Social network Graph Link Prediction - Facebook Challenge

In [8]:

```
#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 [9]:

```
#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 [10]:

```
df_final_train.columns
```

Out[10]:

```
Index(['source_node', 'destination_node', 'indicator_link',
       'jaccard_followers', 'jaccard_followees', 'cosine_followers',
       'cosine_followees', 'num_followers_s', 'num_followers_d',
       '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 [11]:

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

In [31]:

```
df_final_train.head()
```

Out[31]:

	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	num_followers_s	nu
0	0	0.000000	0.000000	0.000000	11	
1	0	0.250000	0.236228	0.400000	7	
2	0	0.096774	0.026307	0.181568	20	
3	0	0.055556	0.047619	0.129099	4	
4	0	0.000000	0.000000	0.000000	1	

5 rows × 52 columns

In [13]:

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

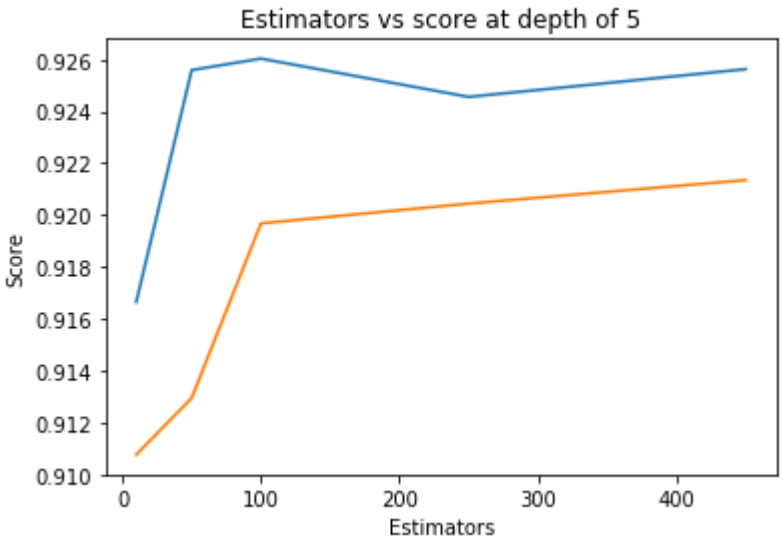
In [14]:

```
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
    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 0.9166589584682269 test Score 0.910774235252413
7
Estimators =  50 Train Score 0.9255901399623983 test Score 0.912958967896277
Estimators =  100 Train Score 0.9260323197292413 test Score 0.91968383477817
44
Estimators =  250 Train Score 0.9245566764834889 test Score 0.92044587607018
74
Estimators =  450 Train Score 0.9256257050933858 test Score 0.92135117368900
94
```

Out[14]:

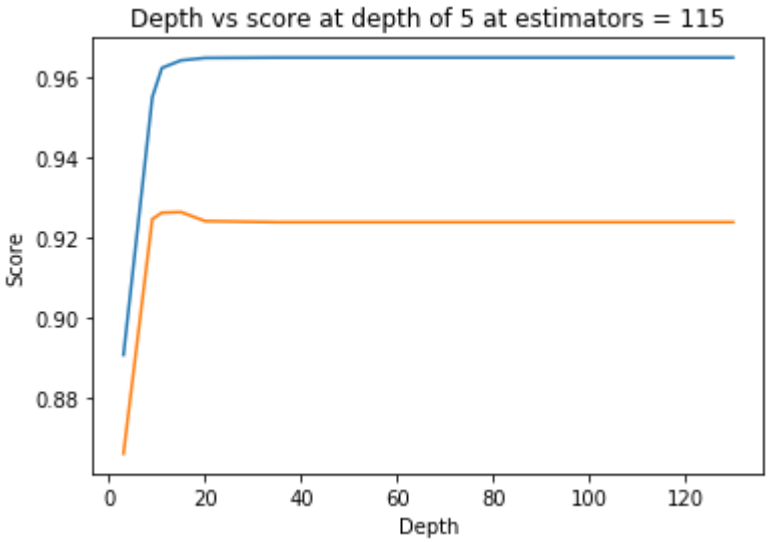
```
Text(0.5, 1.0, 'Estimators vs score at depth of 5')
```



In [15]:

```
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,verbo
    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 0.8906561474934344 test Score 0.865932727035377
depth =  9 Train Score 0.9551662495530925 test Score 0.9246653919694072
depth =  11 Train Score 0.9624994913116022 test Score 0.9262638507837259
depth =  15 Train Score 0.9643945860260944 test Score 0.9264133693364234
depth =  20 Train Score 0.965000354965974 test Score 0.9241576023329573
depth =  35 Train Score 0.9650884453099643 test Score 0.9238954626713204
depth =  50 Train Score 0.9650884453099643 test Score 0.9238954626713204
depth =  70 Train Score 0.9650884453099643 test Score 0.9238954626713204
depth =  130 Train Score 0.9650884453099643 test Score 0.9238954626713204
```



In [16]:

```
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)

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 [0.96307145 0.96274034 0.96053943 0.96236161 0.96450201]
```

In [17]:

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

```
{'split8_test_score': array([0.96216436, 0.96257012, 0.96013052, 0.96189894,
0.96386033]), 'std_score_time': array([0.00184693, 0.00015177, 0.00015997,
0.00021206, 0.00038525]), 'param_max_depth': masked_array(data=[14, 12, 11,
13, 14],
             mask=[False, False, False, False, False],
       fill_value='?',
            dtype=object), 'split0_test_score': array([0.95921066, 0.9600812
6, 0.95722849, 0.95900142, 0.96202275]), 'split3_test_score': array([0.96383
584, 0.96306069, 0.96063473, 0.96172977, 0.96497106]), 'std_fit_time': array
([0.31603506, 0.12260708, 0.08819669, 0.03834582, 0.13443847]), 'mean_fit_ti
me': array([9.04617162, 8.42024336, 7.80498815, 8.23532348, 9.69111624]), 's
plit2_test_score': array([0.96217975, 0.96369168, 0.95907147, 0.96357179, 0.
96678145]), 'mean_score_time': array([0.10955036, 0.10863428, 0.10874414, 0.
10882597, 0.10869744]), 'param_min_samples_split': masked_array(data=[125, 1
38, 179, 165, 111],
             mask=[False, False, False, False, False],
       fill_value='?',
            dtype=object), 'split6_test_score': array([0.96589637, 0.9647321
9, 0.96283474, 0.96506773, 0.96655484]), 'param_min_samples_leaf': masked_ar
ray(data=[51, 33, 56, 49, 28],
             mask=[False, False, False, False, False],
       fill_value='?',
            dtype=object), 'split4_test_score': array([0.96316591, 0.9631957
8, 0.96059313, 0.96156576, 0.96429296]), 'split1_test_score': array([0.96432
564, 0.96373268, 0.96158537, 0.96316646, 0.965412  ]), 'mean_test_score': ar
ray([0.96307145, 0.96274034, 0.96053943, 0.96236161, 0.96450201]), 'split5_t
est_score': array([0.96012208, 0.96003254, 0.95890968, 0.96074044, 0.9615345
6]), 'split9_test_score': array([0.96327942, 0.96180096, 0.95950346, 0.96153
846, 0.9628801 ]), 'split7_test_score': array([0.96653523, 0.96450602, 0.964
90341, 0.96533603, 0.96671051]), 'std_test_score': array([0.00217743, 0.0015
6844, 0.00205995, 0.00184736, 0.00182687]), 'param_n_estimators': masked_arr
ay(data=[117, 109, 106, 108, 121],
             mask=[False, False, False, False, False],
       fill_value='?',
            dtype=object), 'rank_test_score': array([2, 3, 5, 4, 1], dtype=i
nt32), 'params': [{'n_estimators': 117, 'max_depth': 14, 'min_samples_spli
t': 125, 'min_samples_leaf': 51}, {'n_estimators': 109, 'max_depth': 12, 'mi
n_samples_split': 138, 'min_samples_leaf': 33}, {'n_estimators': 106, 'max_d
epth': 11, 'min_samples_split': 179, 'min_samples_leaf': 56}, {'n_estimator
s': 108, 'max_depth': 13, 'min_samples_split': 165, 'min_samples_leaf': 49},
{'n_estimators': 121, 'max_depth': 14, 'min_samples_split': 111, 'min_sample
s_leaf': 28}]}
```

In [18]:

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

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=14, max_features='auto', max_leaf_nodes=Non
e,
                       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 [19]:

```
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 [20]:

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

In [21]:

```
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 0.9654108476778174
Test f1 score 0.9276322776609528
```

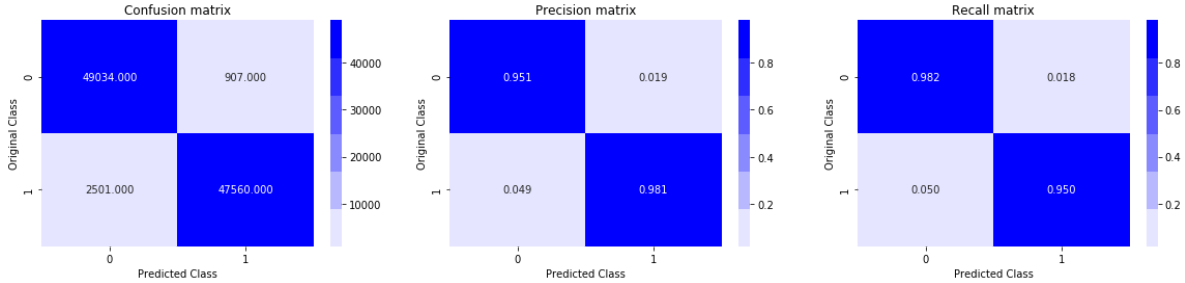
In [22]:

```
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()
```

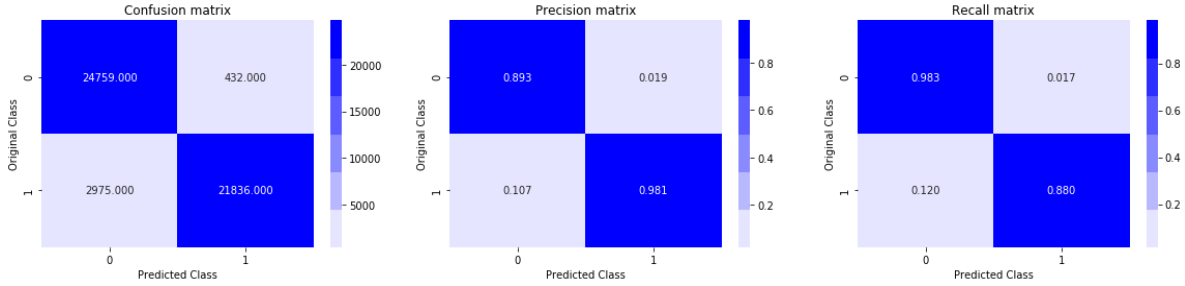
In [23]:

```
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
```

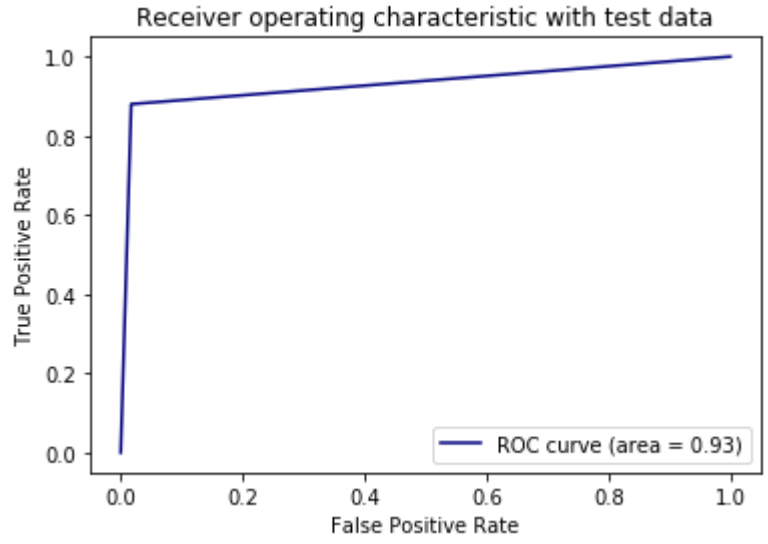



```
Test confusion_matrix
```



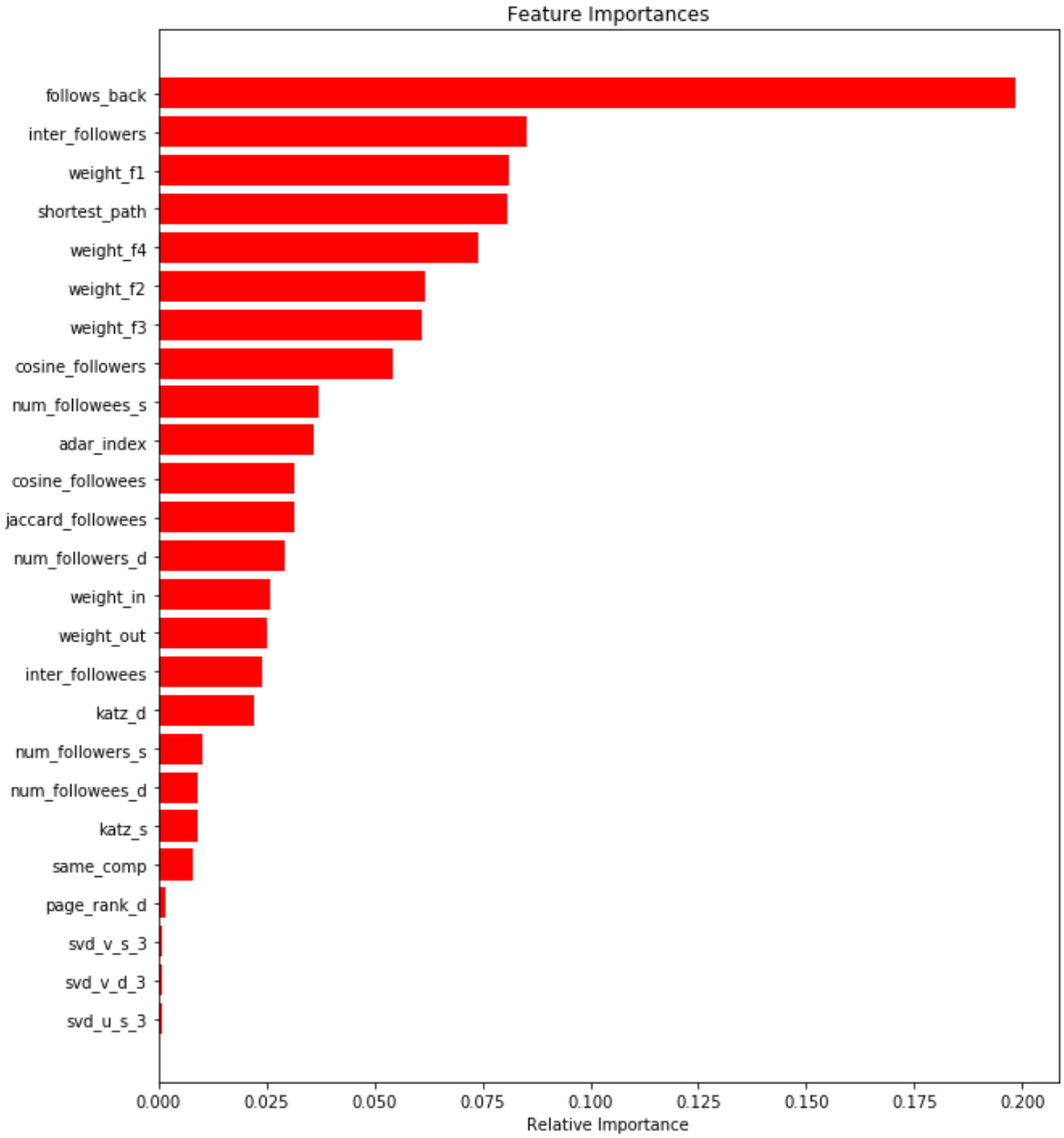
In [24]:

```
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 [25]:

```
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.

set_1

In [94]:

```
t=pd.DataFrame(df_final_train)
```

In [95]:

```
w=t['num_followers_s']
r=t['num_followers_d']
```

In [96]:

```
p=w.shape
```

In [97]:

```
pre=[]
for i in range(0,p[0]):
    pre.append(w[i]*r[i])
```

In [104]:

```
print(pre[0:10])
```

```
[66, 56, 340, 84, 2, 132, 63, 0, 442, 16]
```

In [102]:

```
df_final_train['Preferential Attachment']=pre
```

In [103]:

```
df_final_train.columns
```

Out[103]:

```
Index(['jaccard_followers', 'jaccard_followees', 'cosine_followers',
       'cosine_followees', 'num_followers_s', 'num_followers_d',
       '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',
       'Preferential Attachment'],
      dtype='object')
```

In [105]:

```
w=df_final_test['num_followers_s']
r=df_final_test['num_followers_d']
```

In [107]:

```
p=w.shape
```

In [108]:

```
pre=[]
for i in range(0,p[0]):
    pre.append(w[i]*r[i])
```

In [109]:

```
print(pre[0:10])
```

```
[84, 27, 24, 18, 18, 99, 288, 75, 6, 45]
```

In [110]:

```
df_final_test['Preferential Attachment']=pre
```

In [111]:

```
# normalize the given data as to make thenm on comman scale

from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

'''encode numerical feature '''
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.

normalizer.fit(df_final_train['Preferential Attachment'].values.reshape(-1,1)) # use code f

X_train_ac_norm = normalizer.transform(df_final_train['Preferential Attachment'].values.res

X_test_ac_norm = normalizer.transform(df_final_test['Preferential Attachment'].values.resha

print("After vectorizations")
print(X_train_ac_norm.shape, y_train.shape)

print(X_test_ac_norm.shape, y_test.shape)

print("="*100)
```

```
After vectorizations
(100002, 1) (100002,)
(50002, 1) (50002,)
====================================================================================
========================
```

In [112]:

```
df_final_train['Preferential Attachment']=X_train_ac_norm
df_final_test['Preferential Attachment']=X_test_ac_norm
```

In [115]:

```
df_final_test.columns
```

Out[115]:

```
Index(['jaccard_followers', 'jaccard_followees', 'cosine_followers',
       'cosine_followees', 'num_followers_s', 'num_followers_d',
       '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',
       'Preferential Attachment'],
      dtype='object')
```

task_2

In [138]:

```
a=t['svd_v_s_1']
b=t['svd_v_s_2']
c=t['svd_v_s_3']
d=t['svd_v_s_4']
e=t['svd_v_s_5']
f=t['svd_v_s_6']
g=t['svd_u_s_1']
h=t['svd_u_s_2']
i=t['svd_u_s_3']
j=t['svd_u_s_4']
k=t['svd_u_s_5']
l=t['svd_u_s_6']
```

In [139]:

```
from scipy.sparse import vstack
nwe=np.vstack((a,b,c,d,e,f,i,j,k,l,g,h)).T
```

In [140]:

```
p=nwe.shape
```

In [141]:

```
p[0]
```

Out[141]:

```
100002
```

In [143]:

```
# svd_d
nwe_2=np.vstack((t['svd_v_d_1'],t['svd_v_d_2'],t['svd_v_d_3'],t['svd_v_d_4'],t['svd_v_d_5']
```

In [144]:

```
nwe_2.shape
```

Out[144]:

```
(100002, 12)
```

In [145]:

```
nwe[0]
```

Out[145]:

```
array([-7.21566139e-13,  3.92581658e-13,  1.98370281e-06,  1.54507573e-13,
        8.10821415e-13,  1.71970040e-14,  1.04304248e-05,  6.67803265e-13,
        2.45109502e-13,  3.58494474e-12, -1.66629360e-13,  4.61379396e-13])
```

In [146]:

```
svdot=[]
for i in range(0,p[0]):
    svdot.append(np.dot(nwe[i],nwe_2[i]))
```

In [147]:

```
print(len(svdot))
```

```
100002
```

In [148]:

```
df_final_train['svd_dot']=svdot
```

In [149]:

```
nwe=np.vstack((df_final_test['svd_v_d_1'],df_final_test['svd_v_d_2'],df_final_test['svd_v_d
```

In [150]:

```
nwe_2=np.vstack((df_final_test['svd_v_s_1'],df_final_test['svd_v_s_2'],df_final_test['svd_v
```

In [152]:

```
p=nwe.shape
```

In [153]:

```
svdot=[]
for i in range(0,p[0]):
    svdot.append(np.dot(nwe[i],nwe_2[i]))
```

In [154]:

```
df_final_test['svd_dot']=svdot
```

task_3

In [157]:

```
# parameter tunning xgboost

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)}

xgb_model = xgb.XGBClassifier(random_state=25,n_jobs=-1)

rf_random = RandomizedSearchCV(xgb_model, param_distributions=param_dist,
                                    n_iter=5,cv=10,scoring='f1',random_state=25)

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 [0.98108184 0.98138645 0.98082843 0.98093917 0.9810614 ]
```

In [158]:

```
# printbest estimater
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=0.1, max_delta_step=0, max_depth=12,
              min_child_weight=1, min_samples_leaf=33, min_samples_split=13
8,
              missing=None, n_estimators=109, n_jobs=-1, nthread=None,
              objective='binary:logistic', random_state=25, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=1, seed=None, silent=None,
              subsample=1, verbosity=1)
```

In [160]:

```
# test phase
xg=xgb.XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=12,
              min_child_weight=1, min_samples_leaf=33, min_samples_split=138,
              missing=None, n_estimators=109, n_jobs=-1, nthread=None,
              objective='binary:logistic', random_state=25, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=1, seed=None, silent=None,
              subsample=1, verbosity=1)
```

In [162]:

```
# predicting
xg.fit(df_final_train,y_train)
y_train_pred = xg.predict(df_final_train)
y_test_pred = xg.predict(df_final_test)
```

In [163]:

```
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 0.9973775348827898
Test f1 score 0.9276552077988712
```

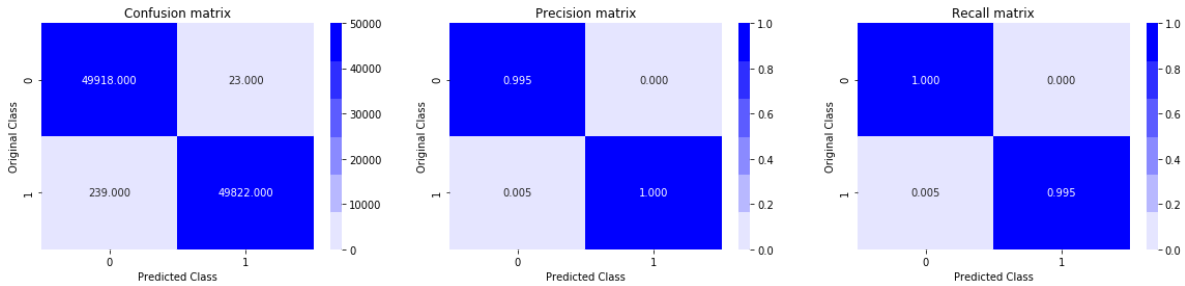
In [164]:

```
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()
```

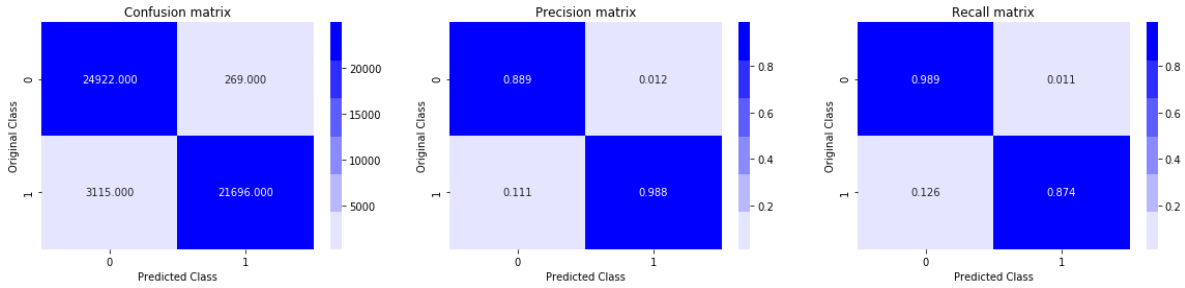
In [165]:

```
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
```

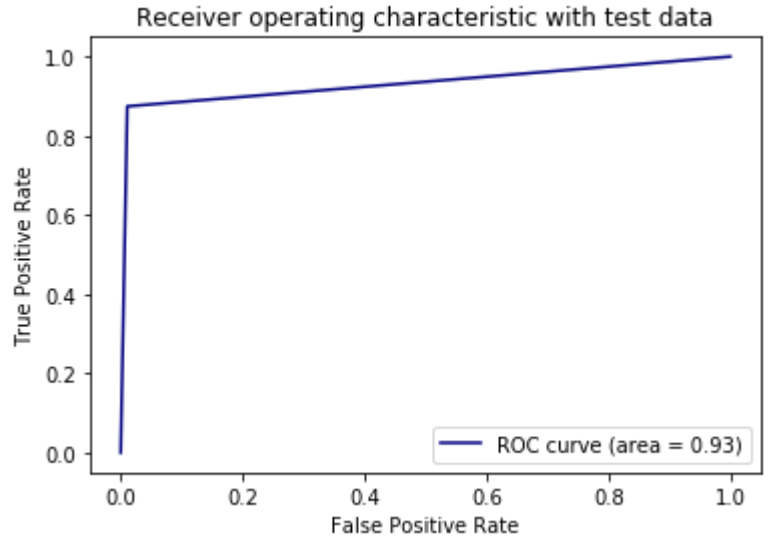



```
Test confusion_matrix
```



In [166]:

```
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()
```



conclusion

In [2]:

```
# Please compare all your models using Prettytable library

from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["model", "train_f1_score", "test_f1_score"]
```

In [3]:

```
x.add_row(['random_forest',0.9654108476778174,0.9276322776609528])
x.add_row(['XGBOOST',0.9973775348827898,0.9276552077988712])
```

In [4]:

```
print(x)
```

```
+---------------+--------------------+--------------------+
|     model     |   train_f1_score   |   test_f1_score    |
+---------------+--------------------+--------------------+
| random_forest | 0.9654108476778174 | 0.9276322776609528 |
|    XGBOOST    | 0.9973775348827898 | 0.9276552077988712 |
+---------------+--------------------+--------------------+
```

In [ ]: