Social network Graph Link Prediction - Facebook Challenge

```
#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
```

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

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491h

Enter your authorization code:
··········
Mounted at /content/drive
```

```
#reading
from pandas import read_hdf
df_final_train = read_hdf('/content/drive/My Drive/fb/data/fea_sample/storage_sample_stage5.h5', 'train_df',mode='r')
df_final_test = read_hdf('/content/drive/My Drive/fb/data/fea_sample/storage_sample_stage5.h5', 'test_df',mode='r')
```

```
df_final_train.columns
```

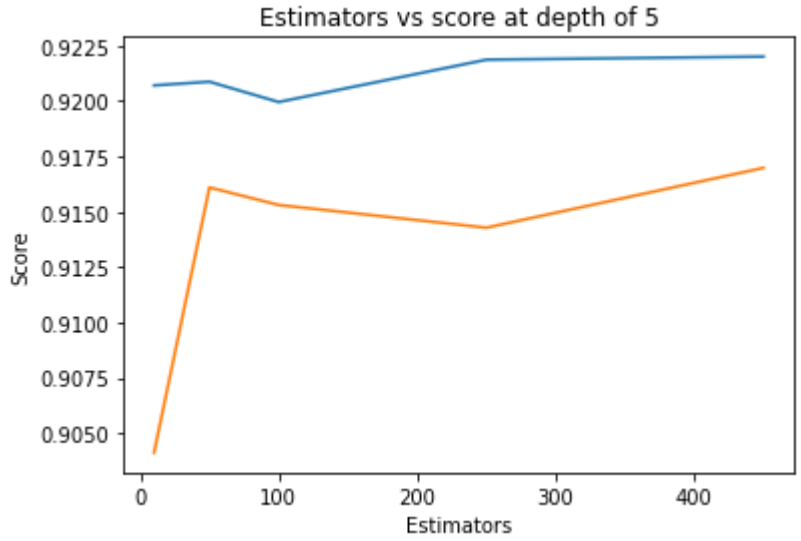
```
Index(['source_node', 'destination_node', 'indicator_link',
       'jaccard_followers', 'jaccard_followees', 'cosine_followers',
       'cosine_followees', 'num_followers_s', 'num_followees_s',
       'num_followees_d', 'inter_followers', 'inter_followees', 'adar_index',
       'follows_back', 'same_comp', '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',
       'svd_u_dot', 'svd_v_dot', 'preferential_attachment'],
      dtype='object')
```

```
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')
```

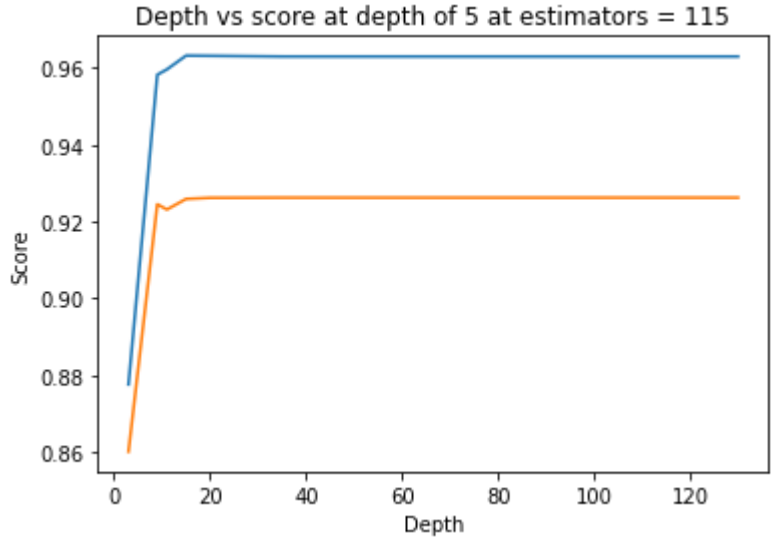
```
Estimators =  10 Train Score 0.9207091900965169 test Score 0.9041201057749724
Estimators =  50 Train Score 0.9208753329503316 test Score 0.916101961607049
Estimators =  100 Train Score 0.9199582027168234 test Score 0.9153082845574506
Estimators =  250 Train Score 0.9218707532771633 test Score 0.9142821073893285
Estimators =  450 Train Score 0.9220104348553445 test Score 0.9169850934285114
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 0.8776223036050721 test Score 0.860115656641658
depth =  9 Train Score 0.9582802450385803 test Score 0.9245545305193984
depth =  11 Train Score 0.9596646180145713 test Score 0.9231709888256379
depth =  15 Train Score 0.9632629036513832 test Score 0.9259376117456511
depth =  20 Train Score 0.9632095482639982 test Score 0.926225887203652
depth =  35 Train Score 0.9630261141389844 test Score 0.9262741633537368
depth =  50 Train Score 0.9630261141389844 test Score 0.9262741633537368
depth =  70 Train Score 0.9630261141389844 test Score 0.9262741633537368
depth =  130 Train Score 0.9630261141389844 test Score 0.9262741633537368
```




```
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(51,75),
              "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 [0.96143223 0.96228888 0.9595566  0.96135154 0.96289924]
mean train scores [0.96224636 0.96336683 0.95976278 0.96194756 0.96384523]
```

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

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=14, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=28, min_samples_split=111,
                       min_weight_fraction_leaf=0.0, n_estimators=73, n_jobs=-1,
                       oob_score=False, random_state=25, verbose=0,
                       warm_start=False)
```

```
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=73, 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
```

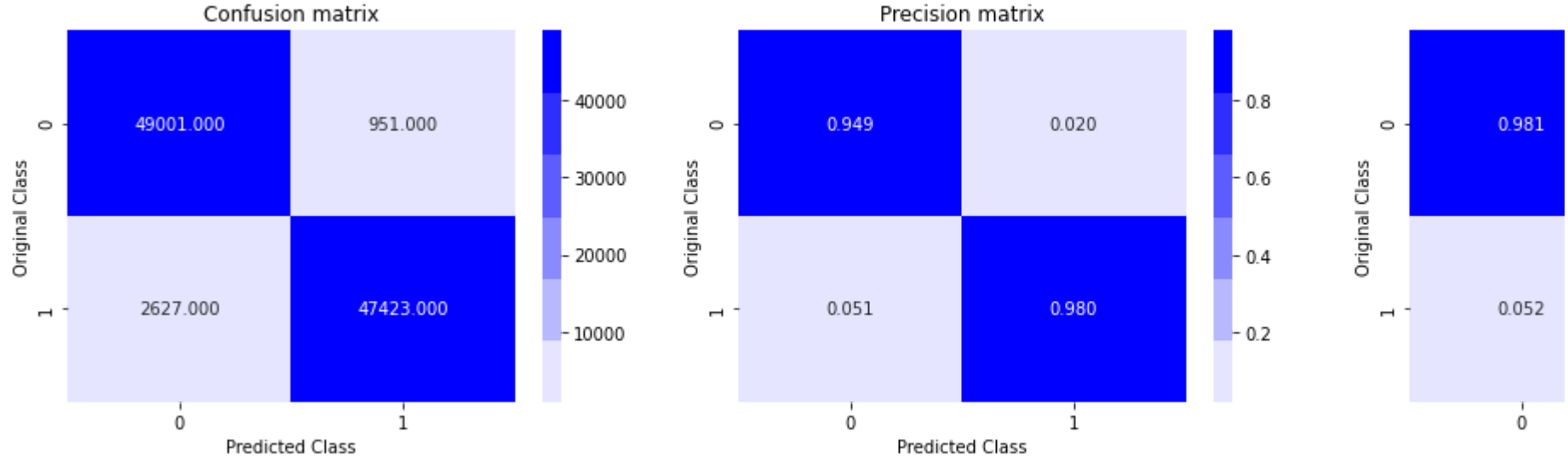
```
print('Train f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))
```

```
Train f1 score 0.9636470779484678
Test f1 score 0.9267624626016602
```

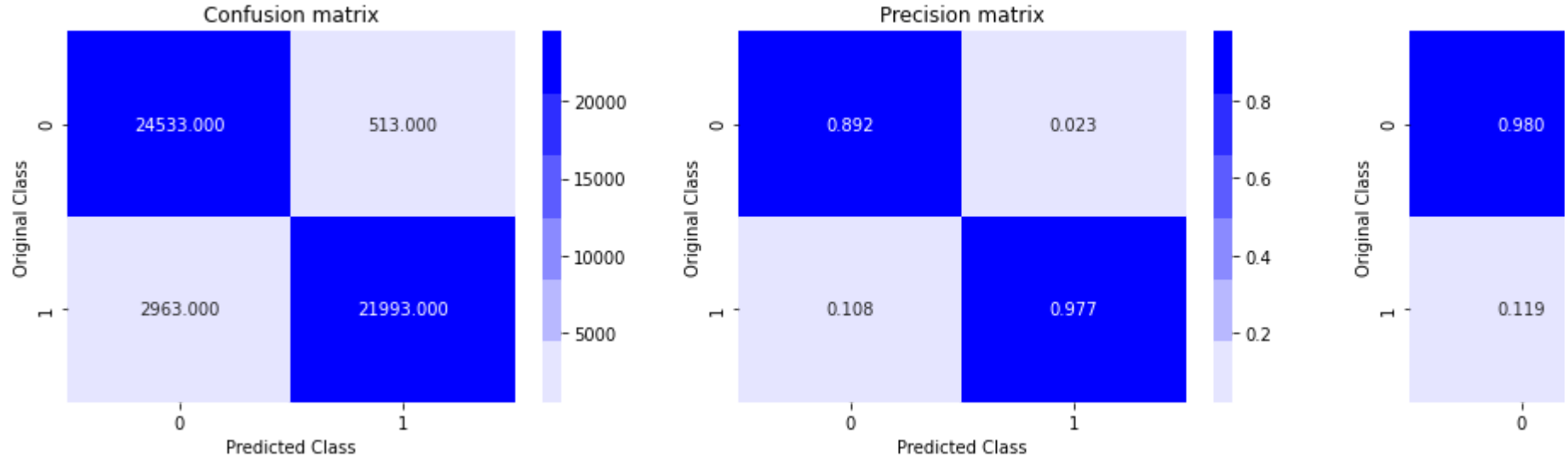
```
from sklearn.metrics import confusion_matrix
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)

    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

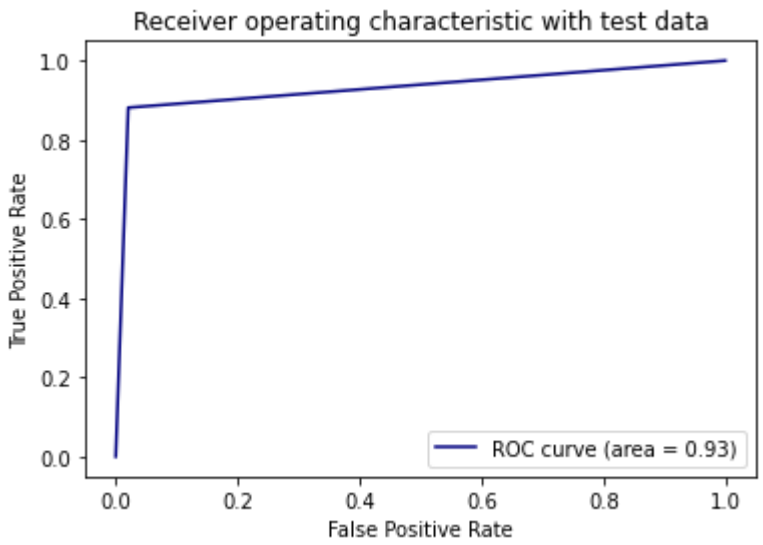


Test confusion_matrix

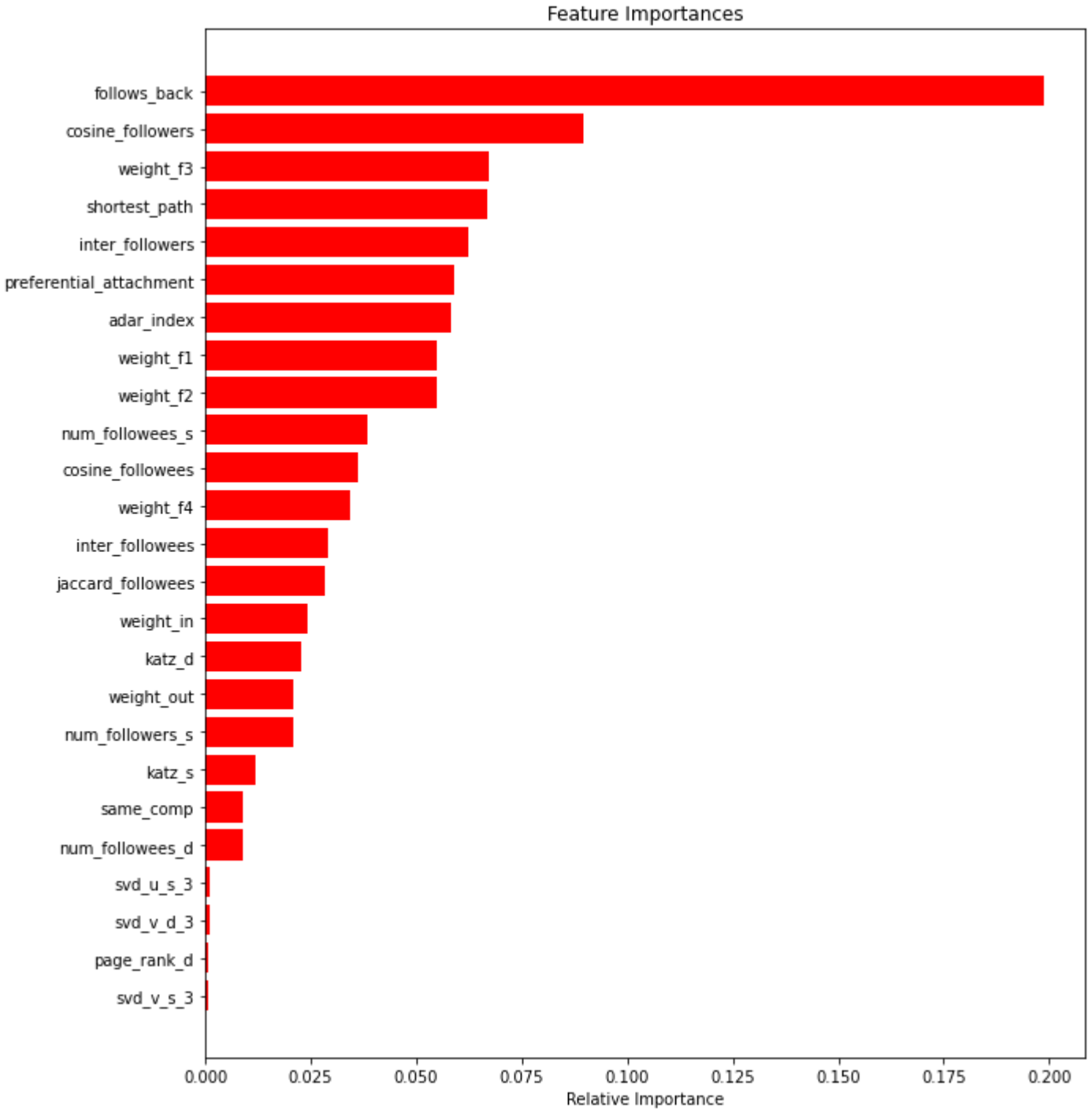



```
from sklearn.metrics import roc_curve, auc
```

```
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()
```


Feature Importances
follows_back
cosine_followers
weight_f3
shortest_path
inter_followers
preferential_attachment
adar_index
weight_f1
weight_f2
num_followees_s
cosine_followees
weight_f4
inter_followees
jaccard_followees
weight_in
katz_d
weight_out
num_followers_s
katz_s
same_comp
num_followees_d
svd_u_s_3
svd_v_d_3
page_rank_d
svd_v_s_3
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200
Relative Importance

Assignments:

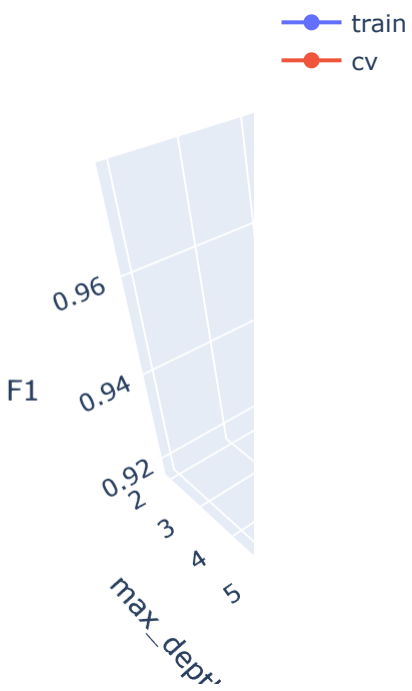
1. Add another feature called Preferential Attachment with followers and followees data of vertex. you can check about Preferential Attachment in belc http://be.amazd.com/link-prediction/
2. Add feature called svd_dot. you can calculate svd_dot as Dot product between sourse node svd and destination node svd features. you can read abc 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.

XGBoost

```
from sklearn.model_selection import GridSearchCV
import xgboost as xgb
import matplotlib.pyplot as plt


tuned_parameters = {'max_depth':[2,3,4,5,6,7,8,9,10],'n_estimators':[10,50,100,150,200,300,500]}

clf = xgb.XGBClassifier(class_weight='balanced', n_jobs=-1)

#Using GridSearchCV
model = GridSearchCV(clf, tuned_parameters, scoring = 'f1',verbose=5,n_jobs=-1,return_train_score=True)
model.fit(df_final_train,y_train)
train_f1 = model.cv_results_['mean_train_score']
cv_f1 = model.cv_results_['mean_test_score']



print(model.best_estimator_)
print(model.score(df_final_test,y_test))
```

```
Fitting 5 folds for each of 63 candidates, totalling 315 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:  1.8min
[Parallel(n_jobs=-1)]: Done  68 tasks      | elapsed: 34.5min
[Parallel(n_jobs=-1)]: Done 158 tasks      | elapsed: 107.1min
[Parallel(n_jobs=-1)]: Done 284 tasks      | elapsed: 300.8min
[Parallel(n_jobs=-1)]: Done 315 out of 315 | elapsed: 370.4min finished
XGBClassifier(base_score=0.5, booster='gbtree', class_weight='balanced',
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              gamma=0, learning_rate=0.1, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=None, n_estimators=500, n_jobs=-1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
0.9217799594493651
```

```
# https://plot.ly/python/3d-axes
import plotly.graph_objs as go
import plotly.offline as offline
max_depth = [2,3,4,5,6,7,8,9,10]
n_estimators = [10,50,100,150,200,300,500]
trace1 = go.Scatter3d(x=n_estimators,y=max_depth,z=train_f1, name = 'train')
trace2 = go.Scatter3d(x=n_estimators,y=max_depth,z=cv_f1, name = 'cv')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='n_estimators'),
        yaxis = dict(title='max_depth'),
        zaxis = dict(title='F1'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```




```
clf = xgb.XGBClassifier(base_score=0.5, booster='gbtree', class_weight='balanced',
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              gamma=0, learning_rate=0.1, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=None, n_estimators=500, n_jobs=-1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

```
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))
```

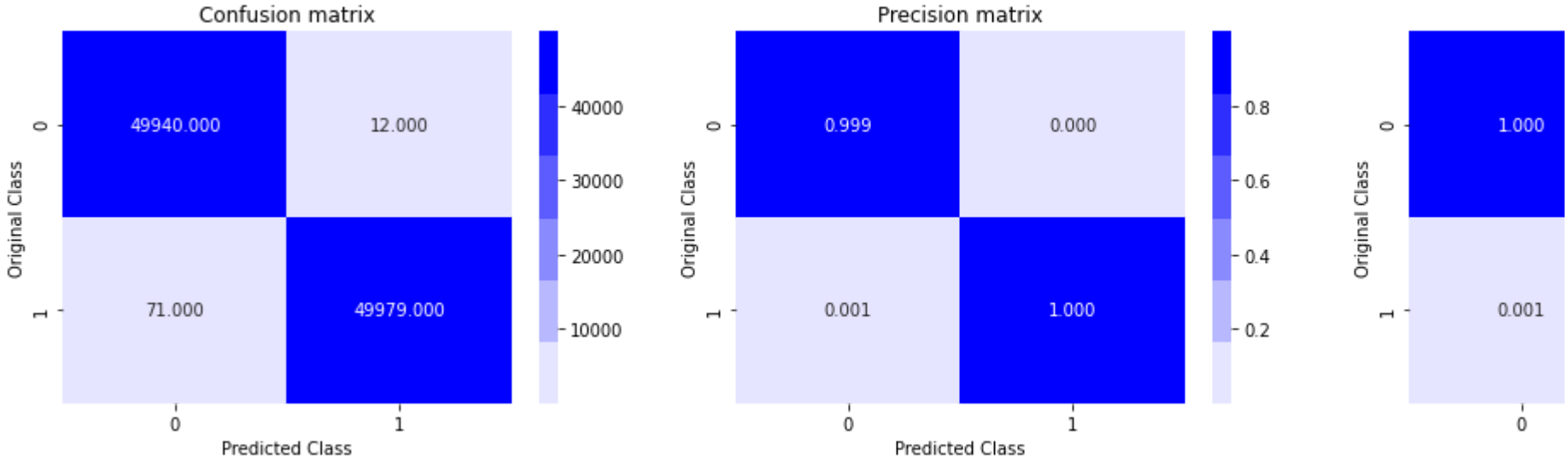
```
Train f1 score 0.9991703401605342
Test f1 score 0.9217799594493651
```

```
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)
```

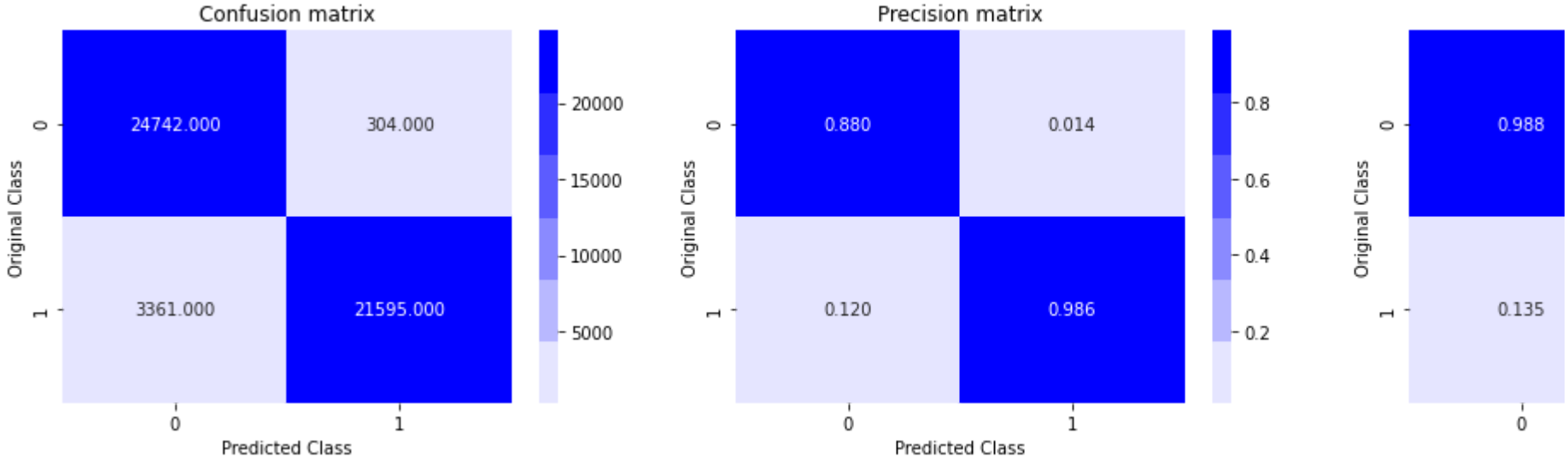
```
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    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
```

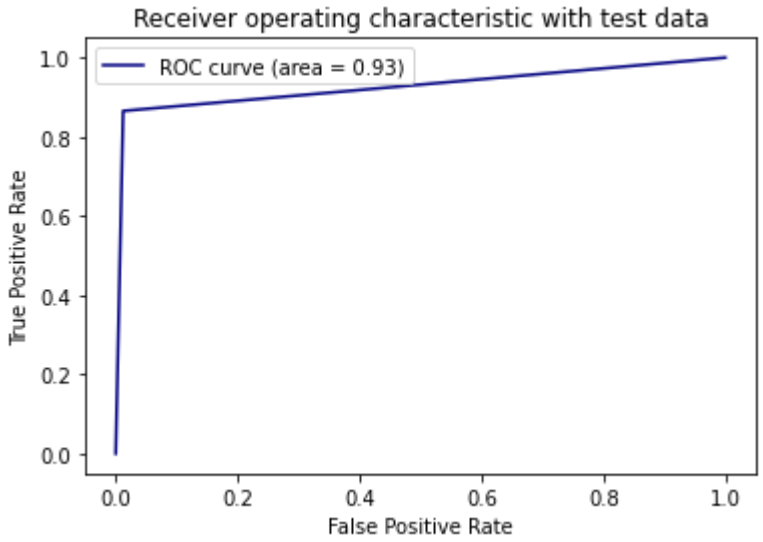



```
Test confusion_matrix
```

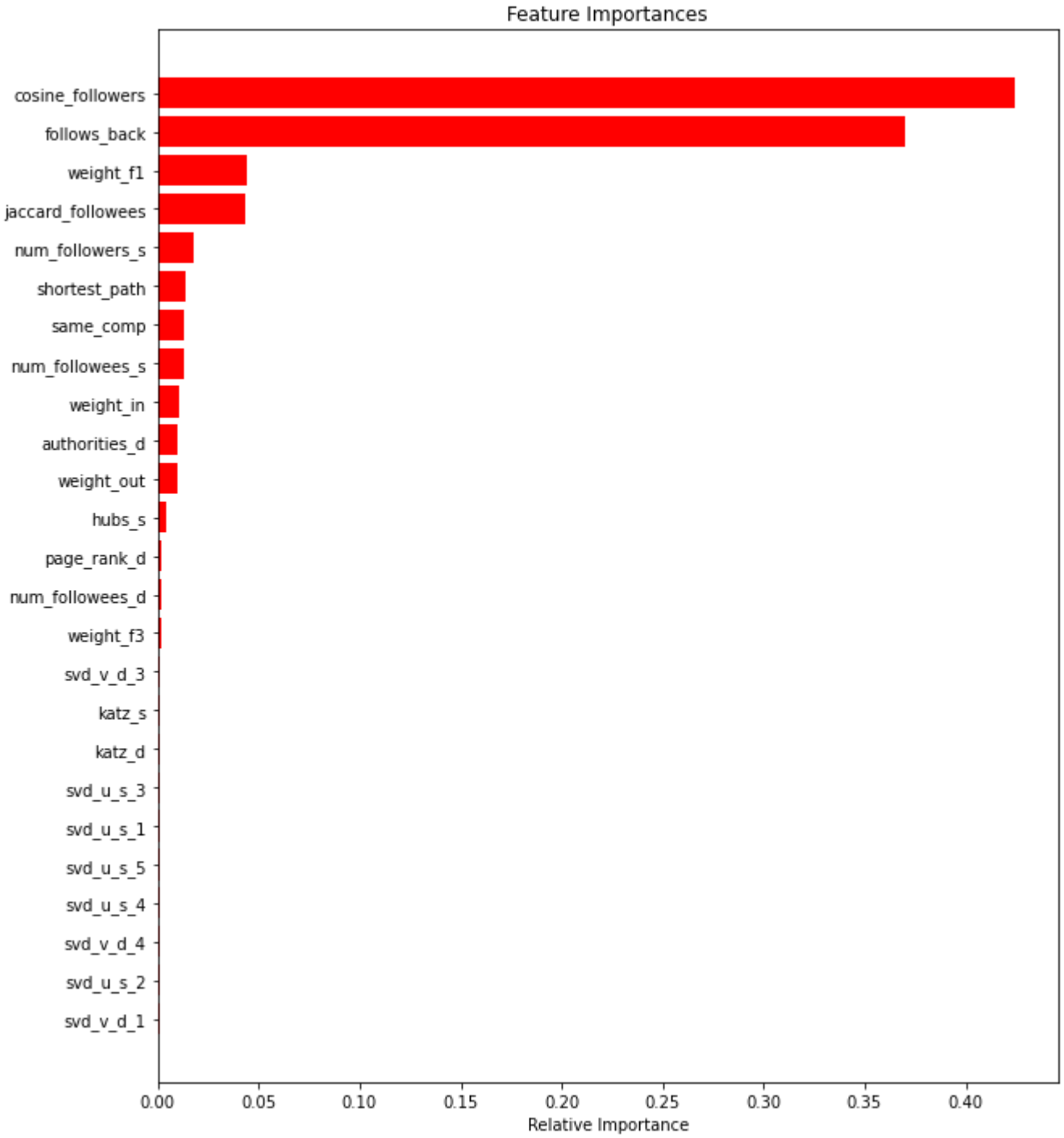



```
from sklearn.metrics import roc_curve, auc
fpr,tpr,ths = roc_curve(y_test,y_test_pred)
auc_sc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='navy',label='ROC curve (area = %0.2f)' % auc_sc)
plt.xlabel('False Positive Rate')
```

```
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()
```


Feature Importances
cosine_followers
follows_back
weight_f1
jaccard_followees
num_followers_s
shortest_path
same_comp
num_followees_s
weight_in
authorities_d
weight_out
hubs_s
page_rank_d
num_followees_d
weight_f3
svd_v_d_3
katz_s
katz_d
svd_u_s_3
svd_u_s_1
svd_u_s_5
svd_u_s_4
svd_v_d_4
svd_u_s_2
svd_v_d_1
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
Relative Importance

Conclusion

```
# Please compare all your models using Prettytable library
# http://zetcode.com/python/prettytable/

from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Model", "Hyperparameter(max_depth)", "Hyperparameter(n_estimators)", "Train F1", "Test F1"]


x.add_row(["Random Forest", 14, 73, 0.96364, 0.92676])
x.add_row(["GBDT", 6, 500, 0.99917, 0.92177])



print(x)
```

```
+---------------+---------------------------+-------------------------------+----------+---------+
|     Model     | Hyperparameter(max_depth) | Hyperparameter(n_estimators) | Train F1 | Test F1 |
+---------------+---------------------------+-------------------------------+----------+---------+
| Random Forest |             14            |              73               | 0.96364  | 0.92676 |
|      GBDT     |             6             |              500              | 0.99917  | 0.92177 |
+---------------+---------------------------+-------------------------------+----------+---------+
```