404e-	7.891250e-	9.458624e-	2.715848e-	1.822074e-	-1.070733e-	2.6

```
U[811112]
```

```
array([ 9.98797970e-12, -2.28369811e-13,  1.43996002e-10, -6.13617161e-13,
        4.18817510e-13,  5.98346173e-15])
```

```
print(U[750092])
```

```
[ 1.02618639e-11 -5.31664593e-13  5.84319190e-10 -1.16604852e-13
  2.25335635e-11  3.22046896e-15]
```

```
df_final_train.head()
```

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followe
0	273084	1505602	1	0	0.0000
1	832016	1543415	1	0	0.1871
2	1325247	760242	1	0	0.3695
3	1368400	1006992	1	0	0.0000
4	140165	1708748	1	0	0.0000

```
df_final_train['Preferential_attachment_followers'] = df_final_train.apply(lambda row: pre
```

```
df_final_train['Preferential_attachment_followees'] = df_final_train.apply(lambda row: pre
```

```
df_final_test['Preferential_attachment_followers'] = df_final_test.apply(lambda row: prefe
```

```
df_final_test['Preferential_attachment_followees'] = df_final_test.apply(lambda row: prefe
```

```
df_final_train.head()
```

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followe
0	273084	1505602	1	0	0.0000
1	832016	1543415	1	0	0.1871
2	1325247	760242	1	0	0.3695
3	1368400	1006992	1	0	0.0000
4	140165	1708748	1	0	0.0000

```
df_final_test.head()
```

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followe
0	848424	784690	1	0	(
1	483294	1255532	1	0	(
2	626190	1729265	1	0	(
3	947219	425228	1	0	(
4	991374	975044	1	0	(

```
df_final_train['svd_dot_u'] = df_final_train.apply(lambda row: svd_dot(row['source_node'],
df_final_train['svd_dot_v'] = df_final_train.apply(lambda row: svd_dot(row['source_node'],
```

```
df_final_test['svd_dot_u'] = df_final_test.apply(lambda row: svd_dot(row['source_node'],ro
df_final_test['svd_dot_v'] = df_final_test.apply(lambda row: svd_dot(row['source_node'],ro
```

```
df_final_train.head()
```

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followe
0	273084	1505602	1	0	0.0000
1	832016	1543415	1	0	0.1871
4	140165	1708748	1	0	0.0000

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

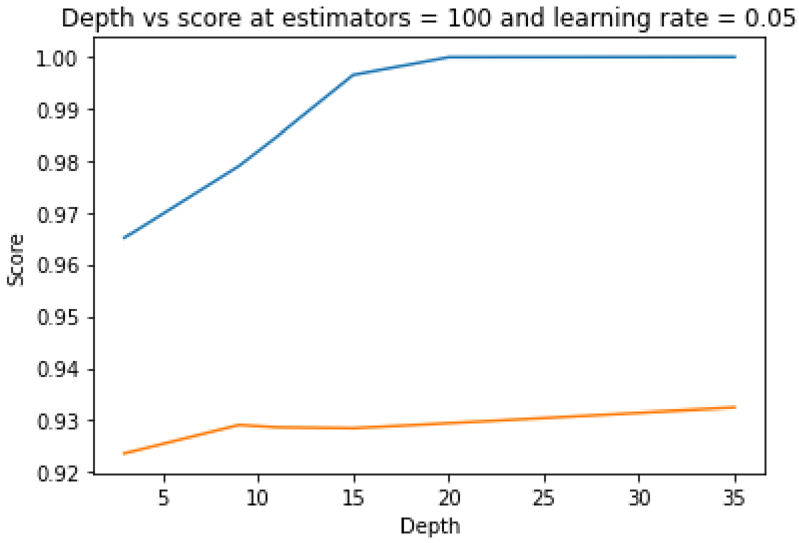
```
df_final_train.drop(['source_node', 'destination_node','indicator_link'],axis=1,inplace=Tr
df_final_test.drop(['source_node', 'destination_node','indicator_link'],axis=1,inplace=Tru
```

```
from xgboost import XGBClassifier
eta = [0.05, 0.10, 0.15, 0.20, 0.25, 0.30]
train_scores = []
test_scores = []
for i in eta:
    clf = XGBClassifier(
            max_depth=5, learning_rate=i, n_estimators=100, n_jobs=-1,random_state=25,verb
    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('Learning rate = ',i,'Train Score',train_sc,'test Score',test_sc)
plt.plot(eta,train_scores,label='Train Score')
plt.plot(eta,test_scores,label='Test Score')
plt.xlabel('Learning rate')
plt.ylabel('Score')
plt.title('Learning rate vs score at depth = 5 and Estimator = 100')
```

```
     Learning rate =  0.05 Train Score 0.972347332116048 test Score 0.9320216179699375
```

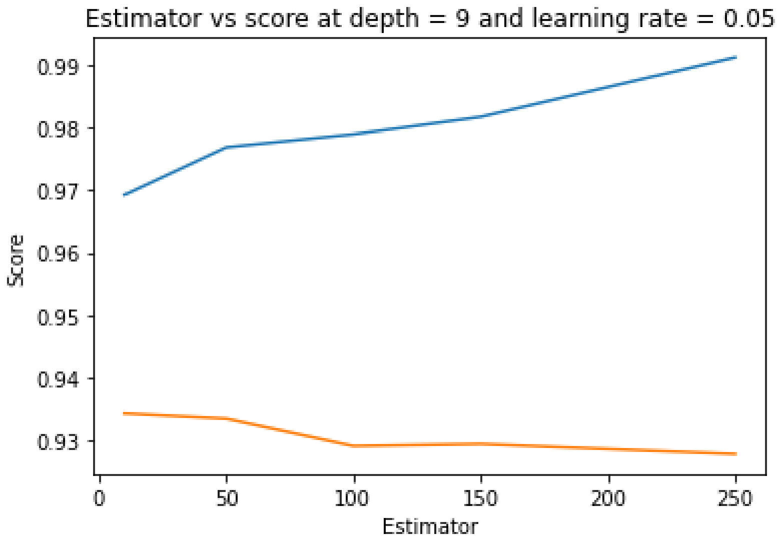
```
from xgboost import XGBClassifier
depths = [3,9,11,15,20,35]
train_scores = []
test_scores = []
for i in depths:
    clf = XGBClassifier(
            max_depth=i, learning_rate=0.05, n_estimators=100, n_jobs=-1,random_state=25,v
    clf.fit(df_final_train,y_train)
    train_sc = f1_score(y_train,clf.predict(df_final_train))
    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 estimators = 100 and learning rate = 0.05')
plt.show()
```

```
     depth =  3 Train Score 0.9651238327243199 test Score 0.9236428209030949
     depth =  9 Train Score 0.9789552419843933 test Score 0.9291214969132529
     depth =  11 Train Score 0.9846051477482383 test Score 0.9286622583926755
     depth =  15 Train Score 0.9964123223698715 test Score 0.9284913127208855
     depth =  20 Train Score 0.9998401694204143 test Score 0.9294535611571647
     depth =  35 Train Score 0.9998801318549595 test Score 0.9324224597802939
```



```
from xgboost import XGBClassifier
estimators = [10,50,100,150,250]
train_scores = []
test_scores = []
for i in estimators:
    clf = XGBClassifier(
            max_depth=9, learning_rate=0.05, n_estimators=i, n_jobs=-1,random_state=25,ver
    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('Estimator')
plt.ylabel('Score')
plt.title('Estimator vs score at depth = 9 and learning rate = 0.05')
plt.show()
```

```
Estimators =  10 Train Score 0.9693152751026731 test Score 0.934330559359528
Estimators =  50 Train Score 0.976886926224527 test Score 0.9334966009373812
Estimators =  100 Train Score 0.9789552419843933 test Score 0.9291214969132529
Estimators =  150 Train Score 0.9817374634244779 test Score 0.9294449271983554
Estimators =  250 Train Score 0.9912308016835253 test Score 0.9278438030560271
```



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

param_dist = {"n_estimators":sp_randint(0,12),
              "max_depth": sp_randint(5,9),
              "gamma": sp_randint(0,10)}

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_trai

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 [0.96339314 0.96524487 0.95210246 0.9424481  0.9467062 ]
mean train scores [0.96420905 0.96690445 0.9527721  0.94282528 0.94797504]
```

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

```
XGBClassifier(gamma=6, max_depth=8, n_estimators=7, n_jobs=-1, random_state=25)
```

```
clf = XGBClassifier(gamma=6, max_depth=8, n_estimators=7, n_jobs=-1, random_state=25)
```

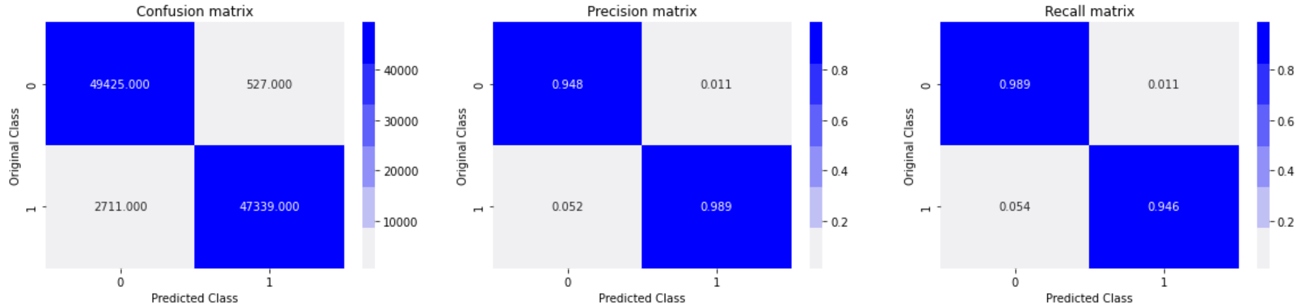
```
clf.fit(df_final_train,y_train)
y_train_pred = clf.predict(df_final_train)
y_test_pred = clf.predict(df_final_test)
```

```
print("Train score = {}".format(f1_score(y_train,y_train_pred)))
print("Test score = {}".format(f1_score(y_test,y_test_pred)))
```

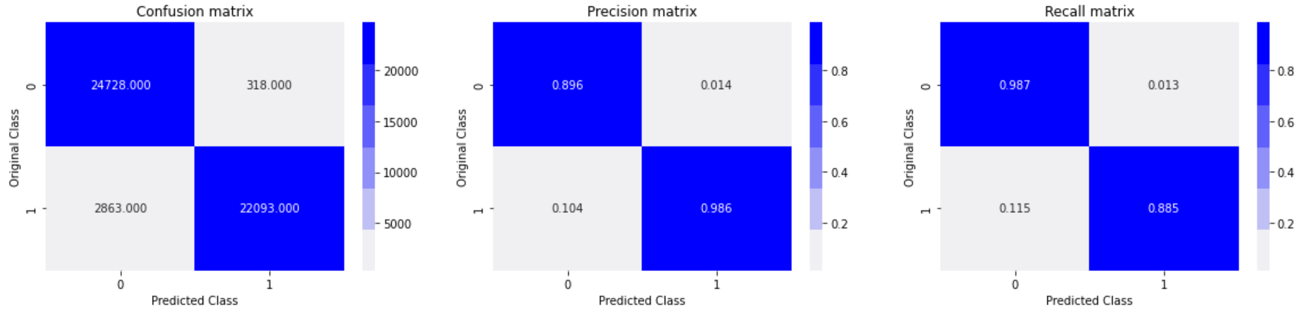
```
Train score = 0.9669308386780505
Test score = 0.9328435408617813
```

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

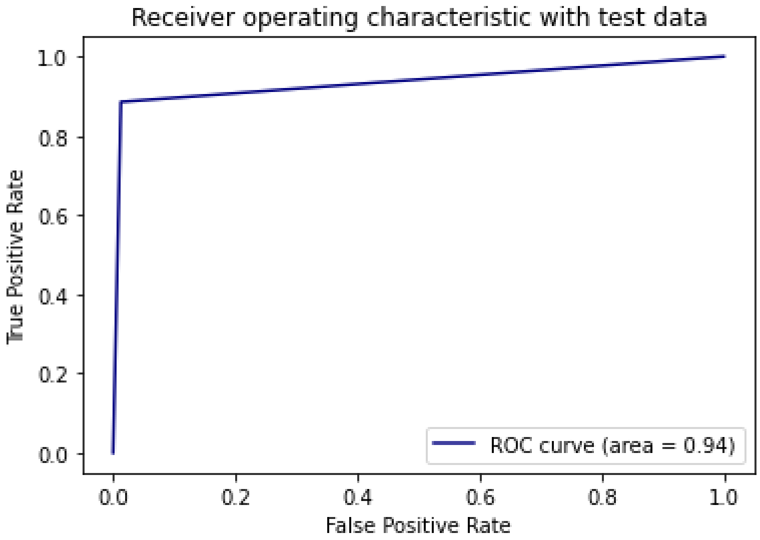
```
Train confusion_matrix
```



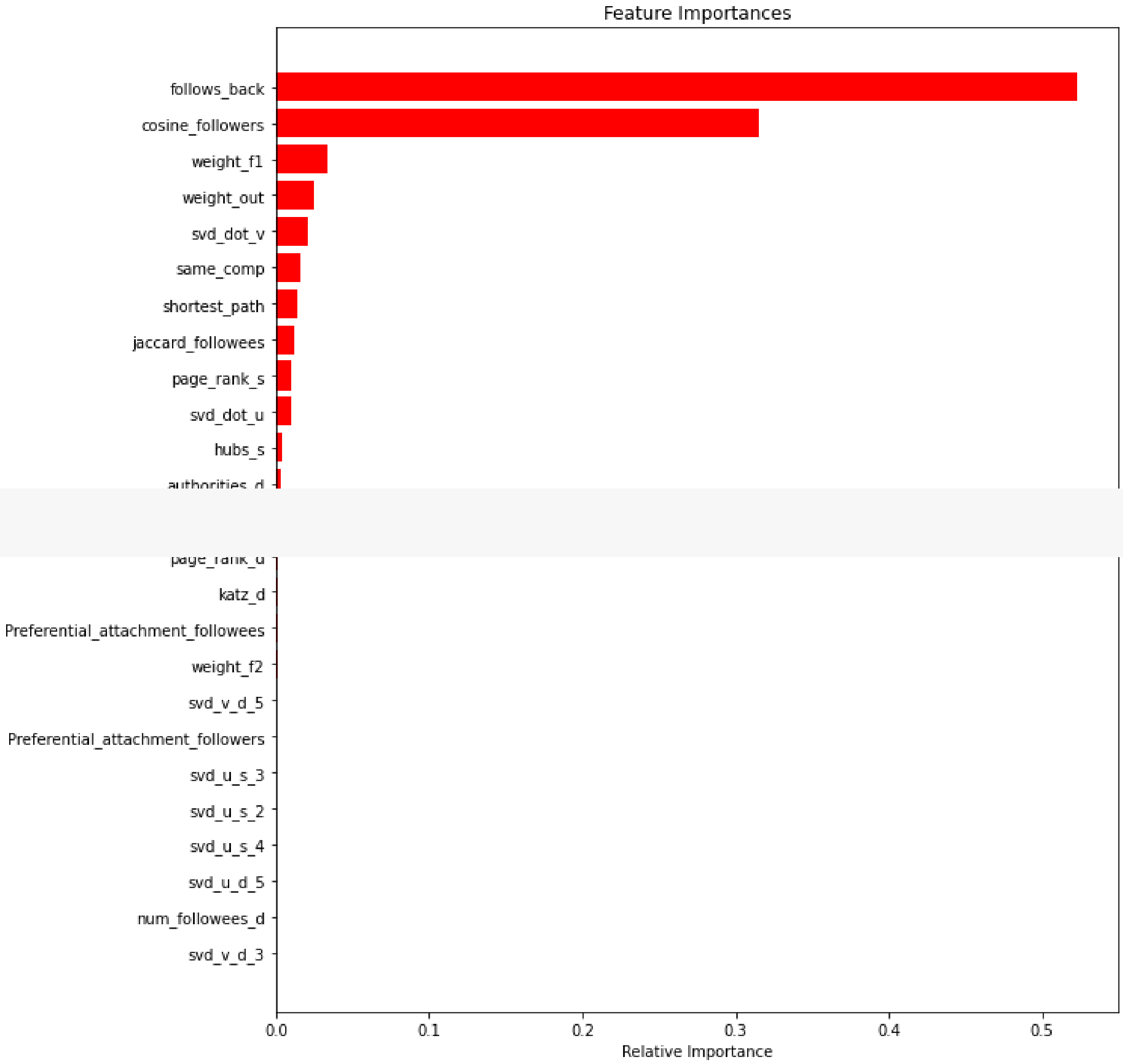
```
Test confusion_matrix
```



```
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()
```



```
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()
```


Feature Importances
follows_back
cosine_followers
weight_f1
weight_out
svd_dot_v
same_comp
shortest_path
jaccard_followees
page_rank_s
svd_dot_u
hubs_s
authorities_d
page_rank_d
katz_d
Preferential_attachment_followees
weight_f2
svd_v_d_5
Preferential_attachment_followers
svd_u_s_3
svd_u_s_2
svd_u_s_4
svd_u_d_5
num_followees_d
svd_v_d_3
0.0
0.1
0.2
0.3
0.4
0.5
Relative Importance