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

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

```python
#!pip3 install wget
```

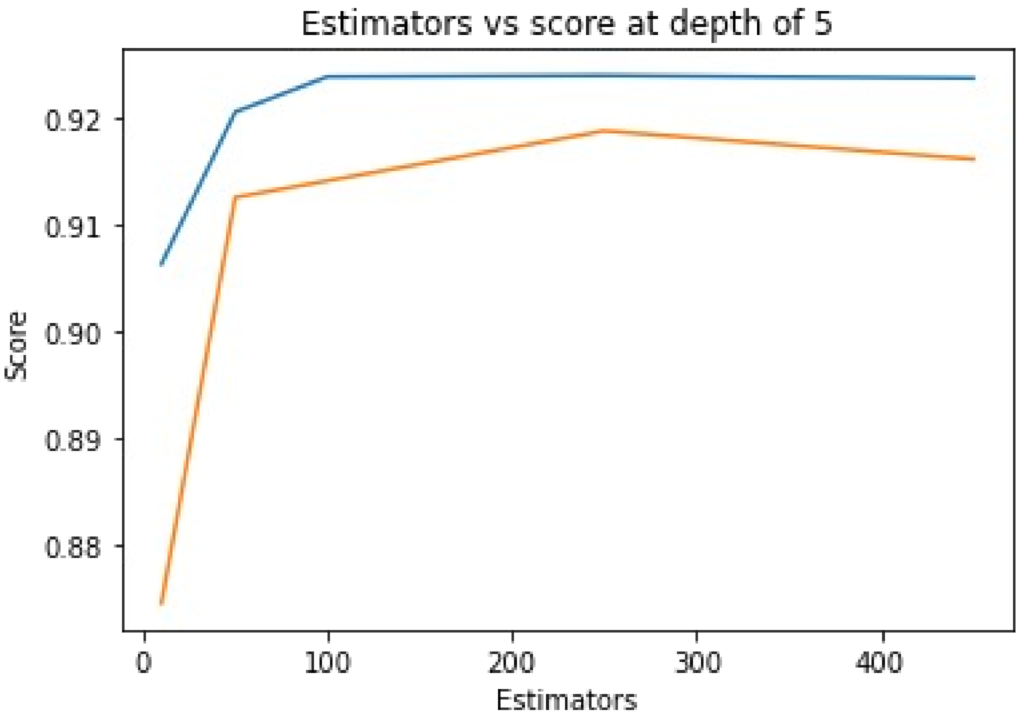
```python
#!wget --header="Host: doc-0o-bk-docs.googleusercontent.com" --
header="User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212
Safari/537.36" --header="Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image
/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9" --
header="Accept-Language: en-US,en;q=0.9" --header="Cookie:
AUTH_nso6dcn1mbidkt5qr539a2jiefc09pqv_nonce=iak2ig7rpq664" --
header="Connection: keep-alive" "https://doc-0o-bk-
docs.googleusercontent.com/docs/securesc/nss2f5s2soorprev6d4t4qp3n5ekp
9nh/
```

```
evl2j2j4t5hronicnhsbdlsblnbl9qk3/1622116650000/06629147635963609455/13
01756526451699381l/1fDJptlCFEWNV5UNGPc4geTykgFI3PDCV?
e=download&authuser=0&nonce=iak2ig7rpq664&user=13017565264516993811&ha
sh=fvl5s6dohfnqle6k8q3koe9jr2mhe6jr" -c -O 'storage_sample_stage4.h5'

#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')

#df_final_train.columns

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

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_impurity_split=None,
            min_samples_leaf=52, min_samples_split=120,
            min_weight_fraction_leaf=0.0, n_estimators=i, n_jobs=-
1,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')
```

```
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
```

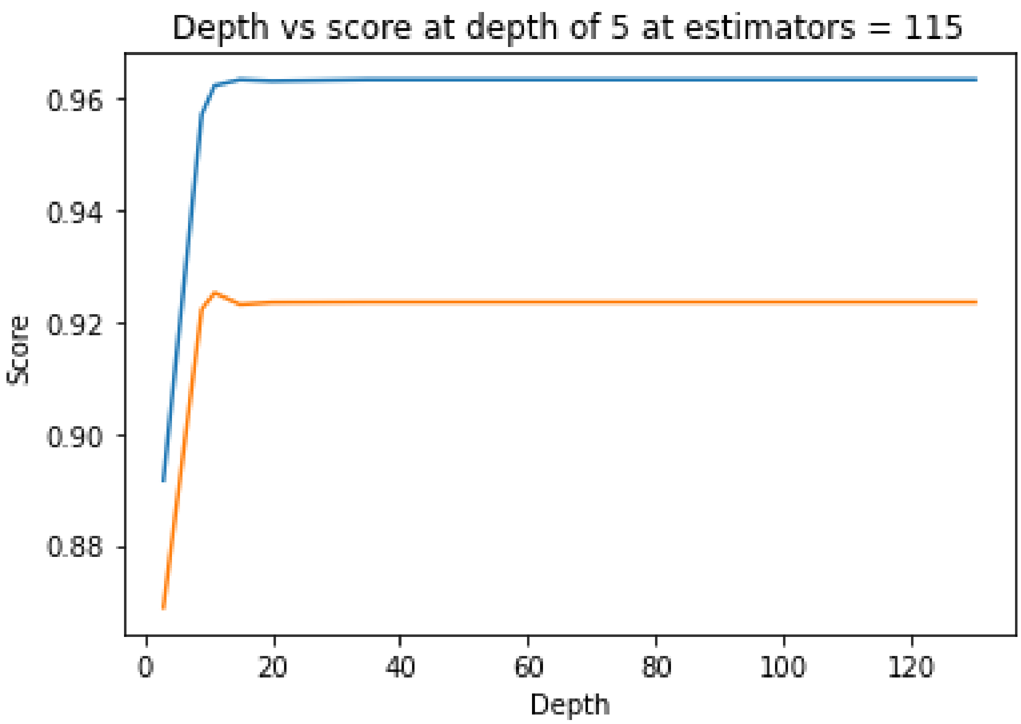
```
Text(0.5, 1.0, 'Estimators vs score at depth of 5')
```




```
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_impurity_split=None,
            min_samples_leaf=52, min_samples_split=120,
            min_weight_fraction_leaf=0.0, n_estimators=115, n_jobs=-
1,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('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 0.8916120853581238 test Score
0.8687934859875491
depth =  9 Train Score 0.9572226298198419 test Score
0.9222953031452904
depth =  11 Train Score 0.9623451340902863 test Score
0.9252318758281279
depth =  15 Train Score 0.9634267621927706 test Score
0.9231288356496615
depth =  20 Train Score 0.9631629153051491 test Score
0.9235051024711141
depth =  35 Train Score 0.9634333127085721 test Score
0.923560165275318
depth =  50 Train Score 0.9634333127085721 test Score
0.923560165275318
depth =  70 Train Score 0.9634333127085721 test Score
0.923560165275318
depth =  130 Train Score 0.9634333127085721 test Score
0.923560165275318
```




```
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(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_train_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 [0.96225042 0.96215492 0.9605708  0.96194014
0.96330005]
mean train scores [0.96294922 0.96266735 0.96115674 0.96263457
0.96430539]
```

```
print(rf_random.cv_results_.keys())
```

```
dict_keys(['mean_fit_time', 'std_fit_time', 'mean_score_time',
'std_score_time', 'param_max_depth', 'param_min_samples_leaf',
'param_min_samples_split', 'param_n_estimators', 'params',
'split0_test_score', 'split1_test_score', 'split2_test_score',
'split3_test_score', 'split4_test_score', 'split5_test_score',
'split6_test_score', 'split7_test_score', 'split8_test_score',
'split9_test_score', 'mean_test_score', 'std_test_score',
'rank_test_score', 'split0_train_score', 'split1_train_score',
'split2_train_score', 'split3_train_score', 'split4_train_score',
'split5_train_score', 'split6_train_score', 'split7_train_score',
'split8_train_score', 'split9_train_score', 'mean_train_score',
'std_train_score'])
```

```
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)
```

```
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)
```

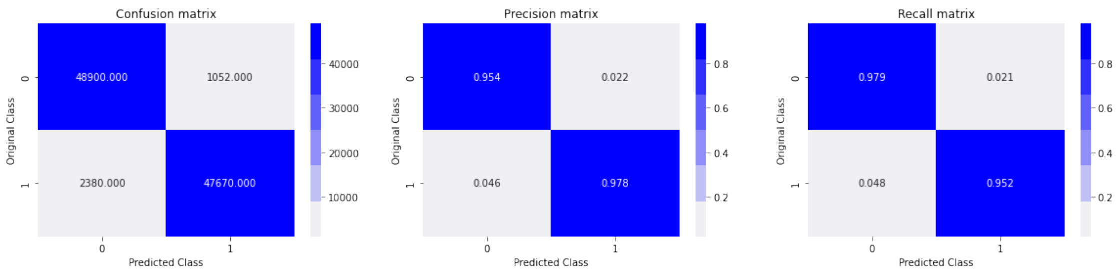
```
from sklearn.metrics import f1_score
print('Train f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))
```

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

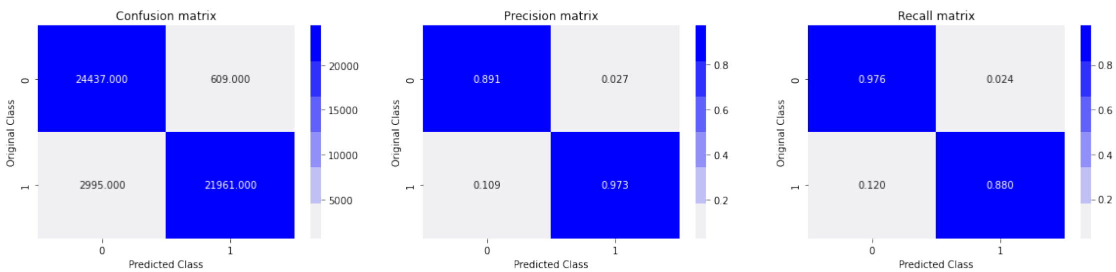
```
from sklearn.metrics import confusion_matrix
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)

    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=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()
```

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

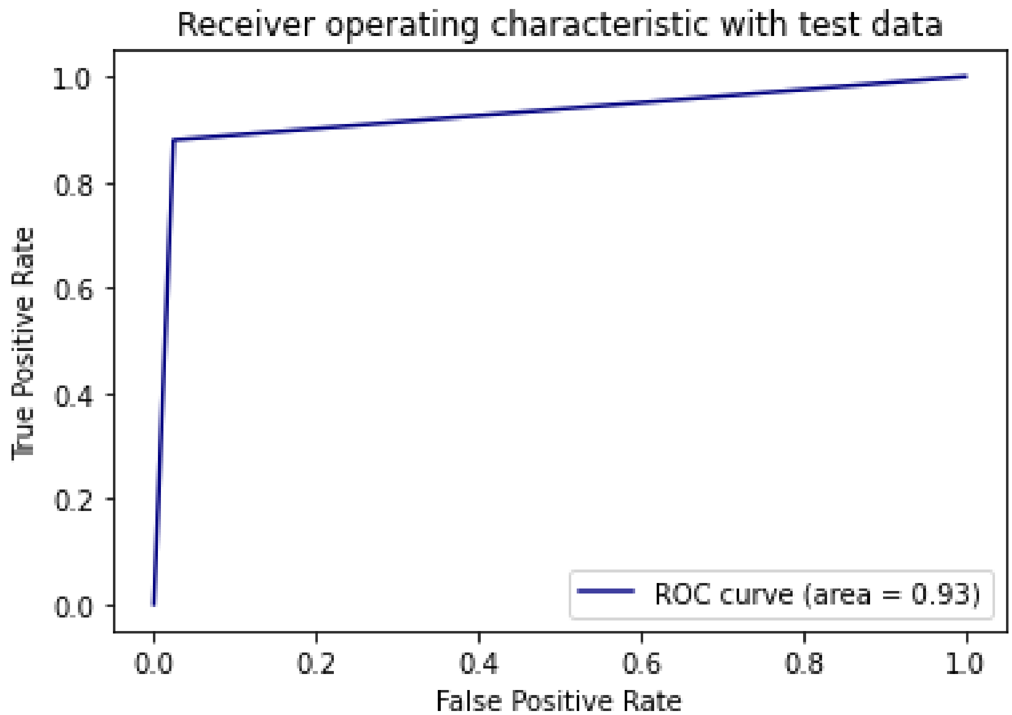
Train confusion_matrix



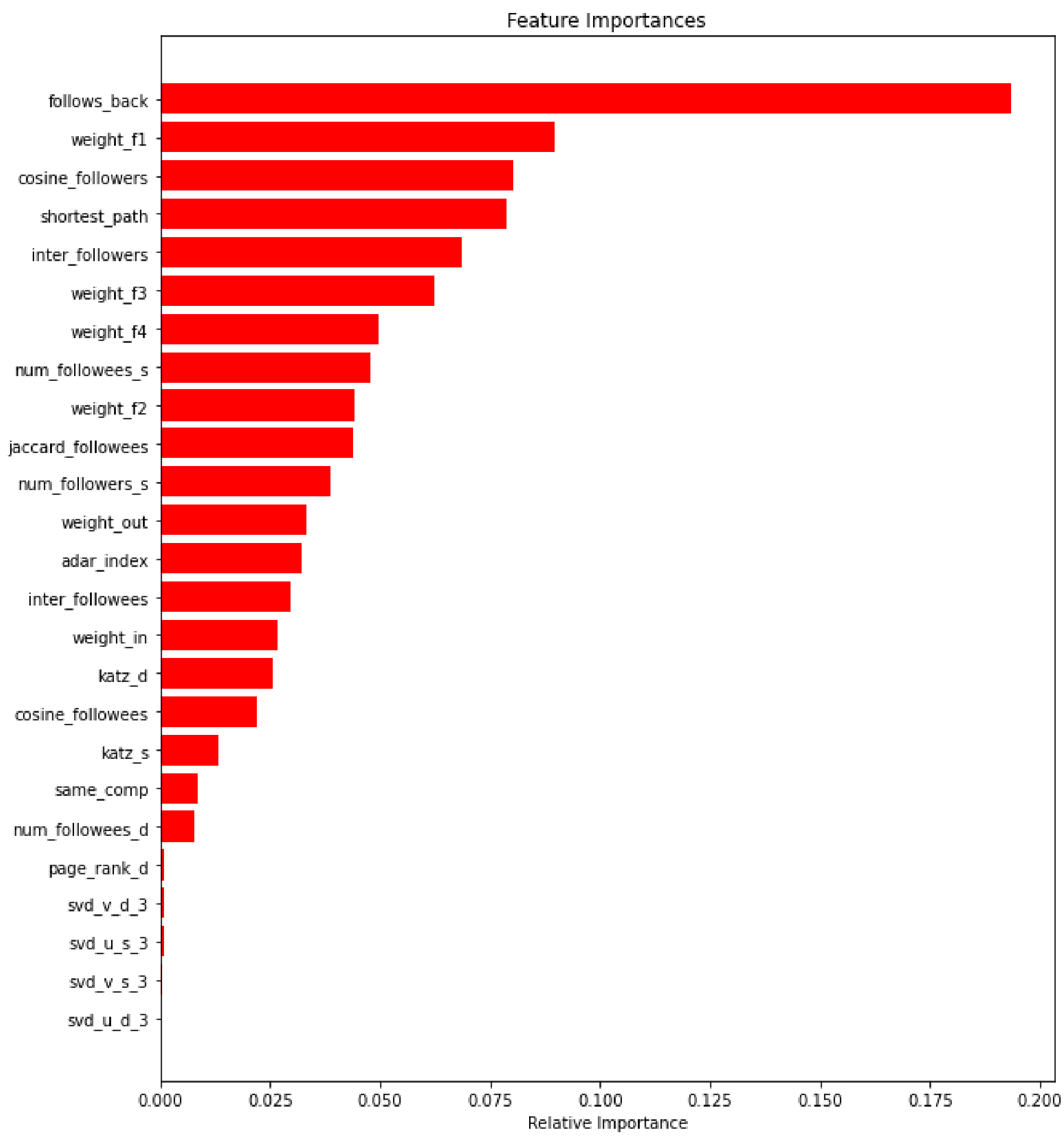
Test confusion_matrix




```
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.

Adding new features [Preferential Attachment ,SVD_dot]

```
df_final_train.columns
```

```
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'],
      dtype='object')
```

Oberservation:

- source_node,destination_node not found hence we have to get that back in the dataframe to compute new features. hence, reverting data to storage_sample_stage4.

```
#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')
```

```
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')
```

```
if os.path.isfile('data/after_eda/train_pos_after_eda.csv'):

train_graph=nx.read_edgelist('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
```

```
def followee_preferential_attachment(u1,u2):
    try:
        u_1 = len(set(train_graph.successors(u1)))
        u_2 = len(set(train_graph.successors(u2)))
        return(u_1*u_2)
    except:
        return(0)

def follower_preferential_attachment(u1,u2):
    try:
        u_1 = len(set(train_graph.predecessors(u1)))
        u_2 = len(set(train_graph.predecessors(u2)))
        return(u_1*u_2)
    except:
        return(0)
```

```
df_final_train.head()
```

```
   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

   jaccard_followees  cosine_followers  cosine_followees
num_followers_s  \
0           0.000000          0.000000          0.000000
```

```
6
1          0.187135         0.028382         0.343828
94
2          0.369565         0.156957         0.566038
28
3          0.000000         0.000000         0.000000
11
4          0.000000         0.000000         0.000000
1

   num_followees_s  num_followees_d  ...     svd_v_s_3
svd_v_s_4  \
0               15                8  ...  1.983691e-06  1.545075e-13

1               61              142  ... -6.236048e-11  1.345726e-02

2               41               22  ... -2.380564e-19 -7.021227e-19

3                5                7  ...  6.058498e-11  1.514614e-11

4               11                3  ...  1.197283e-07  1.999809e-14


      svd_v_s_5     svd_v_s_6     svd_v_d_1     svd_v_d_2
svd_v_d_3  \
0  8.108434e-13  1.719702e-14 -1.355368e-12  4.675307e-13  1.128591e-
06
1  3.703479e-12  2.251737e-10  1.245101e-12 -1.636948e-10 -3.112650e-
10
2  1.940403e-19 -3.365389e-19 -1.238370e-18  1.438175e-19 -1.852863e-
19
3  1.513483e-12  4.498061e-13 -9.818087e-10  3.454672e-11  5.213635e-
08
4  3.360247e-13  1.407670e-14  0.000000e+00  0.000000e+00
0.000000e+00

      svd_v_d_4     svd_v_d_5     svd_v_d_6
0  6.616550e-14  9.771077e-13  4.159752e-14
1  6.738902e-02  2.607801e-11  2.372904e-09
2 -5.901864e-19  1.629341e-19 -2.572452e-19
3  9.595823e-13  3.047045e-10  1.246592e-13
4  0.000000e+00  0.000000e+00  0.000000e+00

[5 rows x 54 columns]
```

```
if not os.path.isfile('data/fea_sample/storage_sample_stage5.h5'):

    df_final_train['followee_preferential_attachment'] =
```

```
df_final_train.apply(lambda row:
followee_preferential_attachment(row['source_node'],row['destination_n
ode']),axis=1)
    df_final_test['followee_preferential_attachment'] =
df_final_test.apply(lambda row:
followee_preferential_attachment(row['source_node'],row['destination_n
ode']),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(lambda row:
follower_preferential_attachment(row['source_node'],row['destination_n
ode']),axis=1)

    hdf = HDFStore('data/fea_sample/storage_sample_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('data/fea_sample/storage_sample_stage5.h5',
'train_df',mode='r')
    df_final_test =
read_hdf('data/fea_sample/storage_sample_stage5.h5',
'test_df',mode='r')


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',
       'followee_preferential_attachment',
'follower_preferential_attachment'],
      dtype='object')
```

```
df_final_train.followee_preferential_attachment
```

```
0          120
1         8662
2          902
3           35
4           33
          ...
99997       10
99998        4
99999        5
100000       0
100001       0
Name: followee_preferential_attachment, Length: 100002, dtype: int64
```

Adding SVD Dot feature

```
#for train datasets

#storing u and v matrices of source and destination nodes in saperate
arrays
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['svd_u_s_1'],df_final_train['svd_u_s_2'],df_final_train
['svd_u_s_3'],df_final_train['svd_u_s_4'],df_final_train['svd_u_s_5'],
df_final_train['svd_u_s_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 =
df_final_train['svd_v_s_1'],df_final_train['svd_v_s_2'],df_final_train
['svd_v_s_3'],df_final_train['svd_v_s_4'],df_final_train['svd_v_s_5'],
df_final_train['svd_v_s_6']

# converting dataframe to np.arrays for dot product computation
svd_u_s_1,svd_u_s_2,svd_u_s_3,svd_u_s_4,svd_u_s_5,svd_u_s_6 =
np.array( svd_u_s_1), np.array( svd_u_s_2), np.array( svd_u_s_3),
np.array( svd_u_s_4), np.array( svd_u_s_5), np.array( svd_u_s_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 =
np.array( svd_v_s_1), np.array( svd_v_s_2), np.array( svd_v_s_3),
np.array( svd_v_s_4), np.array( svd_v_s_5), np.array( svd_v_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 =
df_final_train['svd_u_d_1'],df_final_train['svd_u_d_2'],df_final_train
['svd_u_d_3'],df_final_train['svd_u_d_4'],df_final_train['svd_u_d_5'],
```

```
df_final_train['svd_u_d_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 =
df_final_train['svd_v_d_1'],df_final_train['svd_v_d_2'],df_final_train
['svd_v_d_3'],df_final_train['svd_v_d_4'],df_final_train['svd_v_d_5'],
df_final_train['svd_v_d_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 =
np.array( svd_u_d_1), np.array( svd_u_d_2), np.array( svd_u_d_3),
np.array( svd_u_d_4), np.array( svd_u_d_5), np.array( svd_u_d_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 =
np.array( svd_v_d_1), np.array( svd_v_d_2), np.array( svd_v_d_3),
np.array( svd_v_d_4), np.array( svd_v_d_5), np.array( svd_v_d_6)

svd_u_dot_train=[]
svd_v_dot_train=[]

for i in range(len(svd_u_s_1)):
    source_matrix_u=[]
    source_matrix_v=[]

    destination_matrix_u=[]
    destination_matrix_v=[]

    source_matrix_u.append(svd_u_s_1[i])
    source_matrix_u.append(svd_u_s_2[i])
    source_matrix_u.append(svd_u_s_3[i])
    source_matrix_u.append(svd_u_s_4[i])
    source_matrix_u.append(svd_u_s_5[i])
    source_matrix_u.append(svd_u_s_6[i])
    source_matrix_v.append(svd_v_s_1[i])
    source_matrix_v.append(svd_v_s_2[i])
    source_matrix_v.append(svd_v_s_3[i])
    source_matrix_v.append(svd_v_s_4[i])
    source_matrix_v.append(svd_v_s_5[i])
    source_matrix_v.append(svd_v_s_6[i])
    destination_matrix_u.append(svd_u_d_1[i])
    destination_matrix_u.append(svd_u_d_2[i])
    destination_matrix_u.append(svd_u_d_3[i])
    destination_matrix_u.append(svd_u_d_4[i])
    destination_matrix_u.append(svd_u_d_5[i])
    destination_matrix_u.append(svd_u_d_6[i])
    destination_matrix_v.append(svd_v_d_1[i])
    destination_matrix_v.append(svd_v_d_2[i])
    destination_matrix_v.append(svd_v_d_3[i])
    destination_matrix_v.append(svd_v_d_4[i])
    destination_matrix_v.append(svd_v_d_5[i])
    destination_matrix_v.append(svd_v_d_6[i])
    #print(source_matrix)

svd_u_dot_train.append(np.dot(source_matrix_u,destination_matrix_u))
```

```
svd_v_dot_train.append(np.dot(source_matrix_v,destination_matrix_v))


#for test datasets

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['svd_u_s_1'],df_final_test['svd_u_s_2'],df_final_test['s
vd_u_s_3'],df_final_test['svd_u_s_4'],df_final_test['svd_u_s_5'],df_fi
nal_test['svd_u_s_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 =
df_final_test['svd_v_s_1'],df_final_test['svd_v_s_2'],df_final_test['s
vd_v_s_3'],df_final_test['svd_v_s_4'],df_final_test['svd_v_s_5'],df_fi
nal_test['svd_v_s_6']

svd_u_s_1,svd_u_s_2,svd_u_s_3,svd_u_s_4,svd_u_s_5,svd_u_s_6 =
np.array( svd_u_s_1), np.array( svd_u_s_2), np.array( svd_u_s_3),
np.array( svd_u_s_4), np.array( svd_u_s_5), np.array( svd_u_s_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 =
np.array( svd_v_s_1), np.array( svd_v_s_2), np.array( svd_v_s_3),
np.array( svd_v_s_4), np.array( svd_v_s_5), np.array( svd_v_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 =
df_final_test['svd_u_d_1'],df_final_test['svd_u_d_2'],df_final_test['s
vd_u_d_3'],df_final_test['svd_u_d_4'],df_final_test['svd_u_d_5'],df_fi
nal_test['svd_u_d_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 =
df_final_test['svd_v_d_1'],df_final_test['svd_v_d_2'],df_final_test['s
vd_v_d_3'],df_final_test['svd_v_d_4'],df_final_test['svd_v_d_5'],df_fi
nal_test['svd_v_d_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 =
np.array( svd_u_d_1), np.array( svd_u_d_2), np.array( svd_u_d_3),
np.array( svd_u_d_4), np.array( svd_u_d_5), np.array( svd_u_d_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 =
np.array( svd_v_d_1), np.array( svd_v_d_2), np.array( svd_v_d_3),
np.array( svd_v_d_4), np.array( svd_v_d_5), np.array( svd_v_d_6)

svd_u_dot_test=[]
svd_v_dot_test=[]

for i in range(len(svd_u_s_1)):
    source_matrix_u=[]
    source_matrix_v=[]

    destination_matrix_u=[]
    destination_matrix_v=[]

    source_matrix_u.append(svd_u_s_1[i])
    source_matrix_u.append(svd_u_s_2[i])
```

```
    source_matrix_u.append(svd_u_s_3[i])
    source_matrix_u.append(svd_u_s_4[i])
    source_matrix_u.append(svd_u_s_5[i])
    source_matrix_u.append(svd_u_s_6[i])
    source_matrix_v.append(svd_v_s_1[i])
    source_matrix_v.append(svd_v_s_2[i])
    source_matrix_v.append(svd_v_s_3[i])
    source_matrix_v.append(svd_v_s_4[i])
    source_matrix_v.append(svd_v_s_5[i])
    source_matrix_v.append(svd_v_s_6[i])
    destination_matrix_u.append(svd_u_d_1[i])
    destination_matrix_u.append(svd_u_d_2[i])
    destination_matrix_u.append(svd_u_d_3[i])
    destination_matrix_u.append(svd_u_d_4[i])
    destination_matrix_u.append(svd_u_d_5[i])
    destination_matrix_u.append(svd_u_d_6[i])
    destination_matrix_v.append(svd_v_d_1[i])
    destination_matrix_v.append(svd_v_d_2[i])
    destination_matrix_v.append(svd_v_d_3[i])
    destination_matrix_v.append(svd_v_d_4[i])
    destination_matrix_v.append(svd_v_d_5[i])
    destination_matrix_v.append(svd_v_d_6[i])
    #print(source_matrix)

svd_u_dot_test.append(np.dot(source_matrix_u,destination_matrix_u))

svd_v_dot_test.append(np.dot(source_matrix_v,destination_matrix_v))


if not os.path.isfile('data/fea_sample/storage_sample_stage6.h5'):

    df_final_train['svd_dot_u']=svd_u_dot_train
    df_final_train['svd_dot_v']=svd_u_dot_train
    df_final_test['svd_dot_u']=svd_u_dot_test
    df_final_test['svd_dot_v']=svd_v_dot_test



    hdf = HDFStore('data/fea_sample/storage_sample_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('data/fea_sample/storage_sample_stage6.h5',
'train_df',mode='r')
    df_final_test =
read_hdf('data/fea_sample/storage_sample_stage6.h5',
```

```
'test_df',mode='r')
```

Final features

```
df_final_train.head(4)
```

```
   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

   jaccard_followees  cosine_followers  cosine_followees
num_followers_s  \
0           0.000000          0.000000          0.000000
6
1           0.187135          0.028382          0.343828
94
2           0.369565          0.156957          0.566038
28
3           0.000000          0.000000          0.000000
11

   num_followees_s  num_followees_d  ...     svd_v_d_1
svd_v_d_2  \
0               15                8  ... -1.355368e-12  4.675307e-13

1               61              142  ...  1.245101e-12 -1.636948e-10

2               41               22  ... -1.238370e-18  1.438175e-19

3                5                7  ... -9.818087e-10  3.454672e-11


      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

   followee_preferential_attachment  follower_preferential_attachment
\
0                               120                                66

1                              8662                              1598

2                               902                               980
```

```
3                                 35                                  22



      svd_dot_u     svd_dot_v
0  1.114958e-11  1.114958e-11
1  3.192812e-03  3.192812e-03
2  1.787503e-35  1.787503e-35
3  4.710376e-20  4.710376e-20

[4 rows x 58 columns]
```

```
df_final_test.head(4)
```

```
   source_node  destination_node  indicator_link  jaccard_followers  \
0       848424            784690               1                  0
1       483294           1255532               1                  0
2       626190           1729265               1                  0
3       947219            425228               1                  0

   jaccard_followees  cosine_followers  cosine_followees
num_followers_s  \
0                0.0          0.029161               0.0
14
1                0.0          0.000000               0.0
17
2                0.0          0.000000               0.0
10
3                0.0          0.000000               0.0
37

   num_followees_s  num_followees_d  ...     svd_v_d_1
svd_v_d_2  \
0                6                9  ... -9.994076e-10  5.791910e-10

1                1               19  ... -9.360516e-12  3.206809e-10

2               16                9  ... -4.253075e-13  4.789463e-13

3               10               34  ... -2.162590e-11  6.939194e-12


      svd_v_d_3     svd_v_d_4     svd_v_d_5     svd_v_d_6  \
0  3.512364e-07  2.486658e-09  2.771146e-09  1.727694e-12
1  4.668696e-08  6.665777e-12  1.495979e-10  9.836670e-14
2  3.479824e-07  1.630549e-13  3.954708e-13  3.875785e-14
3  1.879861e-05  4.384816e-12  1.239414e-11  6.483485e-13

   followee_preferential_attachment  follower_preferential_attachment
\
```

```
0                                 54                                  84

1                                 19                                  34

2                                144                                 150

3                                340                                 407


      svd_dot_u     svd_dot_v
0  8.425267e-20  2.074808e-17
1  1.352160e-17  1.188376e-17
2  3.671980e-13  3.904885e-12
3  1.634044e-10  9.819784e-11

[4 rows x 58 columns]
```

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

df_final_train.head(4)
```

```
   jaccard_followers  jaccard_followees  cosine_followers
cosine_followees  \
0                  0           0.000000          0.000000
0.000000
1                  0           0.187135          0.028382
0.343828
2                  0           0.369565          0.156957
0.566038
3                  0           0.000000          0.000000
0.000000

   num_followers_s  num_followees_s  num_followees_d  inter_followers
\
0                6               15                8                0

1               94               61              142               11

2               28               41               22               26

3               11                5                7                0


   inter_followees  adar_index  ...     svd_v_d_1     svd_v_d_2
```

```
svd_v_d_3  \
0                0    0.000000  ... -1.355368e-12  4.675307e-13
1.128591e-06
1               32   16.362912  ...  1.245101e-12 -1.636948e-10 -
3.112650e-10
2               17   10.991826  ... -1.238370e-18  1.438175e-19 -
1.852863e-19
3                0    0.000000  ... -9.818087e-10  3.454672e-11
5.213635e-08

      svd_v_d_4     svd_v_d_5     svd_v_d_6
followee_preferential_attachment  \
0  6.616550e-14  9.771077e-13  4.159752e-14
120
1  6.738902e-02  2.607801e-11  2.372904e-09
8662
2 -5.901864e-19  1.629341e-19 -2.572452e-19
902
3  9.595823e-13  3.047045e-10  1.246592e-13
35

   follower_preferential_attachment     svd_dot_u     svd_dot_v
0                                66  1.114958e-11  1.114958e-11
1                              1598  3.192812e-03  3.192812e-03
2                               980  1.787503e-35  1.787503e-35
3                                22  4.710376e-20  4.710376e-20

[4 rows x 55 columns]
```

Applying XGBoost

```
param_dist = {"n_estimators":sp_randint(105,125),
              "max_depth": sp_randint(2,10),
              "min_child_weight": [2,4,6,8],
              "learning_rate": [0.2,0.4,0.6,0.8]
              }

clf = xgb.XGBClassifier()

rf_random = RandomizedSearchCV(clf, param_distributions=param_dist,

n_iter=5,cv=4,scoring='f1',random_state=25,return_train_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 [0.98160845 0.98179245 0.98171623 0.98212202
0.98209461]
```

```
mean train scores [0.9965505  0.99976355 1.         0.99886723
0.99990676]
```

```
rf_random.best_estimator_
```

```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1,
colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1,
grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.4, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=7, max_leaves=0,
min_child_weight=8,
              missing=nan, monotone_constraints='()',
n_estimators=110,
              n_jobs=0, num_parallel_tree=1, predictor='auto',
random_state=0,
              reg_alpha=0, reg_lambda=1, ...)
```

```
#Best Parameters
print("Best parameters")

print("n_estimators :",rf_random.best_estimator_.n_estimators)
print("max_depth: ",rf_random.best_estimator_.max_depth)
print("min_child_weight: ",rf_random.best_estimator_.min_child_weight)
print("learning_rate",rf_random.best_estimator_.learning_rate)
```

```
Best parameters
n_estimators : 110
max_depth:  7
min_child_weight:  8
learning_rate 0.4
```

```
# using best parameters

clf=xgb.XGBClassifier(base_score=0.5, booster='gbtree',
callbacks=None,
              colsample_bylevel=1, colsample_bynode=1,
colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1,
grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.4, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=7, max_leaves=0,
min_child_weight=8,
              monotone_constraints='()', n_estimators=110,
              n_jobs=0, num_parallel_tree=1, predictor='auto',
```

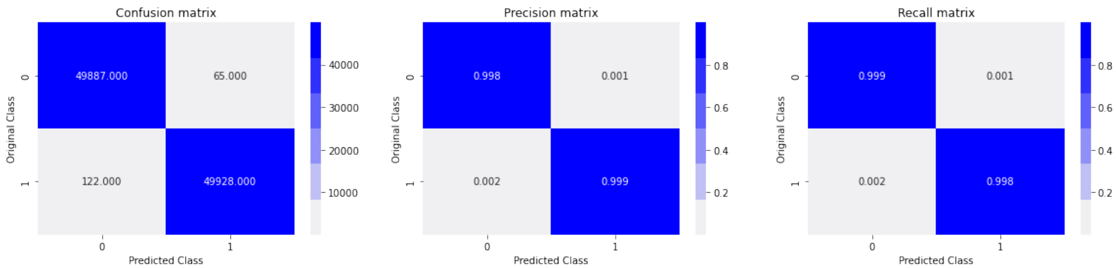
```
random_state=0,
              reg_alpha=0, reg_lambda=1)

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 f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))
```

```
Train f1 score 0.9981308037543857
Test f1 score 0.9175332734197801
```

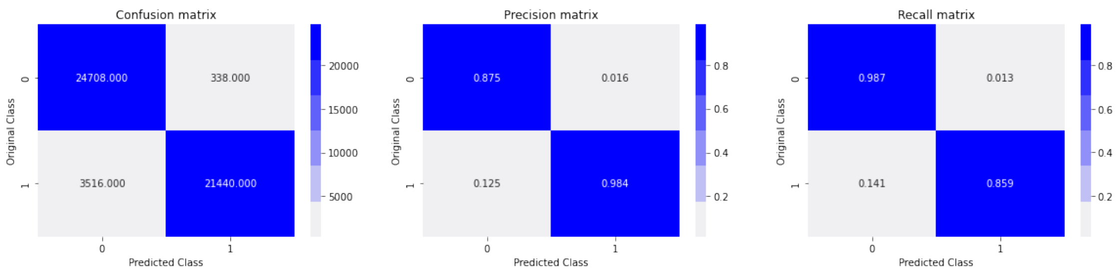
```
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)

    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()

print('Train confusion_matrix')
plot_confusion_matrix(y_train,y_train_pred)
```

```
print('Test confusion_matrix')
plot_confusion_matrix(y_test,y_test_pred)
```

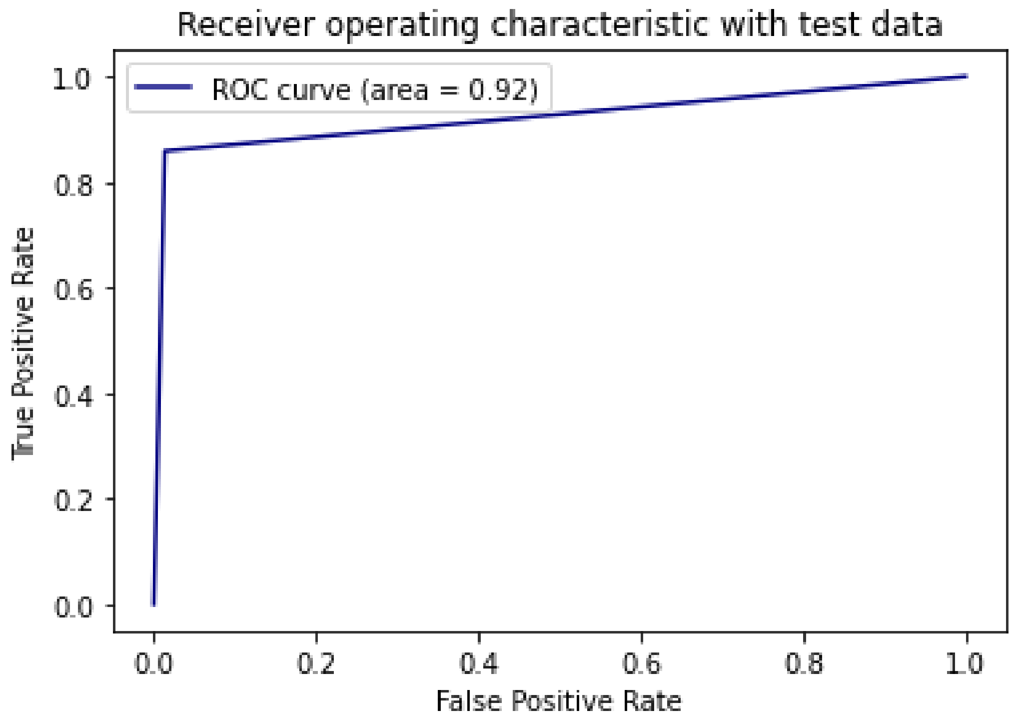
```
Train confusion_matrix
```



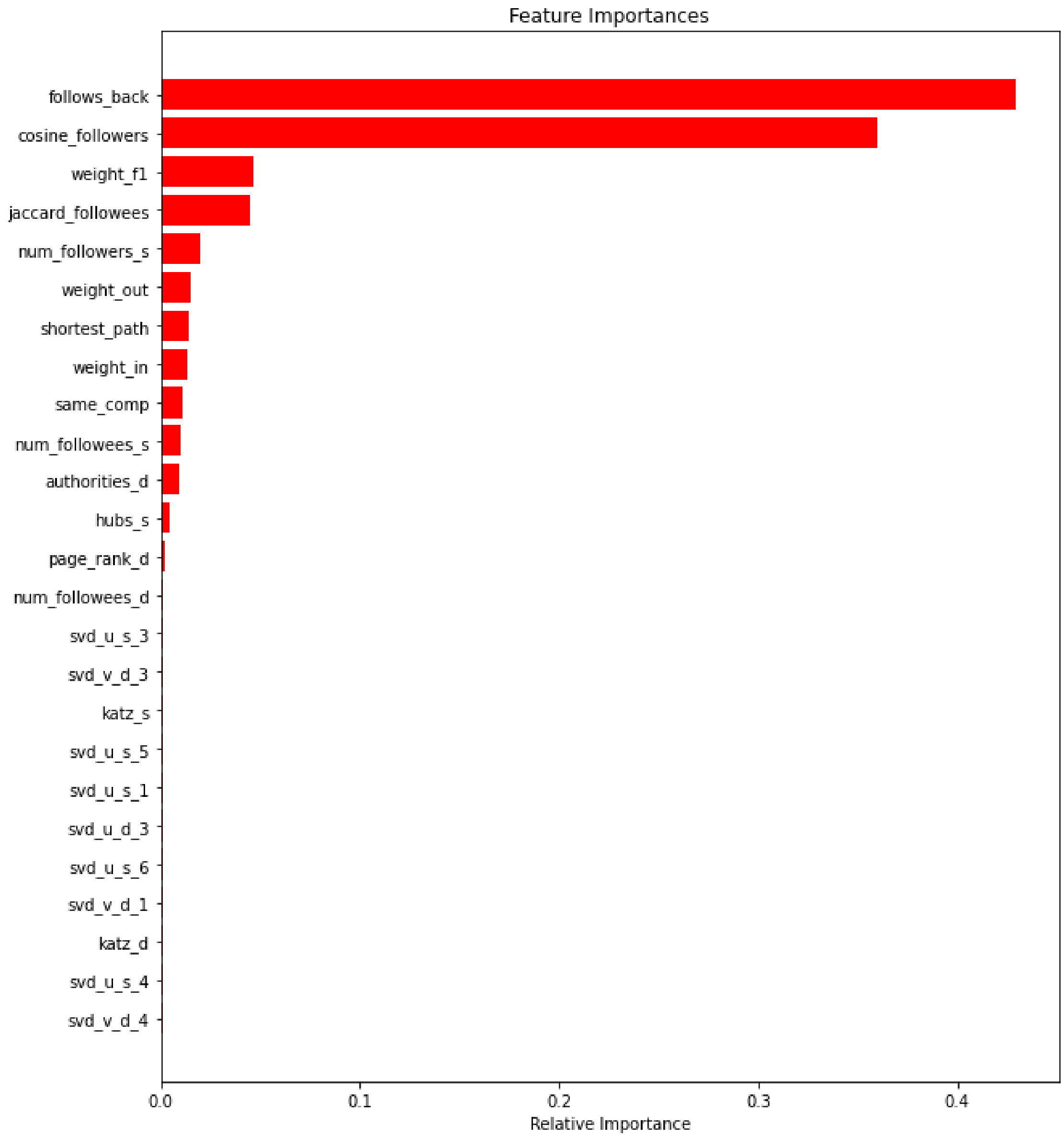

```
Test confusion_matrix
```




```
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()
```



Obervation:

- Preferential attachment and svd_dot features found to be not important as per XGBoost model
- XGBoost performs similar to RandomForest in results but from performace point of view XGBoost took longer duration in hyper-paramter tunning