Social network Graph Link Prediction - Facebook Challenge

Below code is same code present in original notebook (FB_Models.ipynb). code blocks after the Assignments section is written by me.

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

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

import warnings
import networkx as nx
import pdb
import pickle
from pandas import HDFStore,DataFrame
from pandas import read_hdf
from scipy.sparse.linalg import svds, eigs
import gc
from tqdm import tqdm
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
```

In [2]:
```
#reading
from pandas import read_hdf
df_final_train = read_hdf('data/fea_sample/storage_sample_stage4.h5', 'train_df',mode='r')
df_final_test = read_hdf('data/fea_sample/storage_sample_stage4.h5', 'test_df',mode='r')
```

In [3]:
```
df_final_train.columns
```

Out[3]:
```
Index(['source_node', 'destination_node', 'indicator_link',
       'jaccard_followers', 'jaccard_followees', 'cosine_followers',
       'cosine_followees', 'num_followers_s', 'num_followees_s',
       'num_followees_d', 'inter_followers', 'inter_followees', 'adar_index',
       'follows_back', 'same_comp', 'shortest_path', 'weight_in', 'weight_out',
       'weight_f1', 'weight_f2', 'weight_f3', 'weight_f4', 'page_rank_s',
       'page_rank_d', 'katz_s', 'katz_d', 'hubs_s', 'hubs_d', 'authorities_s',
       'authorities_d', 'svd_u_s_1', 'svd_u_s_2', 'svd_u_s_3', 'svd_u_s_4',
       'svd_u_s_5', 'svd_u_s_6', 'svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3',
       'svd_u_d_4', 'svd_u_d_5', 'svd_u_d_6', 'svd_v_s_1', 'svd_v_s_2',
       'svd_v_s_3', 'svd_v_s_4', 'svd_v_s_5', 'svd_v_s_6', 'svd_v_d_1',
       'svd_v_d_2', 'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5', 'svd_v_d_6'],
      dtype='object')
```

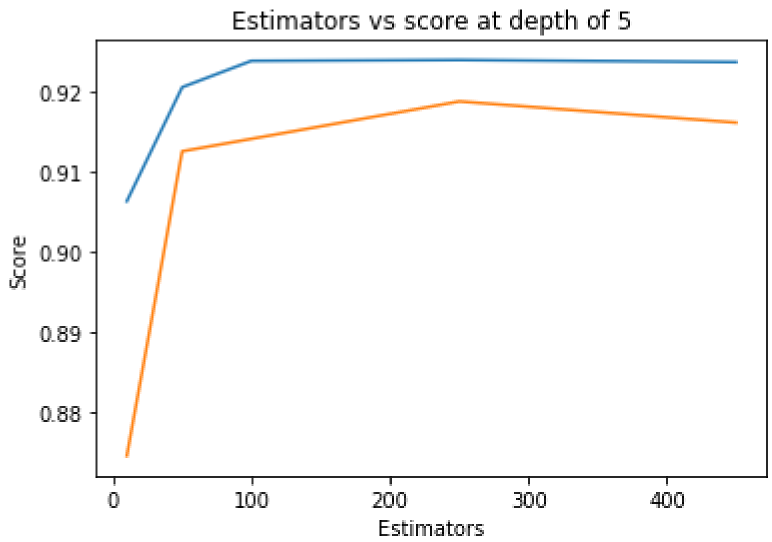
In [4]:
```
y_train = df_final_train.indicator_link
y_test = df_final_test.indicator_link
```

In [5]:
```
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)
```

In [6]:
```
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')
```

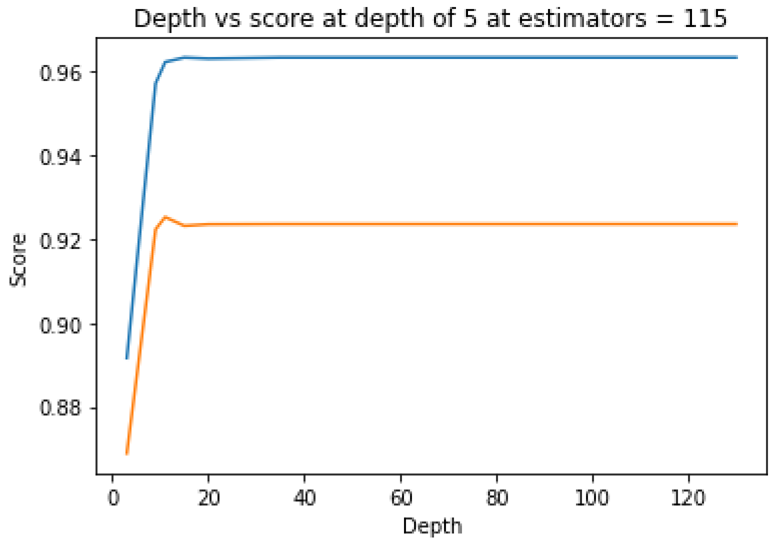
```
Estimators =  10 Train Score 0.9063252121775113 test Score 0.8745605278006858
Estimators =  50 Train Score 0.9205725512208812 test Score 0.9125653355634538
Estimators =  100 Train Score 0.9238690848446947 test Score 0.9141199714153599
Estimators =  250 Train Score 0.9239789348046863 test Score 0.9188007232664732
Estimators =  450 Train Score 0.9237190618658074 test Score 0.9161507685828595
```

Out[6]:
```
Text(0.5, 1.0, 'Estimators vs score at depth of 5')
```


Estimators vs score at depth of 5
Score
0.92
0.91
0.90
0.89
0.88
0
100
200
300
400
Estimators

In [7]:
```
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 0.9235601652753184
depth =  50 Train Score 0.9634333127085721 test Score 0.9235601652753184
depth =  70 Train Score 0.9634333127085721 test Score 0.9235601652753184
depth =  130 Train Score 0.9634333127085721 test Score 0.9235601652753184
```



In [9]:

```
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 [0.96225043 0.96215493 0.96057081 0.96194015 0.96330005]
mean train scores [0.96294922 0.96266735 0.96115674 0.96263457 0.96430539]
```

In [10]:
```
print(rf_random.best_estimator_)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=14, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=28, min_samples_split=111,
                       min_weight_fraction_leaf=0.0, n_estimators=121,
                       n_jobs=-1, oob_score=False, random_state=25, verbose=0,
                       warm_start=False)
```

In [94]:
```
clf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=14, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=28, min_samples_split=111,
            min_weight_fraction_leaf=0.0, n_estimators=121, n_jobs=-1,
            oob_score=False, random_state=25, verbose=0, warm_start=False)
```

In [95]:
```
clf.fit(df_final_train,y_train)
y_train_pred = clf.predict(df_final_train)
y_test_pred = clf.predict(df_final_test)
```

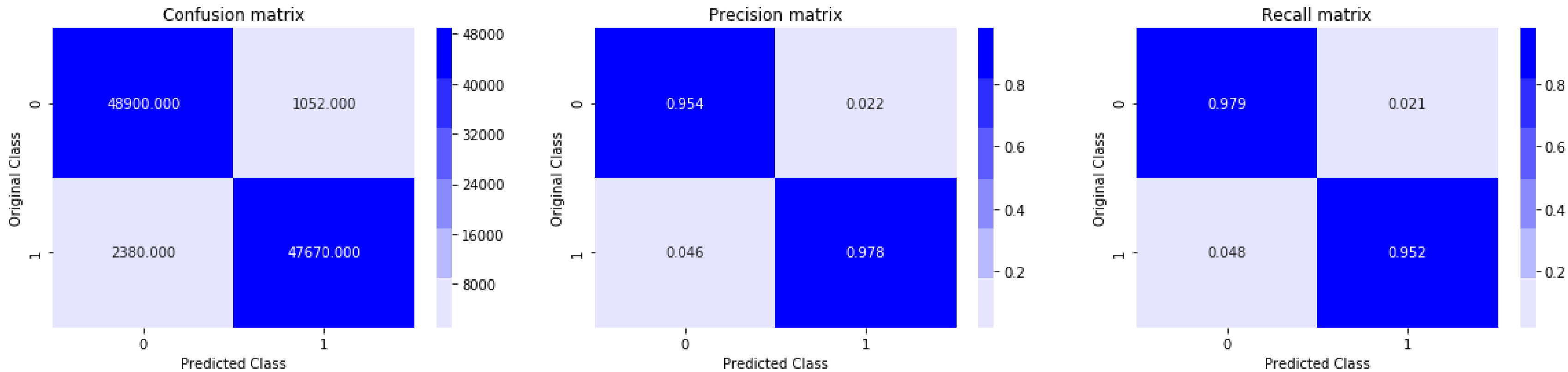
In [13]:
```
from sklearn.metrics import f1_score
print('Train f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))
```

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

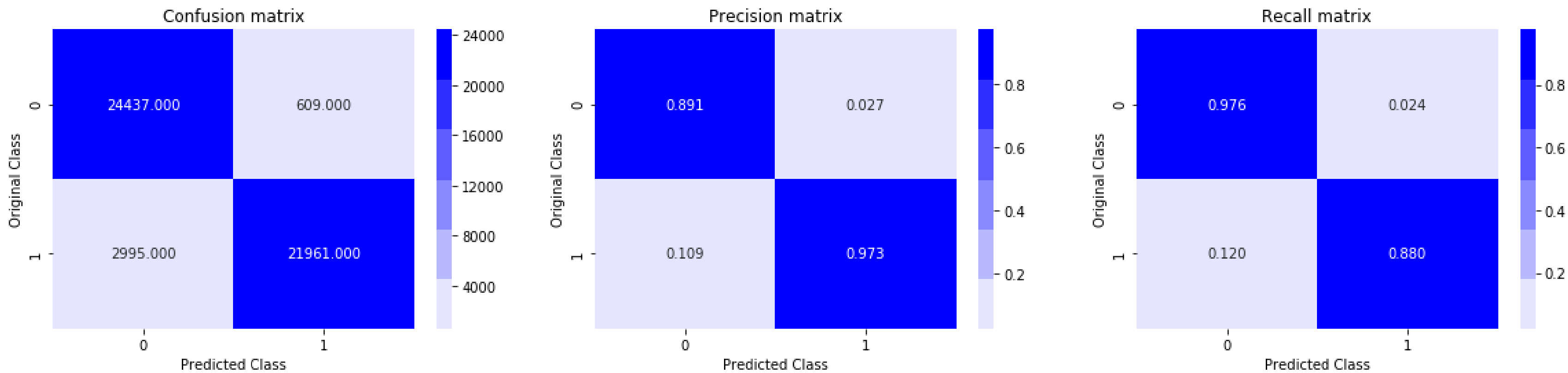
In [14]:
```
from sklearn.metrics import confusion_matrix
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)

    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()
```

In [15]:
```
print('Train confusion_matrix')
plot_confusion_matrix(y_train,y_train_pred)
print('Test confusion_matrix')
plot_confusion_matrix(y_test,y_test_pred)
```

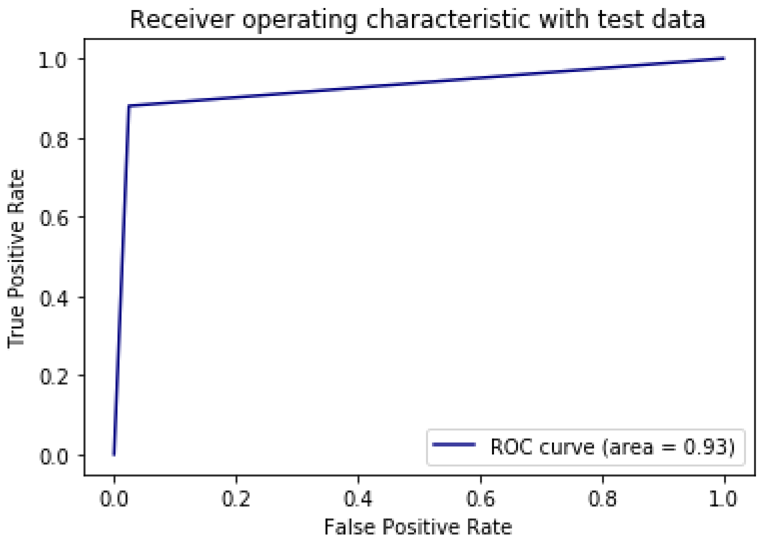
```
Train confusion_matrix
```




```
Test confusion_matrix
```



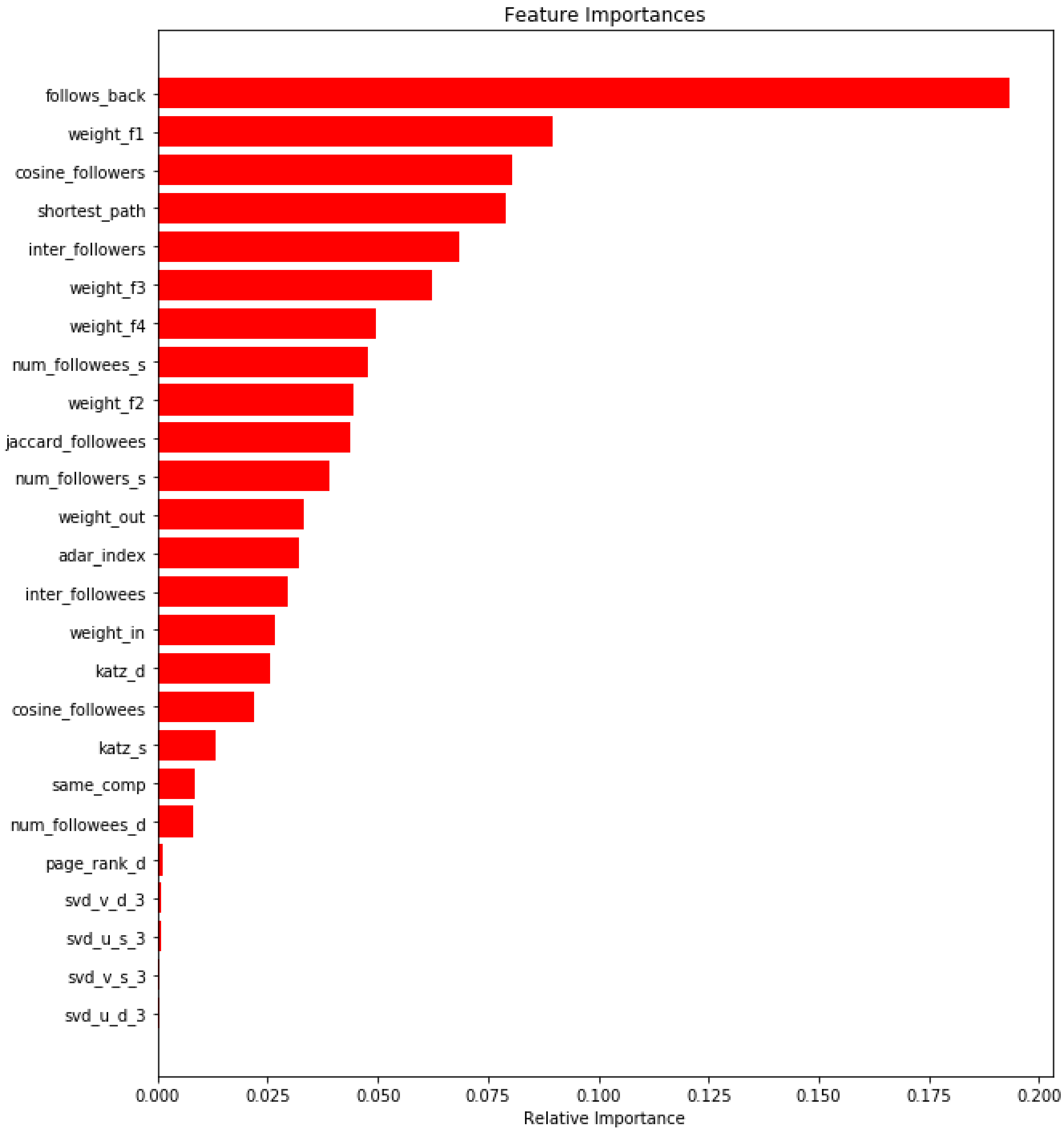
In [16]:
```
from sklearn.metrics import roc_curve, auc
fpr,tpr,ths = roc_curve(y_test,y_test_pred)
auc_sc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='navy',label='ROC curve (area = %0.2f)' % auc_sc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic with test data')
plt.legend()
plt.show()
```



In [96]:
```
sum(clf.feature_importances_)
```

Out[96]: 1.0

In [17]:
```
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
weight_f1
cosine_followers
shortest_path
inter_followers
weight_f3
weight_f4
num_followees_s
weight_f2
jaccard_followees
num_followers_s
weight_out
adar_index
inter_followees
weight_in
katz_d
cosine_followees
katz_s
same_comp
num_followees_d
page_rank_d
svd_v_d_3
svd_u_s_3
svd_v_s_3
svd_u_d_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 below link http://be.amazd.com/link-prediction/ (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 (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.

Below code blocks are written by me.

In [18]:
```
df_final_train.head()
```

Out[18]:

	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	num_followers_s	num_followees_s	num_followees_d	inter_fo
0	0	0.000000	0.000000	0.000000	6	15	8	
1	0	0.187135	0.028382	0.343828	94	61	142	
2	0	0.369565	0.156957	0.566038	28	41	22	
3	0	0.000000	0.000000	0.000000	11	5	7	
4	0	0.000000	0.000000	0.000000	1	11	3	

5 rows × 51 columns

In [19]:
```
df_final_test.head()
```

Out[19]:

	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	num_followers_s	num_followees_s	num_followees_d	inter_fo
0	0	0.0	0.029161	0.000000	14	6	9	
1	0	0.0	0.000000	0.000000	17	1	19	
2	0	0.0	0.000000	0.000000	10	16	9	
3	0	0.0	0.000000	0.000000	37	10	34	
4	0	0.2	0.042767	0.347833	27	15	27	

5 rows × 51 columns

In [20]:
```
print(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')
```

Importing graph data

In [21]:
```
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(
    print(nx.info(train_graph))
else:
    print("please run the FB_EDA.ipynb or download the files from drive")
```

```
Name:
Type: DiGraph
Number of nodes: 1780722
Number of edges: 7550015
Average in degree:   4.2399
Average out degree:   4.2399
```

Adding Preferential Attachment to the columns

In [22]:
```
def pref_attach_followees(a, b):
    try:
        sim = (len(set(train_graph.successors(a)))*len(set(train_graph.successors(b))))
    except:
        return 0
    return sim
```

In [23]:
```
#one test case
print(pref_attach_followees(273084,1505602))
```

```
120
```

In [26]:
```
def pref_attach_followers(a, b):
    try:
        sim = (len(set(train_graph.predecessors(a)))*len(set(train_graph.predecessors(b))))
    except:
        return 0
    return sim
```

In [28]:
```
print(pref_attach_followers(273084,470294))
```

```
220
```

In [29]:
```
print(pref_attach_followers(273084,1505602))
```

```
66
```

In [31]:
```
df_final_train = read_hdf('data/fea_sample/storage_sample_stage4.h5', 'train_df',mode='r')
df_final_test = read_hdf('data/fea_sample/storage_sample_stage4.h5', 'test_df',mode='r')
```

In [32]:
```
df_final_train['pref_attach_followers'] = df_final_train.apply(lambda row:
                                            pref_attach_followers(row['source_node'],row['destination_node']),ax
df_final_train['pref_attach_followees'] = df_final_train.apply(lambda row:
                                            pref_attach_followees(row['source_node'],row['destination_node']),ax
```

In [34]:
```
df_final_test['pref_attach_followers'] = df_final_test.apply(lambda row:
                                            pref_attach_followers(row['source_node'],row['destination_node']),ax
df_final_test['pref_attach_followees'] = df_final_test.apply(lambda row:
                                            pref_attach_followees(row['source_node'],row['destination_node']),ax
```

As we use Tree based models we dont need to normalize the values as we get same results even after normalization.

Adding SVD dot product values as columns

In [38]:
```
df_u_s_cols = ['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_u_d_cols = ['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_v_s_cols = ['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_v_d_cols = ['svd_v_d_1', 'svd_v_d_2', 'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5', 'svd_v_d_6']
```

In [64]:
```
df_train_u_s = df_final_train[df_u_s_cols]
df_train_u_d = df_final_train[df_u_d_cols]
df_train_v_s = df_final_train[df_v_s_cols]
df_train_v_d = df_final_train[df_v_d_cols]

df_test_u_s = df_final_test[df_u_s_cols]
df_test_u_d = df_final_test[df_u_d_cols]
df_test_v_s = df_final_test[df_v_s_cols]
df_test_v_d = df_final_test[df_v_d_cols]

df_train_u_s.columns = [1, 2, 3, 4, 5, 6]
df_train_u_d.columns = [1, 2, 3, 4, 5, 6]
df_train_v_s.columns = [1, 2, 3, 4, 5, 6]
df_train_v_d.columns = [1, 2, 3, 4, 5, 6]

df_test_u_s.columns = [1, 2, 3, 4, 5, 6]
df_test_u_d.columns = [1, 2, 3, 4, 5, 6]
df_test_v_s.columns = [1, 2, 3, 4, 5, 6]
df_test_v_d.columns = [1, 2, 3, 4, 5, 6]
```

In [54]:
```
df_train_u_s.shape
```

Out[54]:
```
(100002, 6)
```

In [65]:
```
df_train_u_s.iloc[0].dot(df_train_u_d.iloc[0])
```

Out[65]:
```
1.114957846286687e-11
```

In [66]:
```
np.array([df_train_u_s.iloc[i].dot(df_train_u_d.iloc[i]) for i in range(df_train_u_s.shape[0])]).shape
```

Out[66]:
```
(100002,)
```

In [67]:
```
df_final_train['svd_u_dot'] = \
    np.array([df_train_u_s.iloc[i].dot(df_train_u_d.iloc[i]) for i in range(df_train_u_s.shape[0])])
df_final_train['svd_v_dot'] = \
    np.array([df_train_v_s.iloc[i].dot(df_train_v_d.iloc[i]) for i in range(df_train_v_s.shape[0])])

df_final_test['svd_u_dot'] = \
    np.array([df_test_u_s.iloc[i].dot(df_test_u_d.iloc[i]) for i in range(df_test_u_s.shape[0])])
df_final_test['svd_v_dot'] = \
    np.array([df_test_v_s.iloc[i].dot(df_test_v_d.iloc[i]) for i in range(df_test_v_s.shape[0])])
```

In [68]:
```
df_final_train.columns
```

Out[68]:
```
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',
       'pref_attach_followers', 'pref_attach_followees', 'svd_u_dot',
       'svd_v_dot'],
      dtype='object')
```

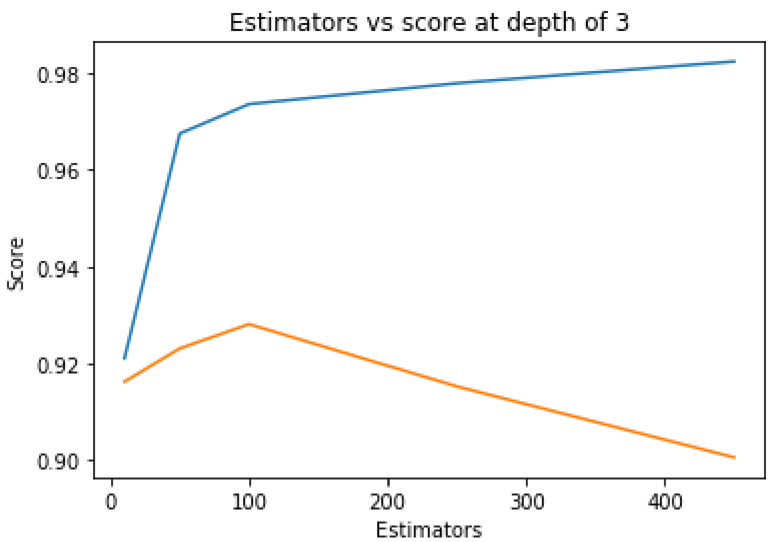
In [69]:
```
df_final_train.drop(['source_node', 'destination_node','indicator_link'],axis=1,inplace=True)
df_final_test.drop(['source_node', 'destination_node','indicator_link'],axis=1,inplace=True)
```

In [71]:
```
from xgboost import XGBClassifier
```

In [72]:
```
estimators = [10,50,100,250,450]
train_scores = []
test_scores = []
for i in estimators:
    clf = XGBClassifier(max_depth=3, n_estimators=i, n_jobs=-1)
    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 3')
```

```
Estimators =  10 Train Score 0.9210492696844526 test Score 0.9162413689582708
Estimators =  50 Train Score 0.9675354922332259 test Score 0.9231354642313546
Estimators =  100 Train Score 0.9736268857840504 test Score 0.9281594571670908
Estimators =  250 Train Score 0.9779501631263698 test Score 0.9152513258755074
Estimators =  450 Train Score 0.9824526059189717 test Score 0.9005723238961547
```

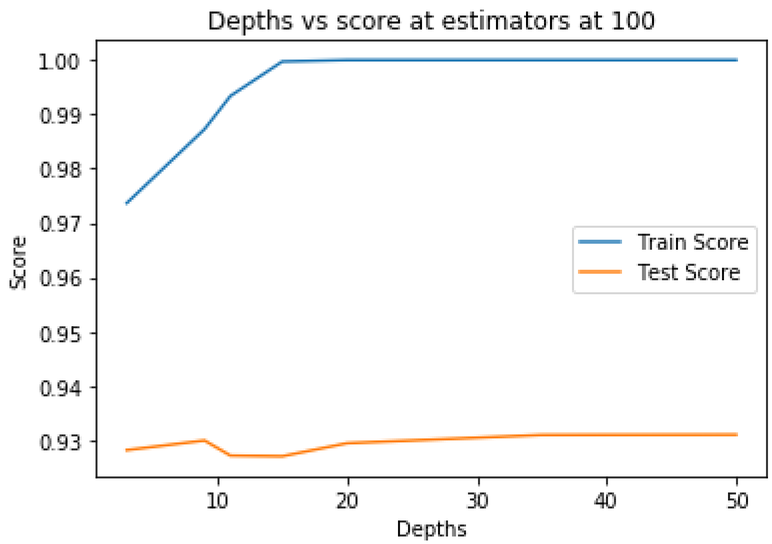
Out[72]:
```
Text(0.5, 1.0, 'Estimators vs score at depth of 3')
```



In [74]:
```
depths = [3,9,11,15,20,35,50]
train_scores = []
test_scores = []
for i in depths:
    clf = XGBClassifier(max_depth=i, n_estimators=100, n_jobs=-1)
    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('Depths')
plt.ylabel('Score')
plt.legend()
plt.title('Depths vs score at estimators at 100')
```

```
Depth =  3 Train Score 0.9736268857840504 test Score 0.9281594571670908
Depth =  9 Train Score 0.9872526234241848 test Score 0.9299300995551789
Depth =  11 Train Score 0.9933756222900273 test Score 0.9271430391032989
Depth =  15 Train Score 0.999730196956222 test Score 0.9270227841656413
Depth =  20 Train Score 1.0 test Score 0.9294266976940977
Depth =  35 Train Score 1.0 test Score 0.9309484544801049
Depth =  50 Train Score 1.0 test Score 0.9310024098423033
```

Out[74]:
```
Text(0.5, 1.0, 'Depths vs score at estimators at 100')
```



The difference between f1-scores is not as big as it seems and the model is overfitting at high depth values. And optimum depth is around 9.

In [81]:
```
param_dist = {"n_estimators":sp_randint(95,110),
              "max_depth": sp_randint(5,10),
              "learning_rate": uniform(0.08, 0.12)}

clf = XGBClassifier(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 [0.9809396  0.97851289 0.9786617  0.98067437 0.98089533]
mean train scores [0.99289803 0.98525409 0.98638145 0.99315784 0.99471193]
```

In [82]:
```
print(rf_random.best_estimator_)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.18441489639526543, max_delta_step=0, max_depth=7,
              min_child_weight=1, missing=None, n_estimators=108, 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)
```

In [83]:
```
clf = XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.18441489639526543, max_delta_step=0, max_depth=7,
              min_child_weight=1, missing=None, n_estimators=108, 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)
```

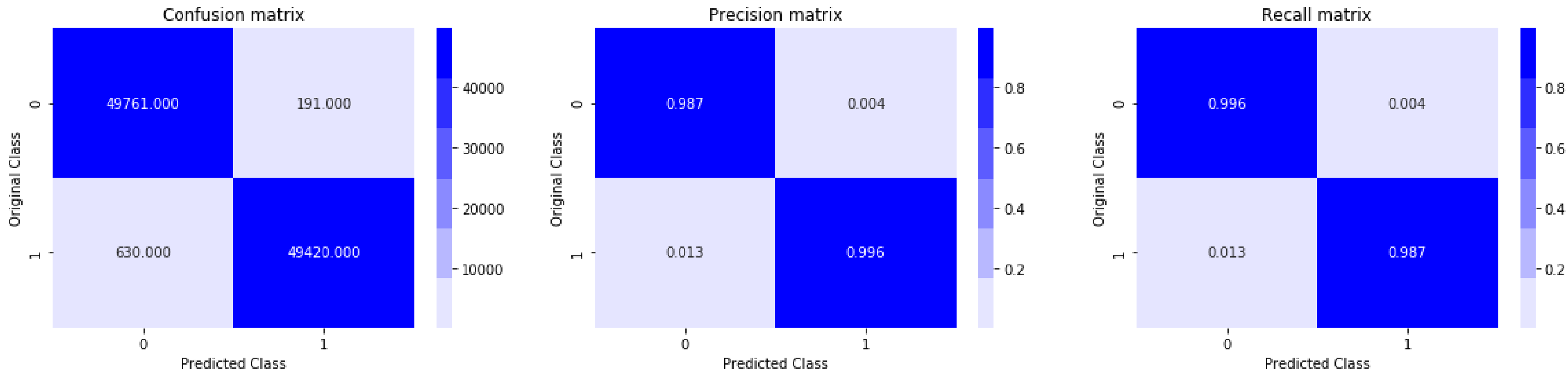
In [84]:
```
clf.fit(df_final_train,y_train)
y_train_pred = clf.predict(df_final_train)
y_test_pred = clf.predict(df_final_test)
```

In [85]:
```
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))
```

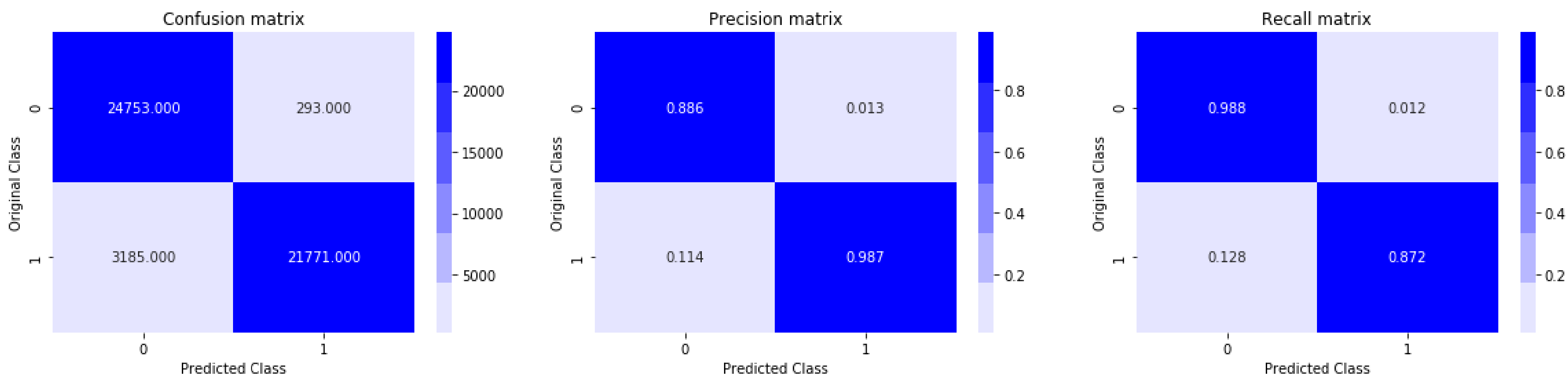
```
Train f1 score 0.9917620734289241
Test f1 score 0.9260314759676734
```

In [86]:
```
print('Train confusion_matrix')
plot_confusion_matrix(y_train,y_train_pred)
print('Test confusion_matrix')
plot_confusion_matrix(y_test,y_test_pred)
```

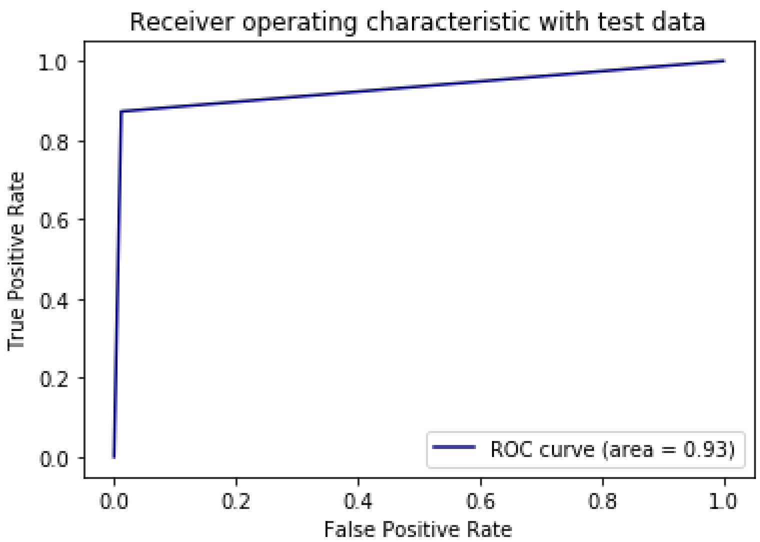
```
Train confusion_matrix
```




```
Test confusion_matrix
```



In [87]:
```
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 [92]:
```
sum(clf.feature_importances_)
```

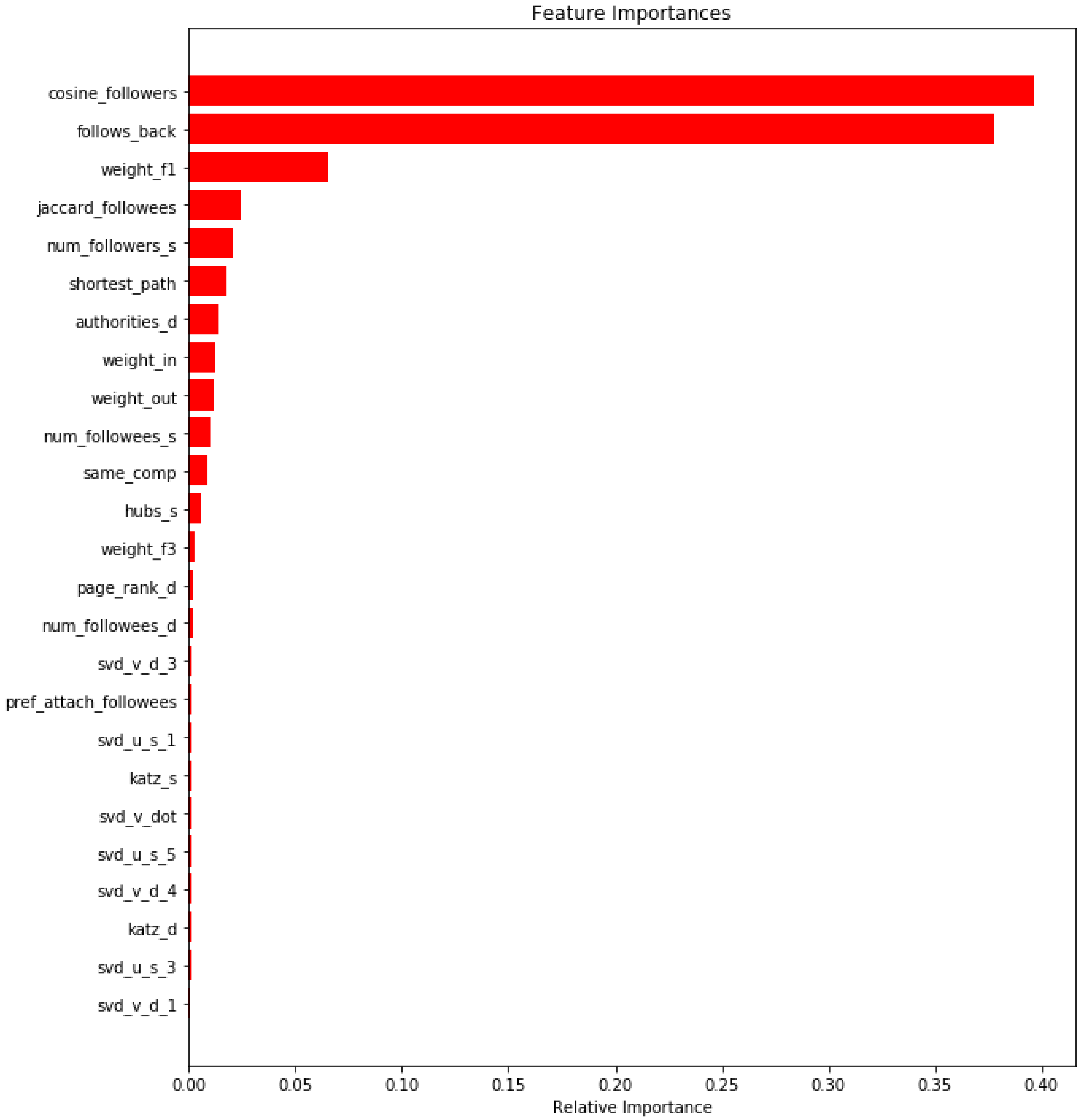
Out[92]:
```
0.9999999891151674
```

In [93]:
```
sum(clf.feature_importances_)
```

Out[93]:
```
0.9999999891151674
```

In [88]:

```
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
authorities_d
weight_in
weight_out
num_followees_s
same_comp
hubs_s
weight_f3
page_rank_d
num_followees_d
svd_v_d_3
pref_attach_followees
svd_u_s_1
katz_s
svd_v_dot
svd_u_s_5
svd_v_d_4
katz_d
svd_u_s_3
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:

Manually writing values into the PrettyTable as I didn't store the values into variables.

In [89]:
```
from prettytable import PrettyTable
```

In [90]:
```
table = PrettyTable()
table.field_names = ['Model', 'n_estimator', 'depth', 'other hyper-param', 'Train f1 Score', 'Test f1 score']
table.add_row(['Random Forest', 250, 5, 'min_samples_leaf=52, min_samples_split=120', 0.924, 0.919])
table.add_row(['Random Forest', 115, 11, 'min_samples_leaf=52, min_samples_split=120', 0.962, 0.925])
table.add_row(['Random Forest', 121, 14, 'min_samples_leaf=28, min_samples_split=111', 0.965, 0.924])
table.add_row(['GBDT', 100, 3, 'learning_rate=0.1', 0.974, 0.928])
table.add_row(['GBDT', 100, 9, 'learning_rate=0.1', 0.987, 0.93])
table.add_row(['GBDT', 108, 7, 'learning_rate=0.184', 0.992, 0.926])
print(table)
```

```
+---------------+-------------+-------+--------------------------------------------+----------------+---------
------+
|     Model     | n_estimator | depth |             other hyper-param              | Train f1 Score | Test f1
score |
+---------------+-------------+-------+--------------------------------------------+----------------+---------
------+
| Random Forest |     250     |   5   | min_samples_leaf=52, min_samples_split=120 |     0.924      |     0.91
9     |
| Random Forest |     115     |   11  | min_samples_leaf=52, min_samples_split=120 |     0.962      |     0.92
5     |
| Random Forest |     121     |   14  | min_samples_leaf=28, min_samples_split=111 |     0.965      |     0.92
4     |
|      GBDT     |     100     |   3   |             learning_rate=0.1              |     0.974      |     0.92
8     |
|      GBDT     |     100     |   9   |             learning_rate=0.1              |     0.987      |      0.9
3     |
|      GBDT     |     108     |   7   |            learning_rate=0.184             |     0.992      |     0.92
6     |
+---------------+-------------+-------+--------------------------------------------+----------------+---------
------+
```

Input data for GBDT includes extra 2 features where as input for Random Forest does not include the extra 2 features. (i.e. Preferential Attachment and SVD dot features)

Conclusion:

- **AUC values for both randomizedCV models (Random Forest and GBDT) are same (0.93).**
- **If we compare f1 score, GBDT is slightly better than Random Forest. May be due to the extra features or the model itself. And from train f1 score we can see that GBDT is overfitting to train data as well.**
- **By seeing feature importances, the new features that are added are not having good importances. And the same features which have high importances in Random Forest also have high importances in GBDT model (Top 3 features are same if order is not considered).**
- **In GBDT model `cosine_followers` and `follows_back` features have very high importance values when compared to other features. whereas in Random Forest model the importance is mostly distributed between several features. i.e. In GBDT both `cosine_followers` and `follows_back` features have around ~0.4+0.38 = ~0.78 out of 1 feature importance. And top 2 features in Random Forest have ~0.2+0.1 = ~0.3 out of 1 feature importance.**

In [ ]: