Social network Graph Link Prediction - Facebook Challenge

In [9]:
```
#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 [10]:
```
!wget --header="Host: doc-0o-bk-docs.googleusercontent.com" --header="User-Agent:
```

```
'wget' is not recognized as an internal or external command,
operable program or batch file.
```

In [11]:
```
#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')
```

In [12]:
```
df_final_train.columns
```

Out[12]:
```
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')
```

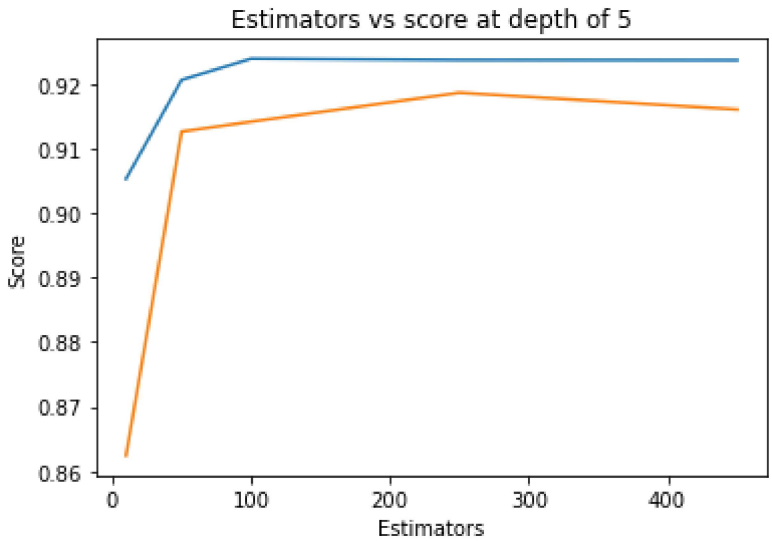
In [13]:
```
y_train = df_final_train.indicator_link
y_test = df_final_test.indicator_link
```

In [14]:
```
df_final_train.drop(['source_node', 'destination_node','indicator_link'],axis=1,i
df_final_test.drop(['source_node', 'destination_node','indicator_link'],axis=1,ir
```

In [15]:
```
estimators = [10,50,100,250,450]
train_scores = []
test_scores = []
for i in estimators:
    clf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gi
            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=
    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')
```

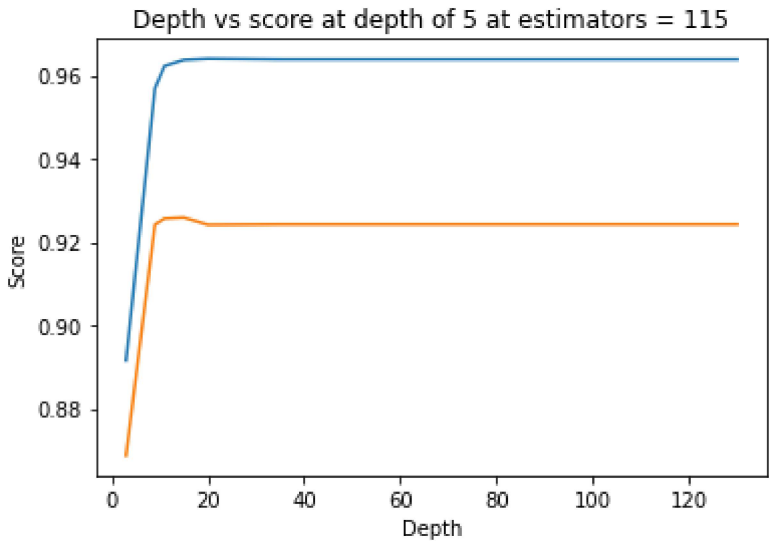
```
Estimators =  10 Train Score 0.9052856222201092 test Score 0.8624388531818281
Estimators =  50 Train Score 0.9205854226435491 test Score 0.9125880704982124
Estimators =  100 Train Score 0.9238690848446947 test Score 0.9141199714153599
Estimators =  250 Train Score 0.9236697170858376 test Score 0.9186132077455165
Estimators =  450 Train Score 0.9236384934787838 test Score 0.9160192846165182
```

Out[15]: Text(0.5, 1.0, 'Estimators vs score at depth of 5')



In [16]:
```
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='gi
            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_stat
    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 0.8687447491437064
depth =  9 Train Score 0.9569047254931946 test Score 0.9241811905363306
depth =  11 Train Score 0.9623061670649243 test Score 0.925683935421321
depth =  15 Train Score 0.9637427612296021 test Score 0.925901777263645
depth =  20 Train Score 0.964040453222158 test Score 0.9241664207730893
depth =  35 Train Score 0.9638798495585089 test Score 0.9242727445074043
depth =  50 Train Score 0.9638798495585089 test Score 0.9242727445074043
depth =  70 Train Score 0.9638798495585089 test Score 0.9242727445074043
depth =  130 Train Score 0.9638798495585089 test Score 0.9242727445074043
```



In [17]:
```
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),'

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

rf_random = RandomizedSearchCV(clf, param_distributions=param_dist,n_iter=5,cv=10

rf_random.fit(df_final_train,y_train)
```

Out[17]:
```
RandomizedSearchCV(cv=10,
                   estimator=RandomForestClassifier(n_jobs=-1, random_state=2
5),
                   n_iter=5,
                   param_distributions={'max_depth': <scipy.stats._distn_infras
tructure.rv_frozen object at 0x000002CC9E8D9AC0>,
                                        'min_samples_leaf': <scipy.stats._distn
_infrastructure.rv_frozen object at 0x000002CC9D8F1EE0>,
                                        'min_samples_split': <scipy.stats._dist
n_infrastructure.rv_frozen object at 0x000002CC9D8C0820>,
                                        'n_estimators': <scipy.stats._distn_inf
rastructure.rv_frozen object at 0x000002CC9E8D4940>},
                   random_state=25, scoring='f1')
```

In [18]:
```
rf_random.cv_results_
```

Out[18]:
```
{'mean_fit_time': array([4.87089736, 4.29577591, 3.74381237, 3.94011602, 4.5239
6684]),
 'std_fit_time': array([0.80717591, 0.36750693, 0.05326612, 0.07915271, 0.06209
38 ]),
 'mean_score_time': array([0.06900346, 0.05209553, 0.04212027, 0.04215767, 0.05
515032]),
 'std_score_time': array([0.02785443, 0.0177478 , 0.00262311, 0.0013712 , 0.026
47442]),
 'param_max_depth': masked_array(data=[14, 12, 11, 13, 14],
              mask=[False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_min_samples_leaf': masked_array(data=[51, 33, 56, 49, 28],
              mask=[False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_min_samples_split': masked_array(data=[125, 138, 179, 165, 111],
              mask=[False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_n_estimators': masked_array(data=[117, 109, 106, 108, 121],
              mask=[False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'max_depth': 14,
   'min_samples_leaf': 51,
   'min_samples_split': 125,
   'n_estimators': 117},
  {'max_depth': 12,
   'min_samples_leaf': 33,
   'min_samples_split': 138,
   'n_estimators': 109},
  {'max_depth': 11,
   'min_samples_leaf': 56,
   'min_samples_split': 179,
   'n_estimators': 106},
  {'max_depth': 13,
   'min_samples_leaf': 49,
   'min_samples_split': 165,
   'n_estimators': 108},
  {'max_depth': 14,
   'min_samples_leaf': 28,
   'min_samples_split': 111,
   'n_estimators': 121}],
 'split0_test_score': array([0.96242892, 0.96275246, 0.96070667, 0.96166329, 0.
96376002]),
 'split1_test_score': array([0.96315843, 0.96277513, 0.96118675, 0.9629855 , 0.
96327856]),
 'split2_test_score': array([0.96285049, 0.96303049, 0.96091767, 0.96214575, 0.
96556613]),
 'split3_test_score': array([0.96500809, 0.96483428, 0.96409427, 0.96494595, 0.
96517514]),
 'split4_test_score': array([0.9629029 , 0.96242892, 0.96174531, 0.96285772, 0.
96292545]),
 'split5_test_score': array([0.96026019, 0.95855819, 0.95979644, 0.95925211, 0.
```

```
9600488 ]),
 'split6_test_score': array([0.96273795, 0.96295547, 0.96083604, 0.96237704, 0.
96493006]),
 'split7_test_score': array([0.96318894, 0.9628801 , 0.96308349, 0.9628801 , 0.
963091  ]),
 'split8_test_score': array([0.96366958, 0.96182741, 0.96164773, 0.96252666, 0.
9647823 ]),
 'split9_test_score': array([0.95914022, 0.95803771, 0.95717475, 0.95902637, 0.
96143698]),
 'mean_test_score': array([0.96253457, 0.96200801, 0.96111891, 0.96206605, 0.96
349944]),
 'std_test_score': array([0.00158765, 0.0019927 , 0.00176025, 0.00167523, 0.001
65968]),
 'rank_test_score': array([2, 4, 5, 3, 1])}
```

In [19]:
```
print(rf_random.best_estimator_)
```

```
RandomForestClassifier(max_depth=14, min_samples_leaf=28, min_samples_split=11
1,
                       n_estimators=121, n_jobs=-1, random_state=25)
```

In [20]:
```
print(rf_random.best_score_)
```

```
0.9634994434948114
```

In [21]:
```
print(rf_random.cv_results_['params'])
```

```
[{'max_depth': 14, 'min_samples_leaf': 51, 'min_samples_split': 125, 'n_estimat
ors': 117}, {'max_depth': 12, 'min_samples_leaf': 33, 'min_samples_split': 138,
'n_estimators': 109}, {'max_depth': 11, 'min_samples_leaf': 56, 'min_samples_sp
lit': 179, 'n_estimators': 106}, {'max_depth': 13, 'min_samples_leaf': 49, 'min
_samples_split': 165, 'n_estimators': 108}, {'max_depth': 14, 'min_samples_lea
f': 28, 'min_samples_split': 111, 'n_estimators': 121}]
```

In [24]:
```
print(rf_random.cv_results_['mean_test_score'])
```

```
[0.96253457 0.96200801 0.96111891 0.96206605 0.96349944]
```

In [22]:
```
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 [23]:
```
clf.fit(df_final_train,y_train)
y_train_pred = clf.predict(df_final_train)
y_test_pred = clf.predict(df_final_test)
```

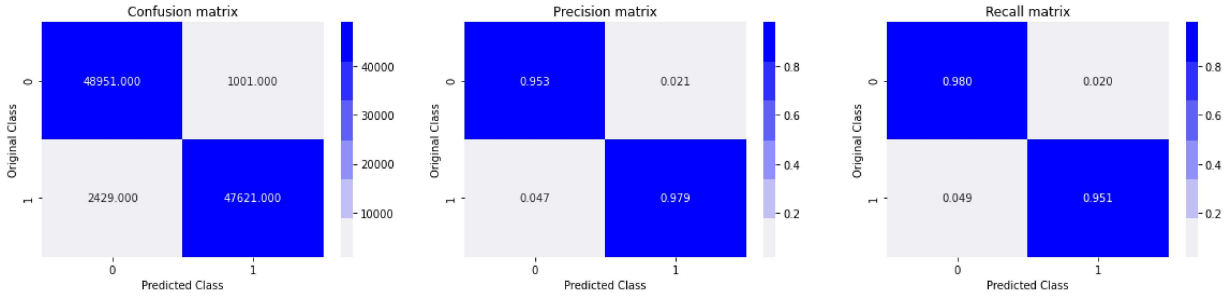
In [25]:
```
from sklearn.metrics import f1_score
print('Train f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))
```

```
Train f1 score 0.9652383654937572
Test f1 score 0.9244579303369367
```

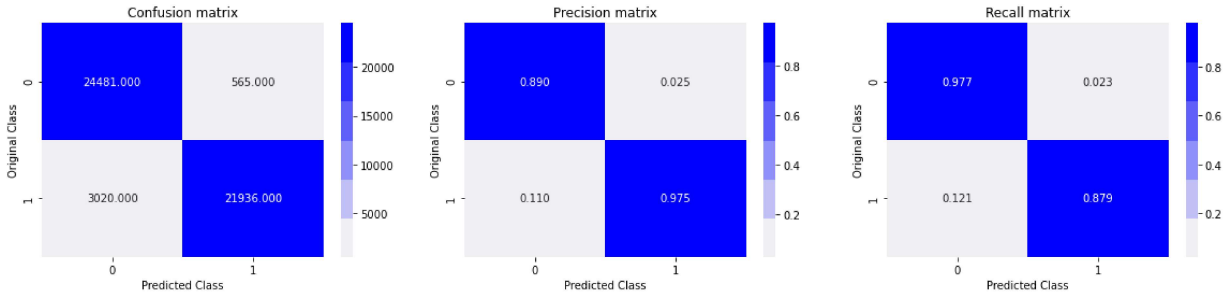
In [26]:
```
from sklearn.metrics import confusion_matrix
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)

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

    plt.show()
```

In [27]:
```
print('Train confusion_matrix')
plot_confusion_matrix(y_train,y_train_pred)
print('Test confusion_matrix')
plot_confusion_matrix(y_test,y_test_pred)
```

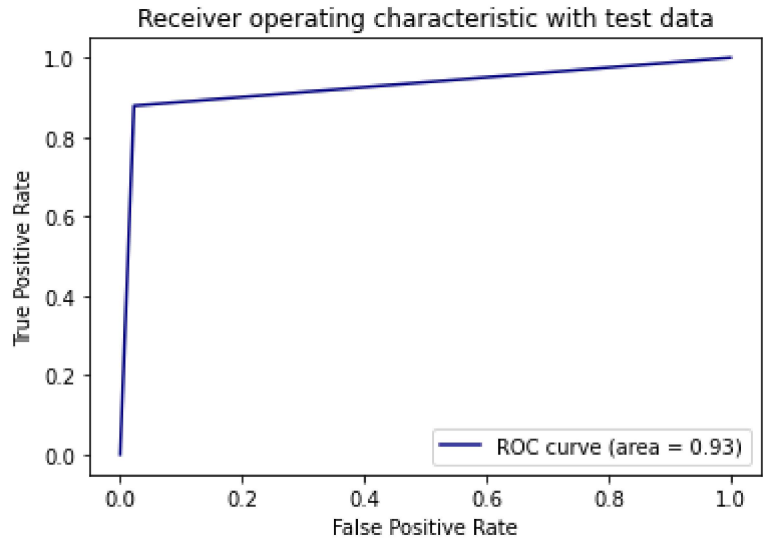
```
Train confusion_matrix
```



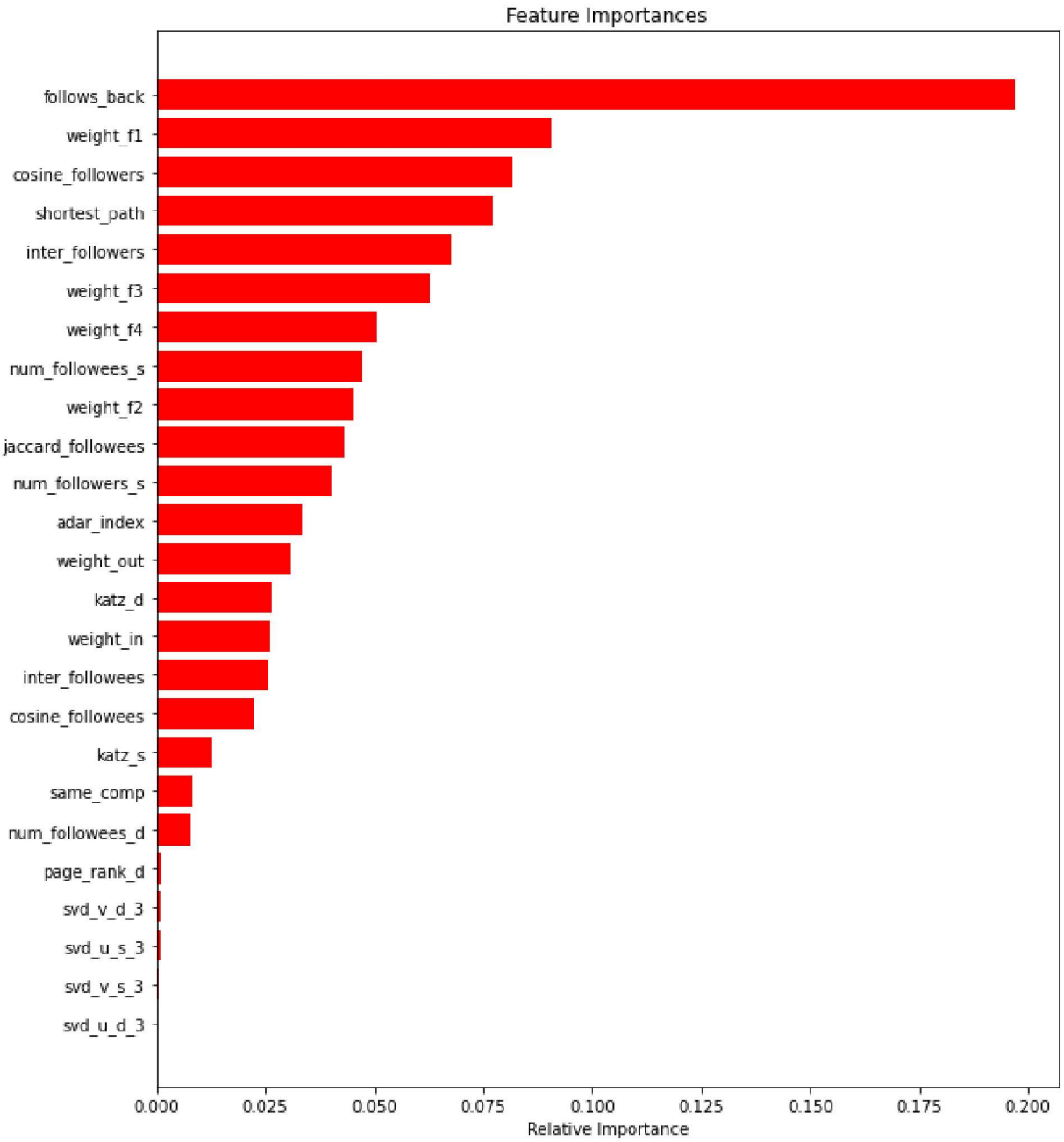

```
Test confusion_matrix
```



In [28]:
```
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 [29]:
```
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/ (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.

Preferential Attachment

In [30]:
```
#reaidng the files

from pandas import read_hdf
df_final_train = read_hdf('storage_sample_stage5.h5', 'train_df',mode='r')
df_final_test = read_hdf('storage_sample_stage5.h5', 'test_df',mode='r')
```

In [31]:
```
print(df_final_train.columns)
print('\n')
print(df_final_test.columns)
```

```
Index(['source_node', 'destination_node', 'indicator_link', 'num_followers_s',
       'num_followers_d', 'num_followees_s', 'num_followees_d',
       'inter_followers', 'inter_followees',
       'preferential_attachement_followers_train'],
      dtype='object')


Index(['source_node', 'destination_node', 'indicator_link', 'num_followers_s',
       'num_followers_d', 'num_followees_s', 'num_followees_d',
       'inter_followers', 'inter_followees',
       'preferential_attachement_followers_test'],
      dtype='object')
```

In [32]:
```
#taking y train and y test
y_train = df_final_train.indicator_link
y_test = df_final_test.indicator_link
```

In [33]:
```
df_final_train.drop(['source_node', 'destination_node','indicator_link'],axis=1,i
df_final_test.drop(['source_node', 'destination_node','indicator_link'],axis=1,ir
```

In [34]:
```
df_final_test.columns
```

Out[34]:
```
Index(['num_followers_s', 'num_followers_d', 'num_followees_s',
       'num_followees_d', 'inter_followers', 'inter_followees',
       'preferential_attachement_followers_test'],
      dtype='object')
```

In [35]:
```
df_final_train.columns
```

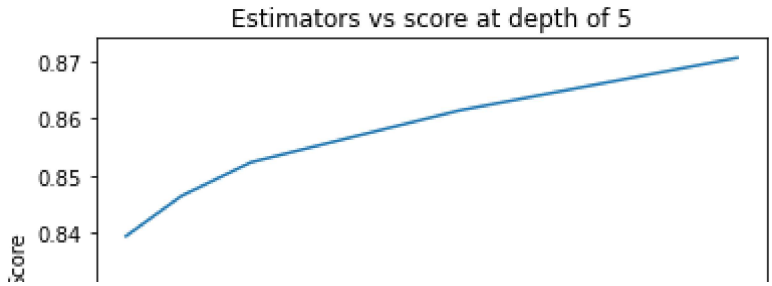
Out[35]:
```
Index(['num_followers_s', 'num_followers_d', 'num_followees_s',
       'num_followees_d', 'inter_followers', 'inter_followees',
       'preferential_attachement_followers_train'],
      dtype='object')
```

In [ ]:

In [36]:
```
from xgboost import XGBClassifier
estimators = [10,50,100,250,450]
train_scores = []
test_scores = []

for i in estimators:
    clf = XGBClassifier(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 5')
```

```
[10:05:13] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
Estimators =  10 Train Score 0.8395218608658195 test Score 0.809916626229653
[10:05:14] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
Estimators =  50 Train Score 0.8465515767883086 test Score 0.8086845690678153
[10:05:15] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
Estimators =  100 Train Score 0.852401280683031 test Score 0.8058810621022602
[10:05:17] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
Estimators =  250 Train Score 0.8614496472097498 test Score 0.8042114498793596
[10:05:23] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
Estimators =  450 Train Score 0.8706164537579373 test Score 0.8012457779532395
```

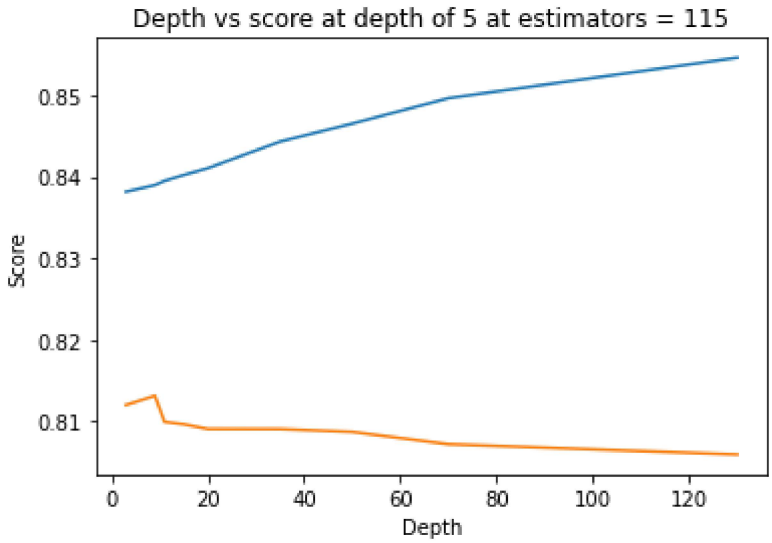
Out[36]: Text(0.5, 1.0, 'Estimators vs score at depth of 5')


Estimators vs score at depth of 5
0.87
0.86
0.85
0.84
core

In [37]:
```
depths = [3,9,11,15,20,35,50,70,130]
train_scores = []
test_scores = []
for i in depths:
    clf = XGBClassifier(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('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()
```

```
[10:05:36] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
depth =  3 Train Score 0.8381913020940203 test Score 0.8119936597393449
[10:05:36] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
depth =  9 Train Score 0.8389998705613325 test Score 0.8131248623651178
[10:05:37] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
depth =  11 Train Score 0.8395218608658195 test Score 0.809916626229653
[10:05:37] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
depth =  15 Train Score 0.8402388470047469 test Score 0.8096245372818616
[10:05:37] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
depth =  20 Train Score 0.8410739580203459 test Score 0.8090452261306533
[10:05:38] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
depth =  35 Train Score 0.8443503187438498 test Score 0.8090332805071315
[10:05:39] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
depth =  50 Train Score 0.8465515767883086 test Score 0.8086845690678153
[10:05:40] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

```
depth =  70 Train Score 0.8496780875302962 test Score 0.8071717349270522
[10:05:42] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
depth =  130 Train Score 0.8546492397972794 test Score 0.8059099478276119
```



In [38]:
```
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),'

clf = XGBClassifier(n_estimators=i)

rf_random = RandomizedSearchCV(clf, param_distributions=param_dist,n_iter=5,cv=10

rf_random.fit(df_final_train,y_train)
```

```
[10:08:22] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.
5.1/src/learner.cc:576:
Parameters: { "min_samples_leaf", "min_samples_split" } might not be used.

  This could be a false alarm, with some parameters getting used by language
bindings but
  then being mistakenly passed down to XGBoost core, or some parameter actual
ly being used
  but getting flagged wrongly here. Please open an issue if you find any such
cases.


[10:08:22] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.
5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation me
tric used with the objective 'binary:logistic' was changed from 'error' to 'l
ogloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.
[10:08:25] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.
5.1/src/learner.cc:576:
Parameters: { "min_samples_leaf", "min_samples_split" } might not be used.
```

In [39]:
```
rf_random.cv_results_
```

Out[39]:
```
{'mean_fit_time': array([4.87638958, 2.85621428, 2.54215028, 3.03675399, 3.6603
112 ]),
 'std_fit_time': array([0.90101082, 0.03010955, 0.02394823, 0.0392718 , 0.05482
75 ]),
 'mean_score_time': array([0.02572582, 0.01377563, 0.01566861, 0.01603422, 0.01
689305]),
 'std_score_time': array([0.00700844, 0.0038944 , 0.00420939, 0.00111674, 0.004
8357 ]),
 'param_max_depth': masked_array(data=[14, 12, 11, 13, 14],
              mask=[False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_min_samples_leaf': masked_array(data=[51, 33, 56, 49, 28],
              mask=[False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_min_samples_split': masked_array(data=[125, 138, 179, 165, 111],
              mask=[False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_n_estimators': masked_array(data=[117, 109, 106, 108, 121],
              mask=[False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'max_depth': 14,
   'min_samples_leaf': 51,
   'min_samples_split': 125,
   'n_estimators': 117},
  {'max_depth': 12,
   'min_samples_leaf': 33,
   'min_samples_split': 138,
   'n_estimators': 109},
  {'max_depth': 11,
   'min_samples_leaf': 56,
   'min_samples_split': 179,
   'n_estimators': 106},
  {'max_depth': 13,
   'min_samples_leaf': 49,
   'min_samples_split': 165,
   'n_estimators': 108},
  {'max_depth': 14,
   'min_samples_leaf': 28,
   'min_samples_split': 111,
   'n_estimators': 121}],
 'split0_test_score': array([0.83150144, 0.83370762, 0.8339381 , 0.83255119, 0.
83116329]),
 'split1_test_score': array([0.83459206, 0.83785221, 0.8372885 , 0.83602722, 0.
83459206]),
 'split2_test_score': array([0.82915427, 0.83283518, 0.8352916 , 0.83239091, 0.
82899352]),
 'split3_test_score': array([0.83639063, 0.83991065, 0.84105397, 0.83903783, 0.
83654865]),
 'split4_test_score': array([0.83560342, 0.83814968, 0.83860726, 0.83598895, 0.
83535748]),
 'split5_test_score': array([0.83471426, 0.83523029, 0.83646456, 0.83581455, 0.
```

```
83471426]),
 'split6_test_score': array([0.82973519, 0.83172974, 0.83411815, 0.83118543, 0.
82934609]),
 'split7_test_score': array([0.83384583, 0.83802668, 0.83526534, 0.83561789, 0.
83403985]),
 'split8_test_score': array([0.83305085, 0.83363453, 0.83611111, 0.83666525, 0.
83229156]),
 'split9_test_score': array([0.83555838, 0.83787783, 0.83695652, 0.83506686, 0.
83531157]),
 'mean_test_score': array([0.83341463, 0.83589544, 0.83650951, 0.83503461, 0.83
323583]),
 'std_test_score': array([0.00238717, 0.00265638, 0.00203426, 0.00222493, 0.002
50541]),
 'rank_test_score': array([4, 2, 1, 3, 5])}
```

In [40]:
```
print(rf_random.best_estimator_)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
              gamma=0, gpu_id=-1, importance_type=None,
              interaction_constraints='', learning_rate=0.300000012,
              max_delta_step=0, max_depth=11, min_child_weight=1,
              min_samples_leaf=56, min_samples_split=179, missing=nan,
              monotone_constraints='()', n_estimators=106, n_jobs=8,
              num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [41]:
```
print(rf_random.best_score_)
```

```
0.8365095119273794
```

In [43]:
```
print(rf_random.best_params_['max_depth'])
```

```
11
```

In [44]:
```
clf = XGBClassifier(n_estimators=i)
```

In [45]:
```
clf.fit(df_final_train,y_train)
y_train_pred = clf.predict(df_final_train)
y_test_pred = clf.predict(df_final_test)
```

```
[10:14:01] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

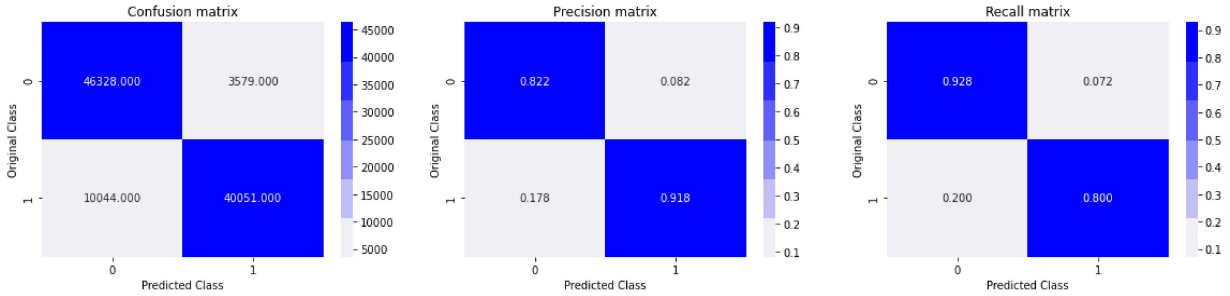
In [46]:
```
from sklearn.metrics import f1_score
print('Train f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))
```

```
Train f1 score 0.8546492397972794
Test f1 score 0.8059099478276119
```

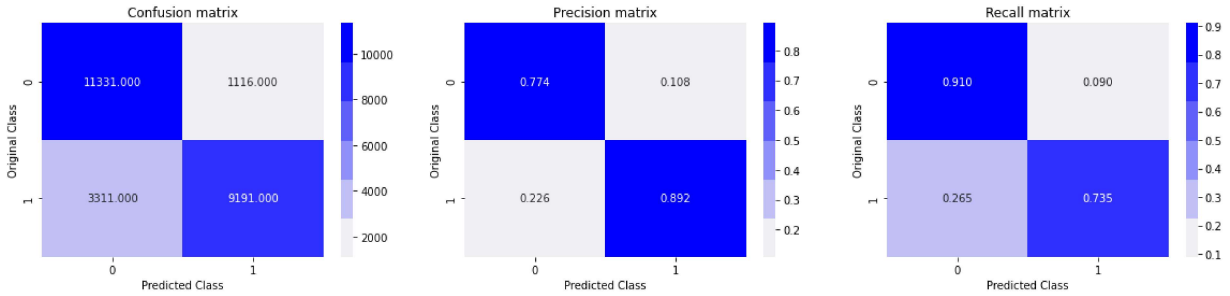
In [47]:
```
from sklearn.metrics import confusion_matrix
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)

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

    plt.show()
```

In [48]:
```
print('Train confusion_matrix')
plot_confusion_matrix(y_train,y_train_pred)
print('Test confusion_matrix')
plot_confusion_matrix(y_test,y_test_pred)
```

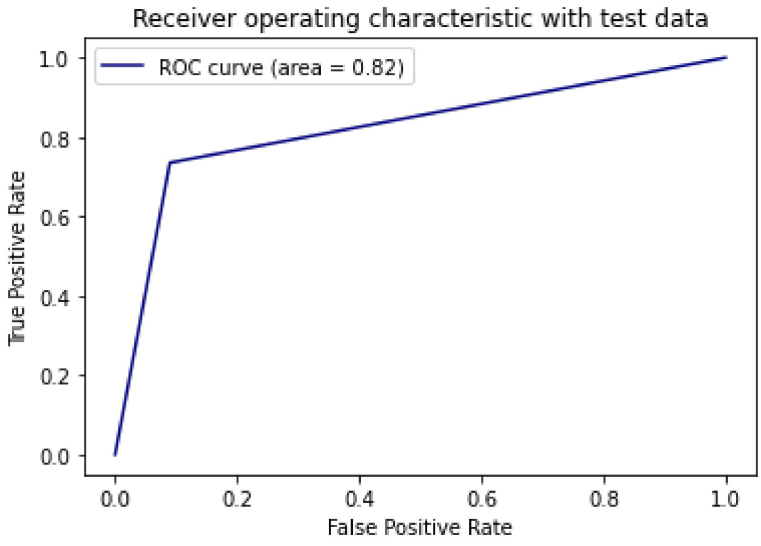
```
Train confusion_matrix
```



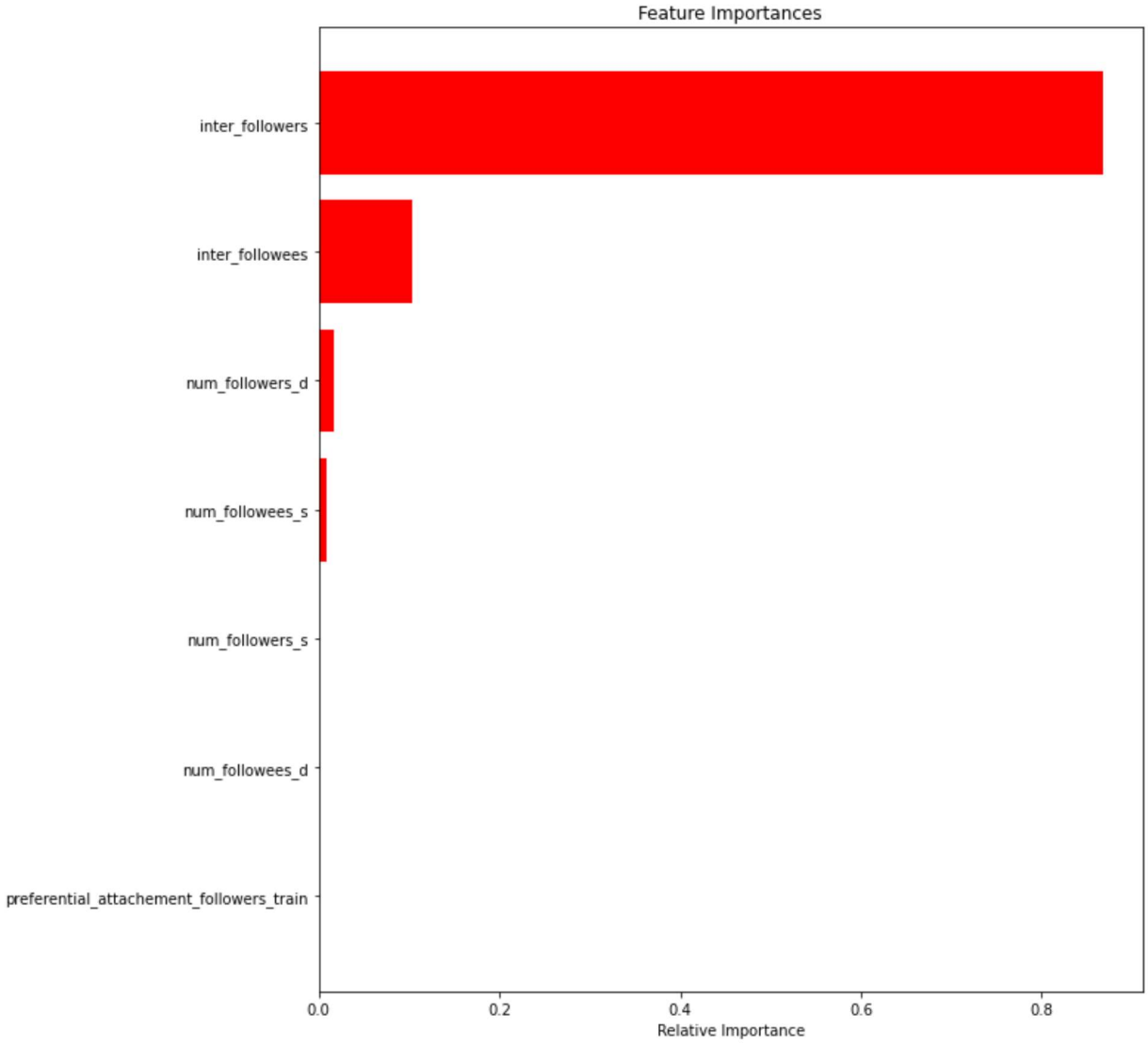

```
Test confusion_matrix
```



In [49]:
```
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 [50]:
```
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()
```



Observation :

1. preferental attachment feature is not that important when it comes to predict links.
2. Train f1 score 0.8546492397972794
3. Test f1 score 0.8059099478276119

SV_DOT feature

In [51]:
```
from pandas import read_hdf
df_final_train = read_hdf('storage_sample_stage6.h5', 'train_df',mode='r')
df_final_test = read_hdf('storage_sample_stage6.h5', 'test_df',mode='r')
```

In [39]:
```
print(df_final_train.columns)
print('\n')
print(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',
       'svd_dot_train'],
      dtype='object')


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

In [40]:
```
y_train = df_final_train.indicator_link
y_test = df_final_test.indicator_link
```

In [41]:
```
df_final_train.drop(['source_node', 'destination_node','indicator_link'],axis=1,i
df_final_test.drop(['source_node', 'destination_node','indicator_link'],axis=1,ir
```

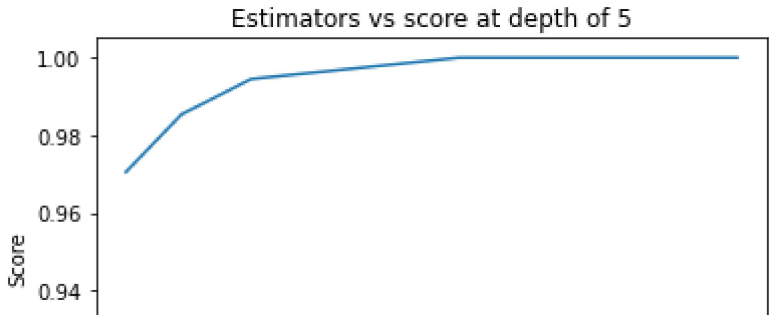
In [42]:
```
print(df_final_test.columns)

print('\n')

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',
       'svd_dot_test'],
      dtype='object')


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

In [43]:
```
from xgboost import XGBClassifier
estimators = [10,50,100,250,450]
train_scores = []
test_scores = []

for i in estimators:
    clf = XGBClassifier(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 5')
```

```
[20:14:06] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
Estimators =  10 Train Score 0.9705669604197443 test Score 0.9306724916760335
[20:14:10] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
Estimators =  50 Train Score 0.9854176940030392 test Score 0.9286698463169052
[20:14:27] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
Estimators =  100 Train Score 0.9945013671464198 test Score 0.9213233724653149
[20:15:03] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
Estimators =  250 Train Score 1.0 test Score 0.9050829088544489
[20:16:32] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
Estimators =  450 Train Score 1.0 test Score 0.8978167462758184
```

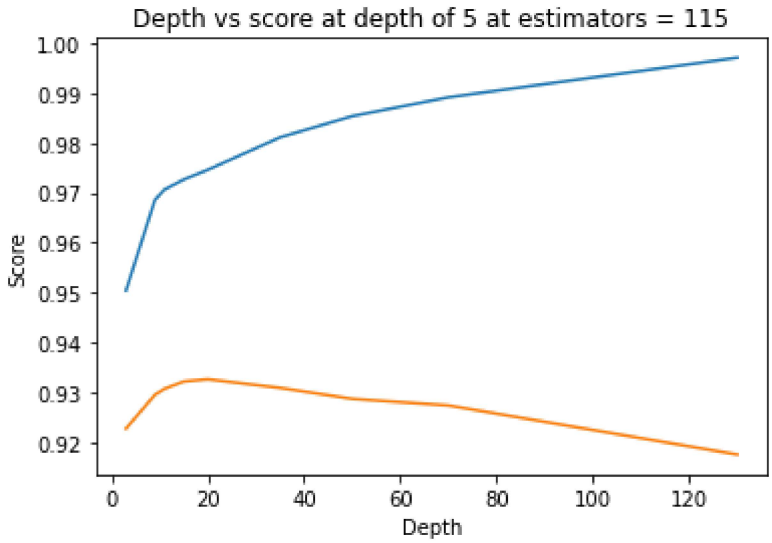
Out[43]: Text(0.5, 1.0, 'Estimators vs score at depth of 5')


Estimators vs score at depth of 5
1.00
0.98
0.96
0.94
Score

In [44]:
```
depths = [3,9,11,15,20,35,50,70,130]
train_scores = []
test_scores = []
for i in depths:
    clf = XGBClassifier(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('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()
```

```
[20:19:40] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
depth =  3 Train Score 0.9504458020308759 test Score 0.9227147401908802
[20:19:41] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
depth =  9 Train Score 0.9686050187105523 test Score 0.9294801119497923
[20:19:45] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
depth =  11 Train Score 0.9706843042482829 test Score 0.9306867771046053
[20:19:49] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
depth =  15 Train Score 0.9726994706521849 test Score 0.9321563895768507
[20:19:54] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
depth =  20 Train Score 0.9746537705050996 test Score 0.9325944882722447
[20:20:01] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
depth =  35 Train Score 0.9811602388252381 test Score 0.9308673847260723
[20:20:15] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
depth =  50 Train Score 0.9854176940030392 test Score 0.9286698463169052
[20:20:33] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

```
depth =  70 Train Score 0.9892034669425224 test Score 0.9273399014778324
[20:21:00] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
depth =  130 Train Score 0.9971490021507528 test Score 0.9174968399854317
```



In [45]:
```
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),'

clf = XGBClassifier(n_estimators=i)

rf_random = RandomizedSearchCV(clf, param_distributions=param_dist,n_iter=5,cv=10

rf_random.fit(df_final_train,y_train)
```

```
Parameters: { "min_samples_leaf", "min_samples_split" } might not be used.

  This could be a false alarm, with some parameters getting used by language
bindings but
  then being mistakenly passed down to XGBoost core, or some parameter actual
ly being used
  but getting flagged wrongly here. Please open an issue if you find any such
cases.


[20:24:57] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.
5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation me
tric used with the objective 'binary:logistic' was changed from 'error' to 'l
ogloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.
[20:25:27] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.
5.1/src/learner.cc:576:
Parameters: { "min_samples_leaf", "min_samples_split" } might not be used.

  This could be a false alarm, with some parameters getting used by language
```

In [46]:
```
rf_random.cv_results_
```

Out[46]:
```
{'mean_fit_time': array([37.67240968, 27.70712278, 26.15834789, 28.17588127,
2.03189301]),
 'std_fit_time': array([13.3341547 ,  0.37190748,  0.29388517,  0.21514046,  1.
03132046]),
 'mean_score_time': array([0.0241519 , 0.01944757, 0.01775222, 0.01834438, 0.01
924756]),
 'std_score_time': array([0.01134821, 0.00360268, 0.00177279, 0.0004812 , 0.000
8978 ]),
 'param_max_depth': masked_array(data=[14, 12, 11, 13, 14],
              mask=[False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_min_samples_leaf': masked_array(data=[51, 33, 56, 49, 28],
              mask=[False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_min_samples_split': masked_array(data=[125, 138, 179, 165, 111],
              mask=[False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_n_estimators': masked_array(data=[117, 109, 106, 108, 121],
              mask=[False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'max_depth': 14,
   'min_samples_leaf': 51,
   'min_samples_split': 125,
   'n_estimators': 117},
  {'max_depth': 12,
   'min_samples_leaf': 33,
   'min_samples_split': 138,
   'n_estimators': 109},
  {'max_depth': 11,
   'min_samples_leaf': 56,
   'min_samples_split': 179,
   'n_estimators': 106},
  {'max_depth': 13,
   'min_samples_leaf': 49,
   'min_samples_split': 165,
   'n_estimators': 108},
  {'max_depth': 14,
   'min_samples_leaf': 28,
   'min_samples_split': 111,
   'n_estimators': 121}],
 'split0_test_score': array([0.98242795, 0.98325143, 0.98255814, 0.9811245 , 0.
98253012]),
 'split1_test_score': array([0.98080595, 0.98121924, 0.9814313 , 0.98182183, 0.
98121169]),
 'split2_test_score': array([0.98424486, 0.982942  , 0.98524541, 0.98474814, 0.
98404095]),
 'split3_test_score': array([0.98154834, 0.9824491 , 0.98254414, 0.98194584, 0.
98144247]),
 'split4_test_score': array([0.98151125, 0.98061276, 0.98021492, 0.9809122 , 0.
98171222]),
 'split5_test_score': array([0.98026183, 0.98048682, 0.97978071, 0.97865055, 0.
```

```
97996174]),
 'split6_test_score': array([0.98332999, 0.98293858, 0.98284683, 0.98314269, 0.
98322451]),
 'split7_test_score': array([0.98290082, 0.98309519, 0.98260432, 0.98189863, 0.
9829077 ]),
 'split8_test_score': array([0.98124561, 0.98234704, 0.9819567 , 0.98083292, 0.
98134777]),
 'split9_test_score': array([0.981026  , 0.98082907, 0.98253363, 0.9809122 , 0.
9811245 ]),
 'mean_test_score': array([0.98193026, 0.98201712, 0.98217161, 0.98159895, 0.98
195037]),
 'std_test_score': array([0.00119004, 0.00105109, 0.00143701, 0.00151686, 0.001
14485]),
 'rank_test_score': array([4, 2, 1, 5, 3])}
```

In [47]:
```
print(rf_random.best_estimator_)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
              gamma=0, gpu_id=-1, importance_type=None,
              interaction_constraints='', learning_rate=0.300000012,
              max_delta_step=0, max_depth=11, min_child_weight=1,
              min_samples_leaf=56, min_samples_split=179, missing=nan,
              monotone_constraints='()', n_estimators=106, n_jobs=8,
              num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [48]:
```
clf = XGBClassifier(n_estimators=i)
```

In [49]:
```
clf.fit(df_final_train,y_train)
y_train_pred = clf.predict(df_final_train)
y_test_pred = clf.predict(df_final_test)
```

```
[20:48:58] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

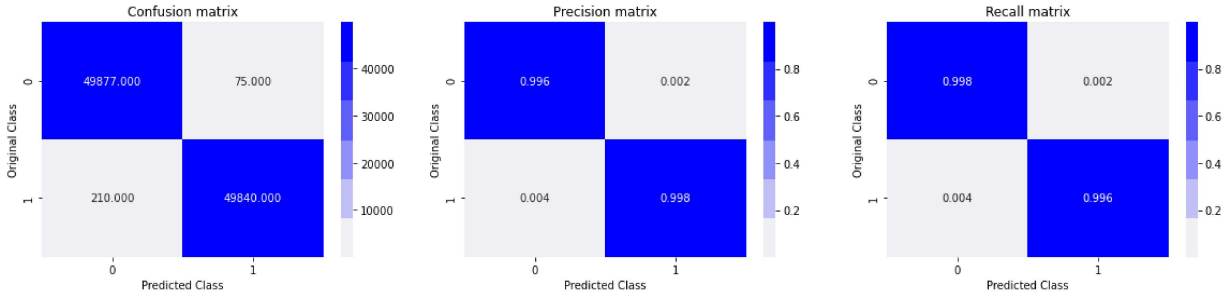
In [50]:
```
from sklearn.metrics import f1_score
print('Train f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))
```

```
Train f1 score 0.9971490021507528
Test f1 score 0.9174968399854317
```

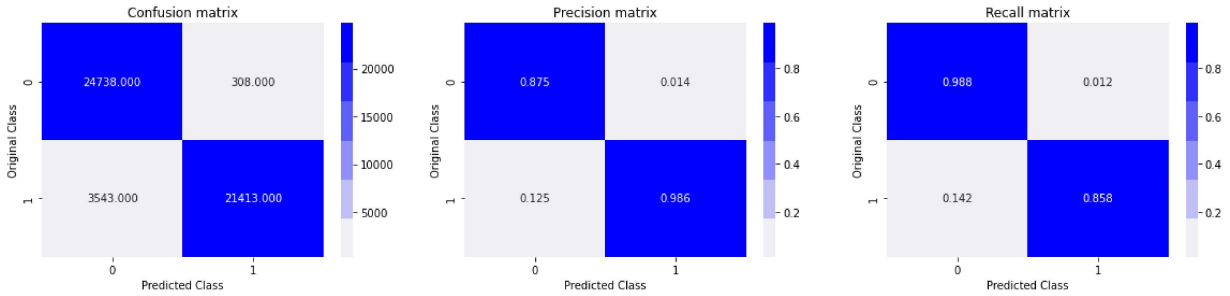
In [51]:
```
from sklearn.metrics import confusion_matrix
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)

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

    plt.show()
```

In [52]:
```
print('Train confusion_matrix')
plot_confusion_matrix(y_train,y_train_pred)
print('Test confusion_matrix')
plot_confusion_matrix(y_test,y_test_pred)
```

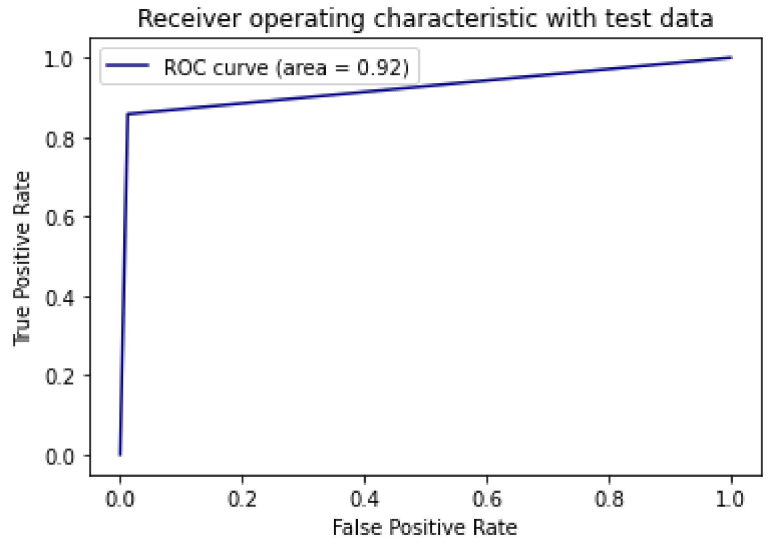
```
Train confusion_matrix
```



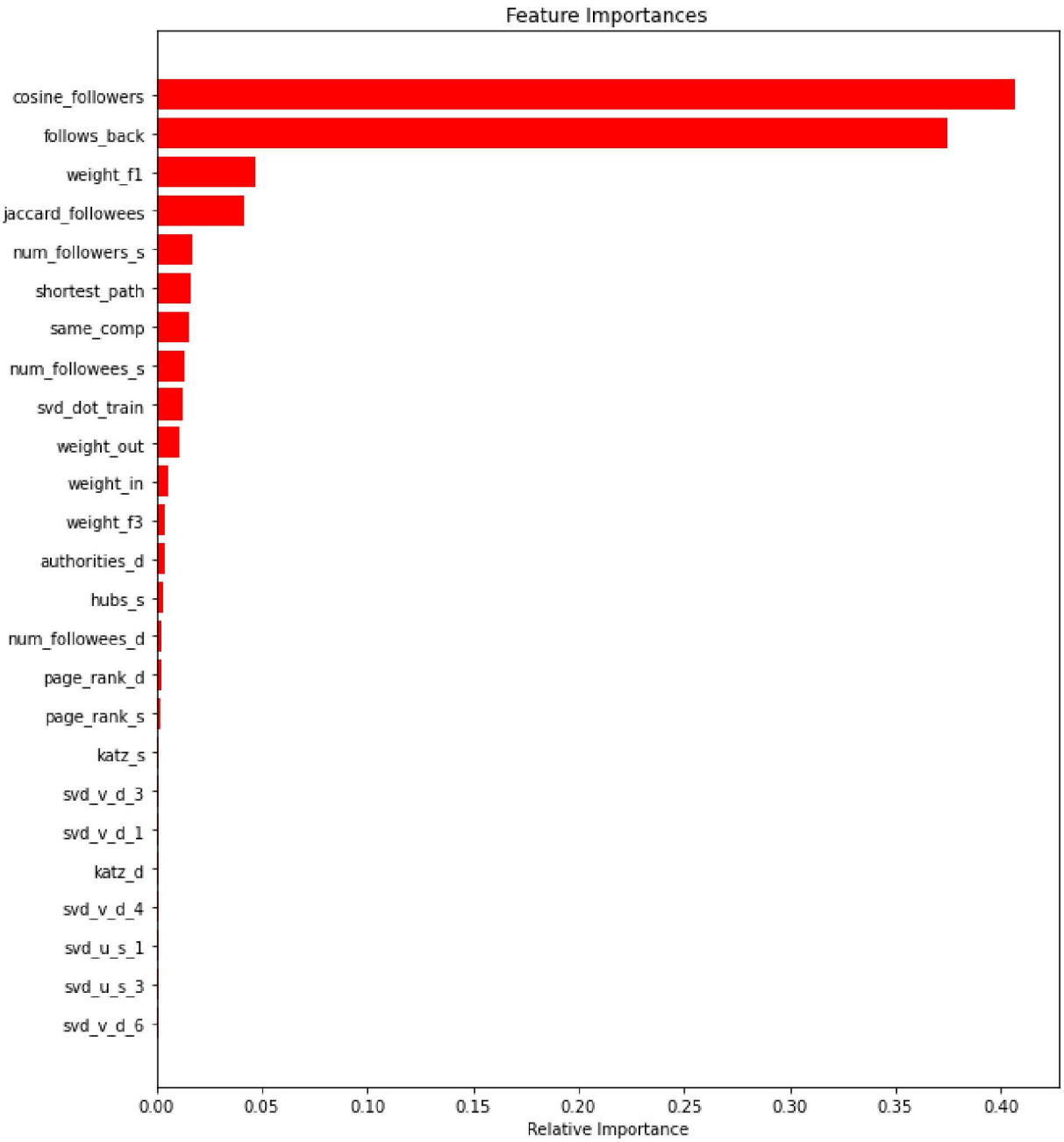

```
Test confusion_matrix
```



In [53]:
```
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 [54]:
```
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()
```



Observation:

1.It can be seen that svd_dot_train feature is of minimal importance.

2.Train f1 score 0.9971490021507528.

3.Test f1 score 0.9174968399854317.

In [ ]: