Social network Graph Link Prediction - Facebook Challenge

In [2]:
```
#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 [7]:

```
#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 [17]:

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

In [ ]:

```

```

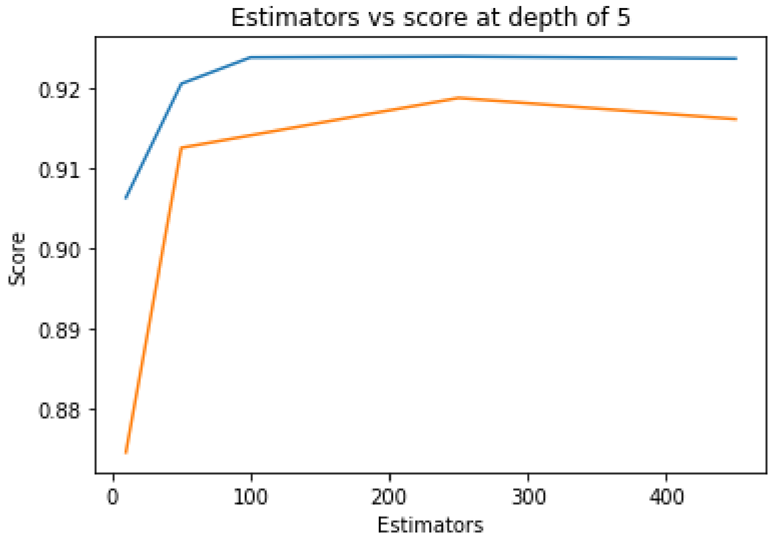
In [18]:

```
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 [0]:
```
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=Fa
    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
```

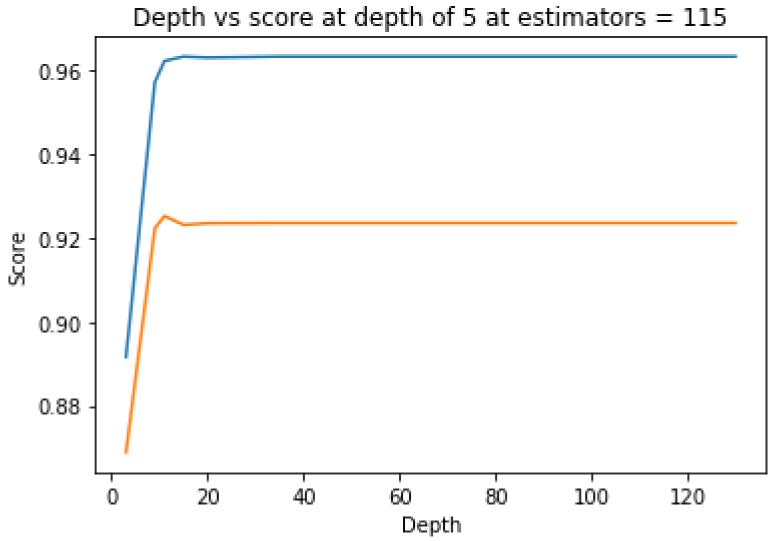
Out[6]:
```
Text(0.5,1,'Estimators vs score at depth of 5')
```


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

In [0]:

```
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=
    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 [0]:

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

In [0]:
```
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 [0]:
```
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 [0]:
```
clf.fit(df_final_train,y_train)
y_train_pred = clf.predict(df_final_train)
y_test_pred = clf.predict(df_final_test)
```

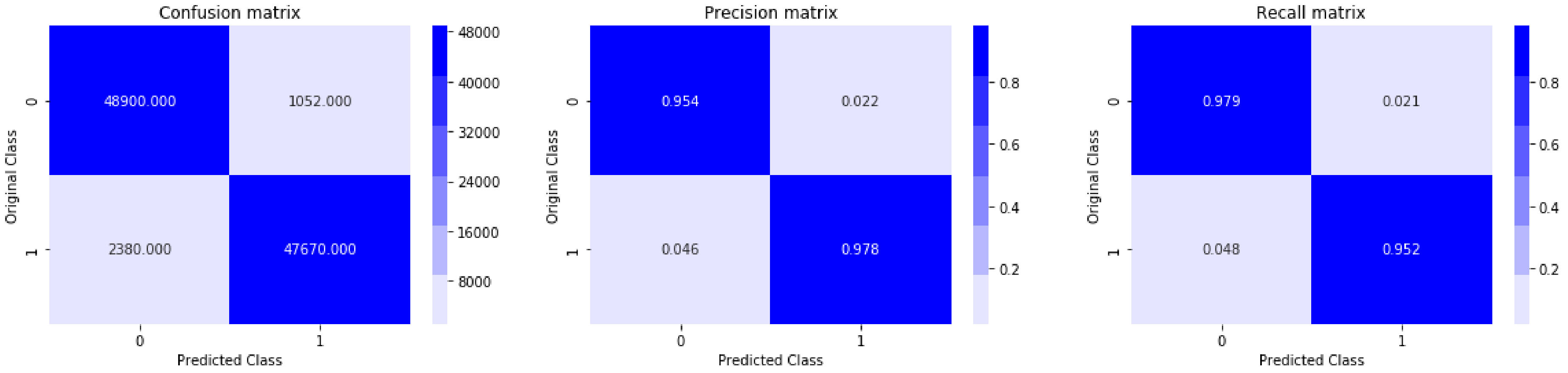
In [0]:
```
from sklearn.metrics import f1_score
print('Train f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))
```

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

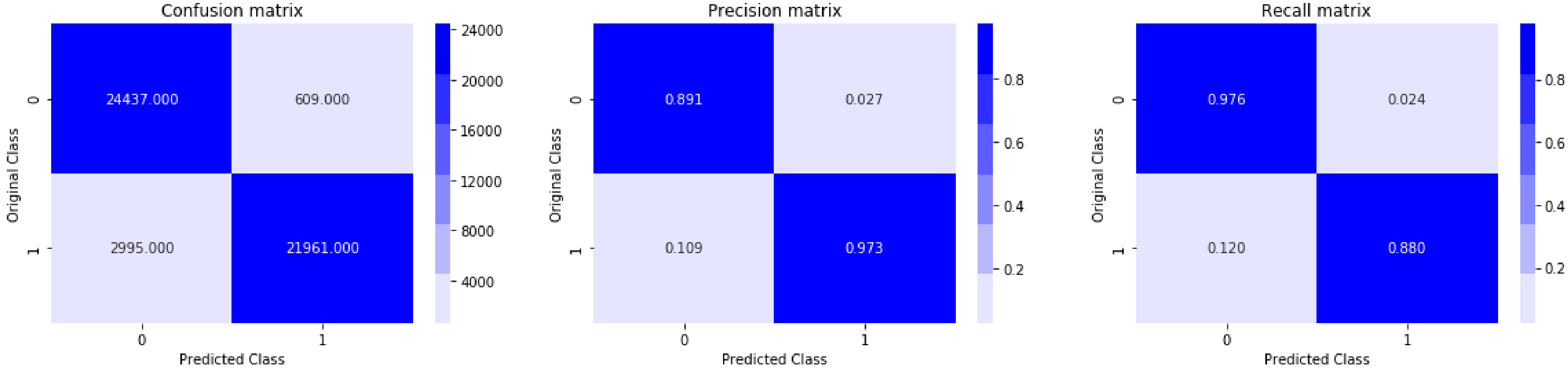
In [49]:
```
from sklearn.metrics import confusion_matrix
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)

    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 [0]:
```
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
```

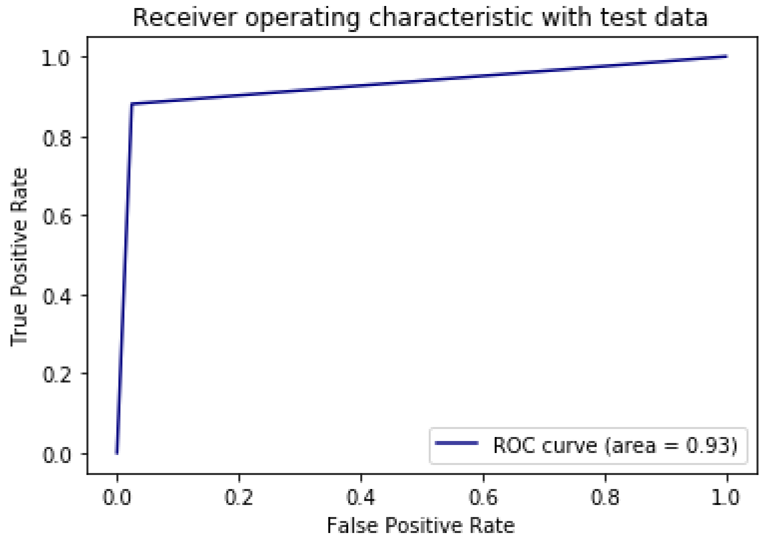



```
Test confusion_matrix
```

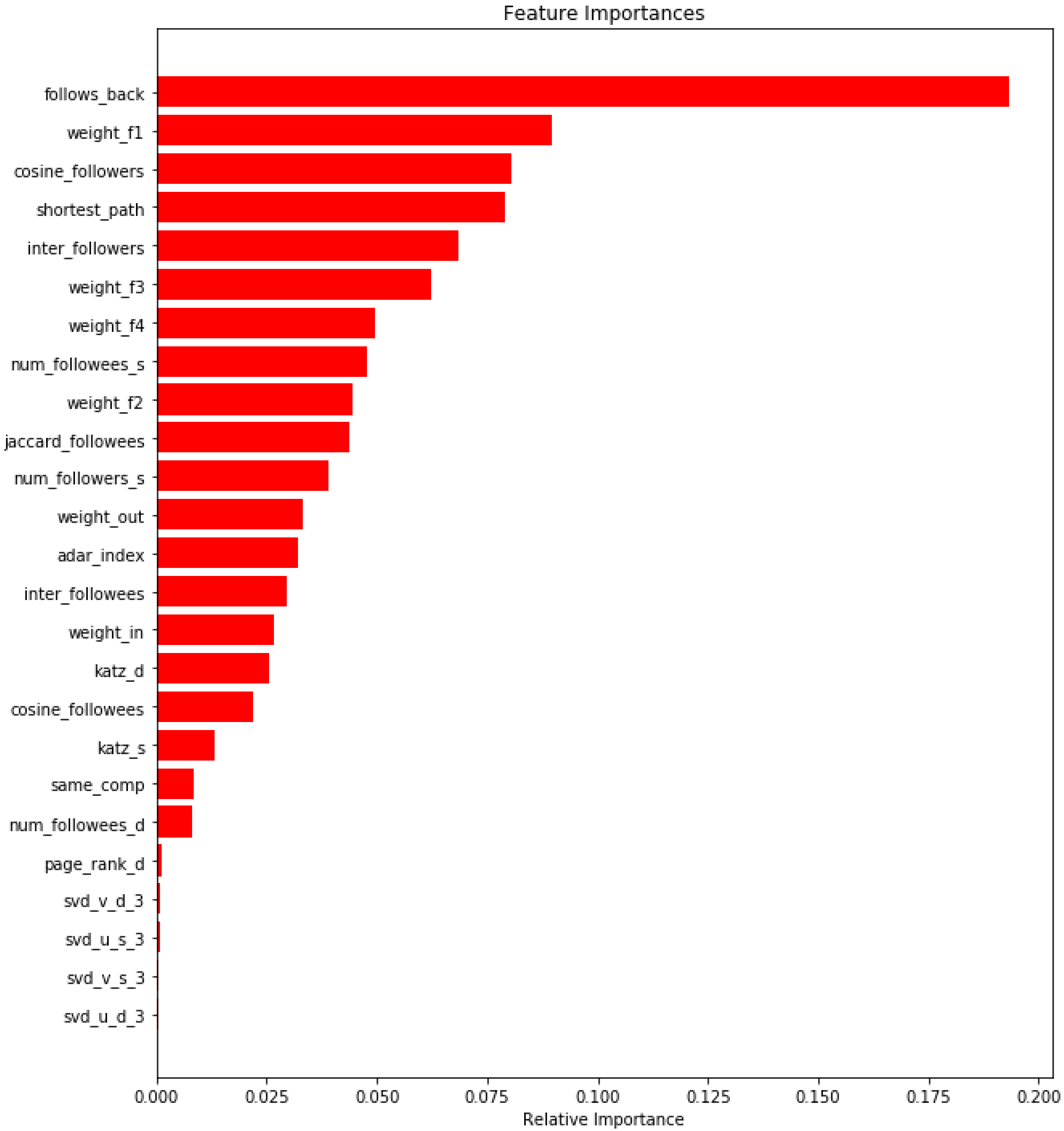


In [0]:

```
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 [0]:
```
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()
```

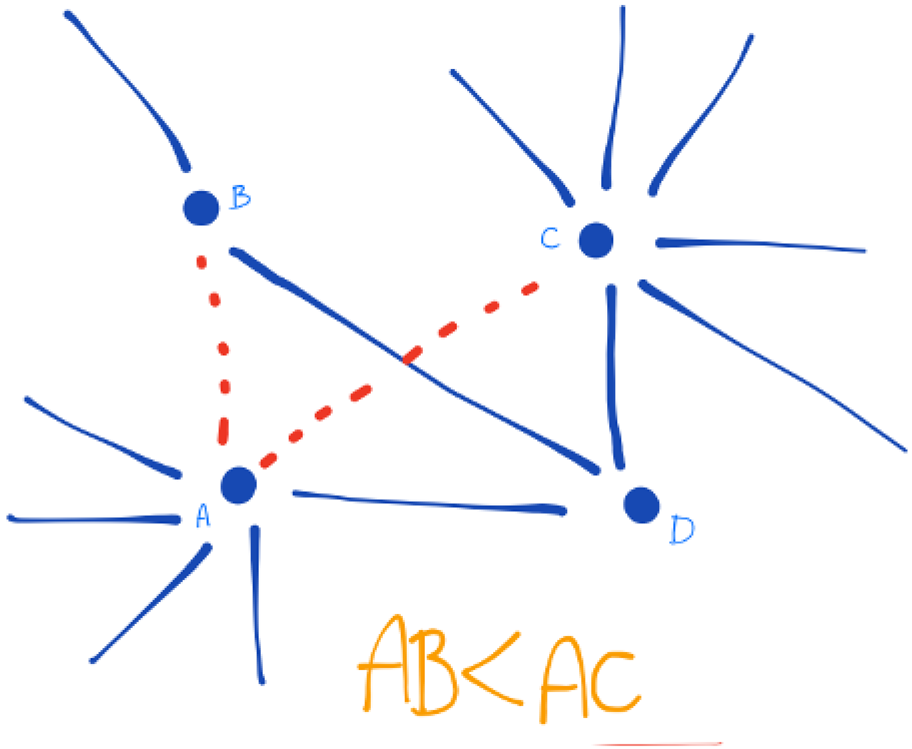

Feature Importances
follows_back
weight_f1
cosine_followers
shortest_path
inter_followers
weight_f3
weight_f4
num_followees_s
weight_f2
jaccard_followees
num_followers_s
weight_out
adar_index
inter_followees
weight_in
katz_d
cosine_followees
katz_s
same_comp
num_followees_d
page_rank_d
svd_v_d_3
svd_u_s_3
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.

Preferential Attachment :

One well-known concept in social networks is that users with many friends tend to create more connections in the future. This is due to the fact that in some social networks, like in finance, the rich get richer. We estimate how "rich" our two vertices are by calculating the multiplication between the number of friends (|Γ(x)|) or followers each vertex has. It may be noted that the similarity index does not require any node neighbor information; therefore, this similarity index has the lowest computational complexity.



In [19]:
```
df_final_train.columns
# df_final_train["num_followees_d"]
```

Out[19]:
```
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',
       'num_followers_d'],
      dtype='object')
```

In [12]:
```
# Kind of like friends of friends and mutual friends
# Perferntial Attachment is two vertices are by calculating the multiplication
# between the number of friends (|Γ(x)|) or followers each vertex has.
# So Source and Destination Followees multiplication
```

In [20]:

```
df_final_train['perferntial_attach'] = df_final_train["num_followees_s"] * df_final_train["num_followees_d"
df_final_train['perferntial_attach_follower'] = df_final_train["num_followers_s"] * df_final_train["num_fol


df_final_train.head()
```

Out[20]:

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	num_followers_d	perferntial_attach	perferntial_attacl
1.719702e-14	-1.355368e-12	4.675307e-13	1.128591e-06	6.616550e-14	9.771077e-13	4.159752e-14	6	120	
2.251737e-10	1.245101e-12	-1.636948e-10	-3.112650e-10	6.738902e-02	2.607801e-11	2.372904e-09	94	8662	
3.365389e-19	-1.238370e-18	1.438175e-19	-1.852863e-19	-5.901864e-19	1.629341e-19	-2.572452e-19	28	902	
4.498061e-13	-9.818087e-10	3.454672e-11	5.213635e-08	9.595823e-13	3.047045e-10	1.246592e-13	11	35	
1.407670e-14	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	1	33	

In [21]:
```
df_final_test['perferntial_attach'] = df_final_test["num_followees_s"] * df_final_test["num_followees_d"]
df_final_test['perferntial_attach_follower'] = df_final_test["num_followers_s"] * df_final_test["num_follow
df_final_test.head()
```

Out[21]:

inter_followees	adar_index	...	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	num_followers_d	perferntial_attach
0	0.000000	...	5.535503e-14	-9.994076e-10	5.791910e-10	3.512364e-07	2.486658e-09	2.771146e-09	1.727694e-12	14	54
0	0.000000	...	4.701436e-15	-9.360516e-12	3.206809e-10	4.668696e-08	6.665777e-12	1.495979e-10	9.836670e-14	17	19
0	0.000000	...	4.199834e-14	-4.253075e-13	4.789463e-13	3.479824e-07	1.630549e-13	3.954708e-13	3.875785e-14	10	144
0	0.000000	...	2.817657e-13	-2.162590e-11	6.939194e-12	1.879861e-05	4.384816e-12	1.239414e-11	6.483485e-13	37	340
7	6.136433	...	9.656662e-14	-8.742904e-12	7.467370e-12	1.256880e-05	3.636983e-12	3.948463e-12	2.415863e-13	27	405

Adding SVD Dot Feature that is product of source and destination SVDs

In [ ]:
```
# Dot product of Source and Destination Node SVD
```

In [25]:
```
source = ['svd_u_s_1', 'svd_u_s_2','svd_u_s_3', 'svd_u_s_4', 'svd_u_s_5', 'svd_u_s_6']
source_v = ['svd_v_s_1','svd_v_s_2', 'svd_v_s_3', 'svd_v_s_4', 'svd_v_s_5', 'svd_v_s_6']
destination = ['svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3', 'svd_u_d_4', 'svd_u_d_5','svd_u_d_6']
destination_v = ['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 [22]:

```
# source_array = []
# destination_array = []
# svd_dot = []

# for num in tqdm(range(df_final_train.shape[0])):
#     for i,j in (zip(source,destination)):
#         source_array.append(np.array(df_final_train[i].iloc[num]))
#         destination_array.append(np.array(df_final_train[j].iloc[num]))
#     svd_dot.append(np.dot(source_array, destination_array))
# df_final_train['svd_dot']=svd_dot

```

In [32]:

```
# Corrections
f = 0;
for i,j in tqdm(zip(source,destination)):
    f = f + df_final_train[i] * df_final_train[j]
df_final_train['svd_dot_source'] = f

f = 0;
for i,j in tqdm(zip(source_v,destination_v)):
    f = f + df_final_train[i] * df_final_train[j]
df_final_train['svd_dot_dest'] = f

f = 0;
for i,j in tqdm(zip(source,destination)):
    f = f + df_final_test[i] * df_final_test[j]
df_final_test['svd_dot_source'] = f

f = 0;
for i,j in tqdm(zip(source_v,destination_v)):
    f = f + df_final_test[i] * df_final_test[j]
df_final_test['svd_dot_dest'] = f

```

```
6it [00:00, 482.16it/s]
6it [00:00, 480.27it/s]
6it [00:00, 623.44it/s]
6it [00:00, 337.19it/s]
```

In [33]:
```
df_final_train.head()
```

Out[33]:

'd_v_d_2	svd_v_d_3	svd_v_d_4	svd_v_d_5	svd_v_d_6	num_followers_d	perferntial_attach	perferntial_attach_follower	svd_dot_source	svd
;307e-13	1.128591e-06	6.616550e-14	9.771077e-13	4.159752e-14	6	120	36	1.114958e-11	2.2:
;36948e-10	-3.112650e-10	6.738902e-02	2.607801e-11	2.372904e-09	94	8662	8836	3.192812e-03	9.0(
3175e-19	-1.852863e-19	-5.901864e-19	1.629341e-19	-2.572452e-19	28	902	784	1.787503e-35	2.4(
4672e-11	5.213635e-08	9.595823e-13	3.047045e-10	1.246592e-13	11	35	121	4.710376e-20	3.1!
000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	1	33	1	7.773952e-14	0.0(

In [35]:
```
df_final_test.head()
```

Out[35]:

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

5 rows × 56 columns

In [ ]:
```

```

Random Forest

In [37]:
```
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,
                                  cv=2,scoring='f1')

rf_random.fit(df_final_train,y_train)

```

Out[37]:
```
RandomizedSearchCV(cv=2, error_score='raise-deprecating',
                   estimator=RandomForestClassifier(bootstrap=True,
                                                    class_weight=None,
                                                    criterion='gini',
                                                    max_depth=None,
                                                    max_features='auto',
                                                    max_leaf_nodes=None,
                                                    min_impurity_decrease=0.0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
                                                    min_weight_fraction_leaf=0.0,
                                                    n_estimators='warn',
                                                    n_jobs=-1, oob_scor...
                                                    random_state=25, verbose=0,
                                                    warm_start=False),
                   iid='warn', n_iter=10, n_jobs=None,
                   param_distributions={'max_depth': <scipy.stats._distn_infrastructure.rv_frozen object at 0
x1a3f8af7f0>,
                                        'n_estimators': <scipy.stats._distn_infrastructure.rv_frozen object a
t 0x1a3f9455c0>},
```

```
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring='f1', verbose=0)
```

In [38]:
```
print('mean test scores',rf_random.cv_results_['mean_test_score'])
```

```
mean test scores [0.9663106  0.96309182 0.96826787 0.9609609  0.96657034 0.96474245
 0.96476606 0.96826012 0.96827655 0.96651507]
```

In [39]:
```
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=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=114,
                       n_jobs=-1, oob_score=False, random_state=25, verbose=0,
                       warm_start=False)
```

In [44]:
```
n_estimator_rf = 114
max_depth_rf = 14
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=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=114,
                       n_jobs=-1, oob_score=False, random_state=25, verbose=0,
                       warm_start=False)
```

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)
```

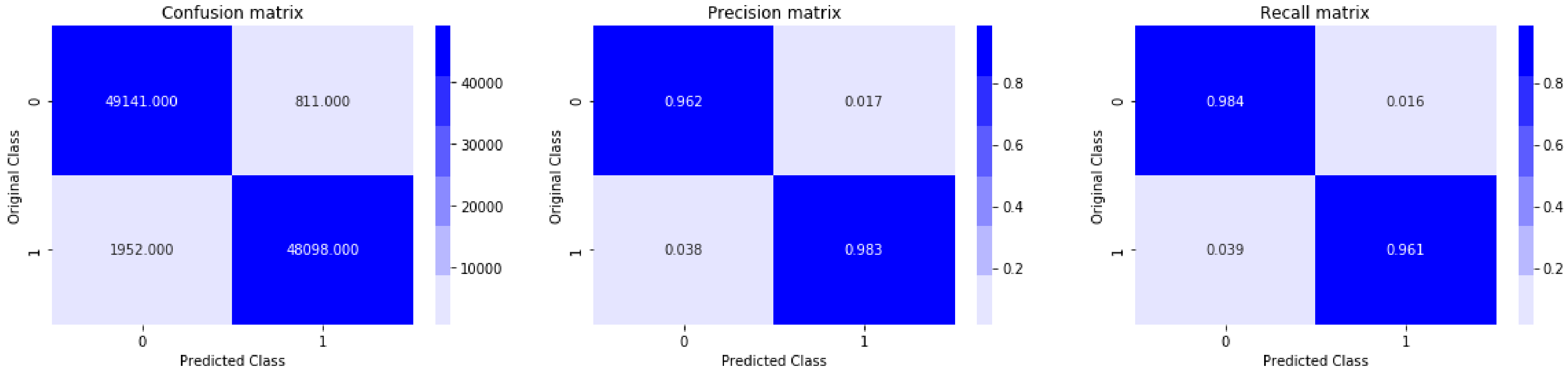
In [46]:
```
from sklearn.metrics import f1_score

Train_f1_Score_rf = f1_score(y_train,y_train_pred)
Test_f1_Score_rf = f1_score(y_test,y_test_pred)

print('Train f1 score',Train_f1_Score_rf)
print('Test f1 score',Test_f1_Score_rf)
```

```
Train f1 score 0.9720793459917744
Test f1 score 0.9188237291012475
```

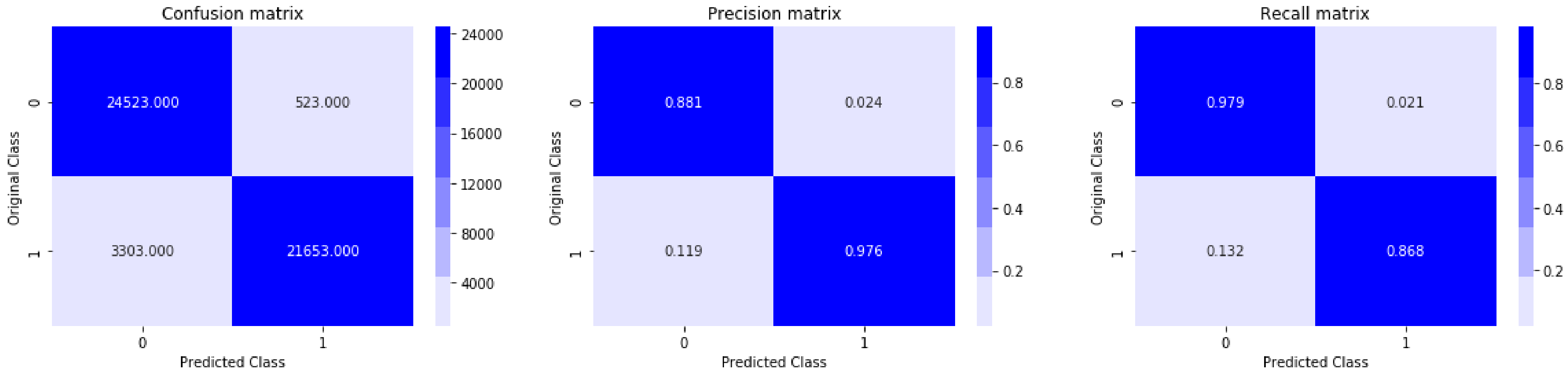
In [50]:

```
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 [ ]:
```
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()
```

XGboost

In [40]:
```
from datetime import datetime
from sklearn.metrics import f1_score
from sklearn.model_selection import RandomizedSearchCV
from xgboost import XGBClassifier
```

In [42]:
```
start = datetime.now()
clf = XGBClassifier()
params = {"n_estimators":sp_randint(105,125),
              "max_depth": sp_randint(10,15)
              }
rf = RandomizedSearchCV(clf, param_distributions=params,cv=2,scoring='f1', n_jobs=-1)



rf.fit(df_final_train,y_train)
print('mean test scores',rf.cv_results_['mean_test_score'])
print(datetime.now() - start)
```

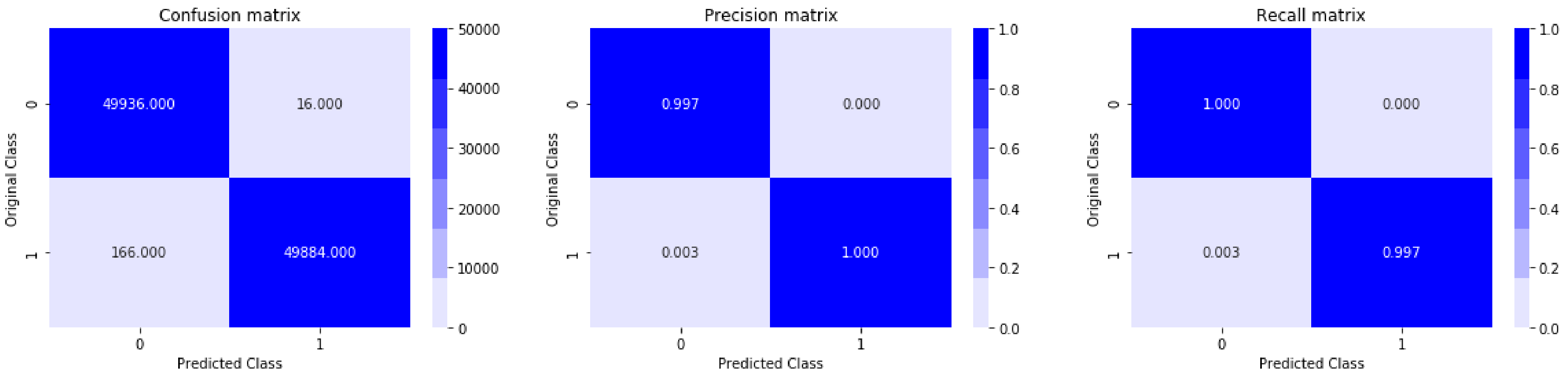
```
mean test scores [0.97926699 0.97903336 0.97881123 0.97901685 0.9791557  0.97929291
 0.97896439 0.97933791 0.97906177 0.97911945]
0:14:49.802030
```

In [43]:
```
print(rf.best_estimator_)
```

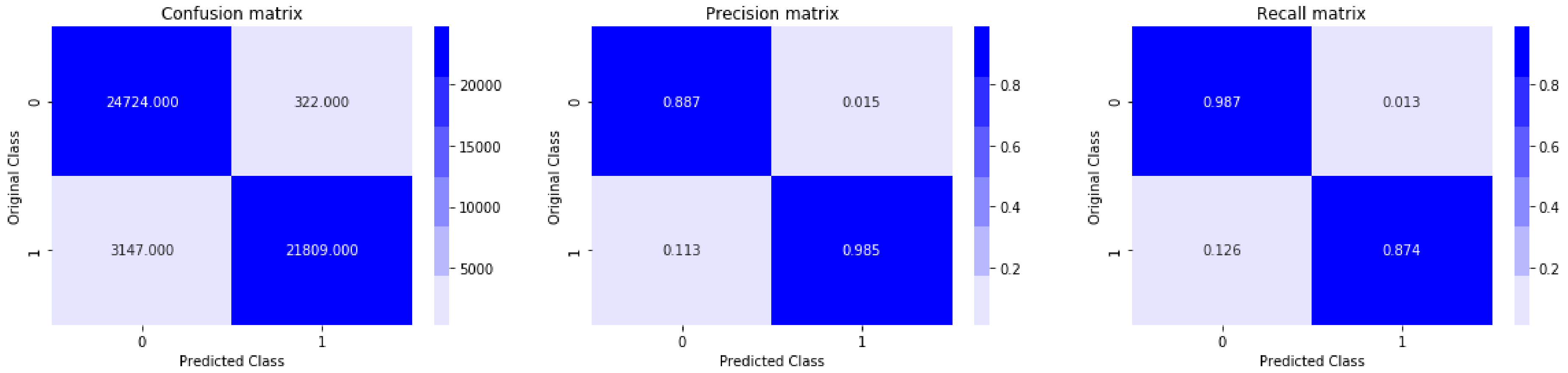
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
              max_depth=12, min_child_weight=1, missing=None, n_estimators=118,
              n_jobs=1, nthread=None, objective='binary:logistic',
              random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
              seed=None, silent=True, subsample=1)
```

In [51]:
```
n_estimator_xgb = 118
max_depth_xgb = 12
xgb =  XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
              max_depth=12, min_child_weight=1, missing=None, n_estimators=118,
              n_jobs=1, nthread=None, objective='binary:logistic',
              random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
              seed=None, silent=True, subsample=1)
xgb.fit(df_final_train,y_train)
y_pred_train = xgb.predict(df_final_train)
y_pred_test = xgb.predict(df_final_test)

Train_f1_Score_xgb = f1_score(y_train,y_pred_train)
Test_f1_Score_xgb = f1_score(y_test,y_pred_test)

print('Train f1 score',Train_f1_Score_xgb)
print('Test f1 score',Test_f1_Score_xgb)

print('Train confusion_matrix')
plot_confusion_matrix(y_train,y_pred_train)
print('Test confusion_matrix')
plot_confusion_matrix(y_test,y_pred_test)
```

```
Train f1 score 0.9981790895447723
Test f1 score 0.9263278611931106
Train confusion_matrix
```

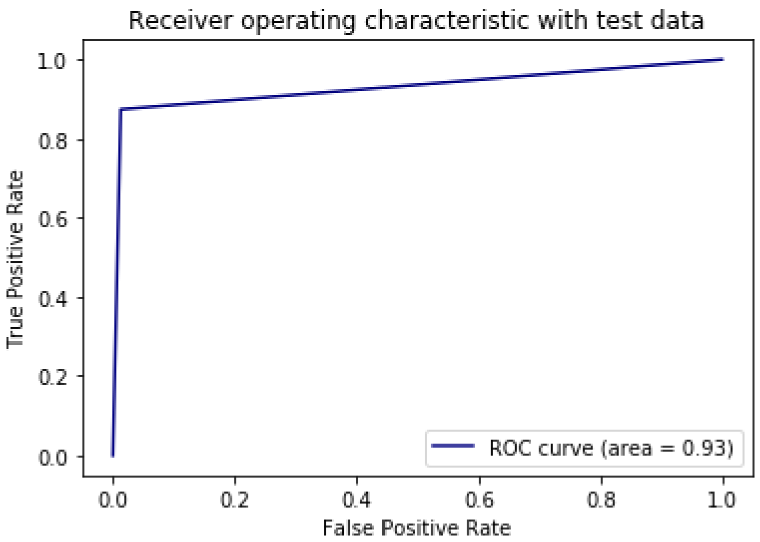



```
Test confusion_matrix
```

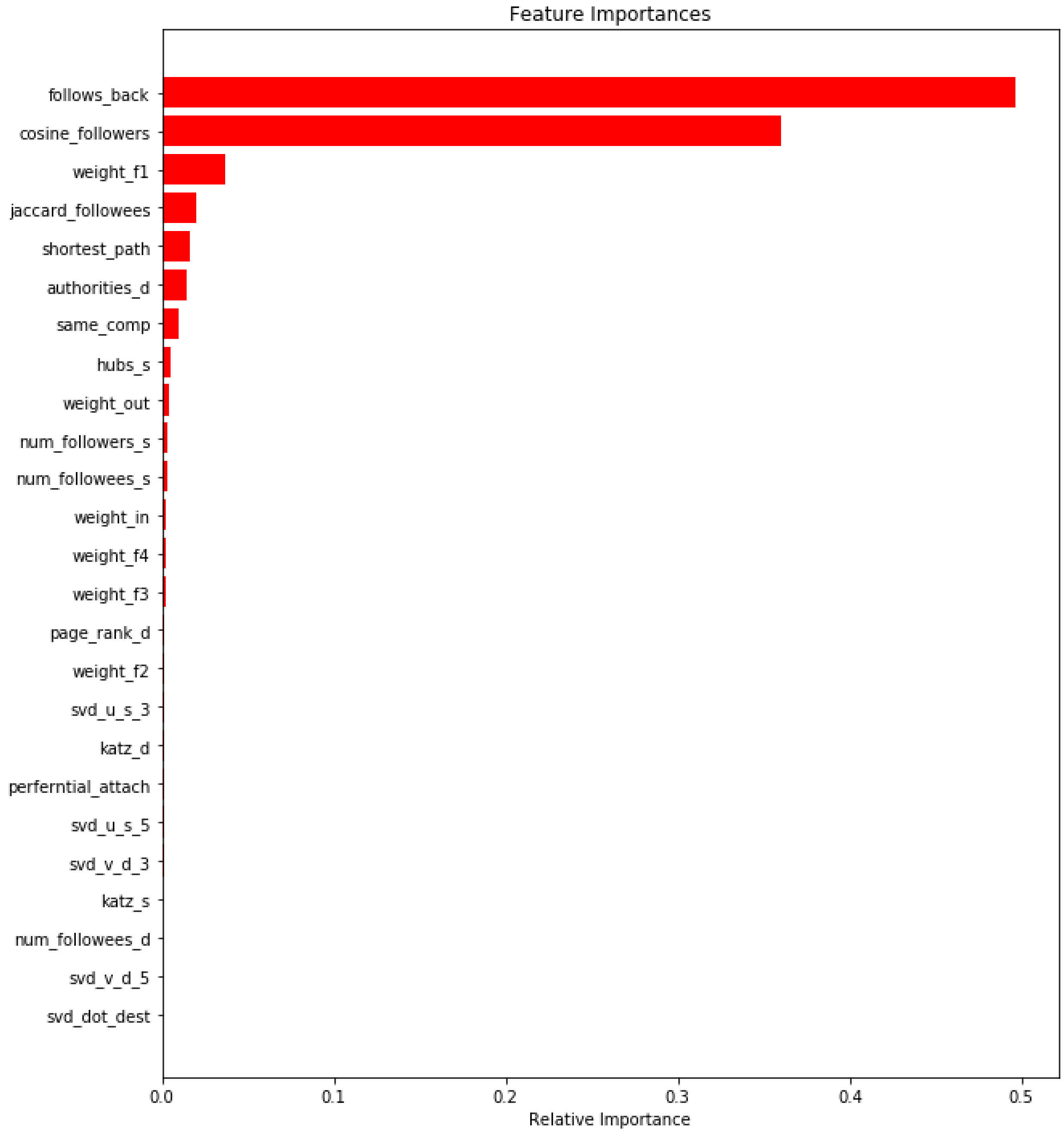


In [52]:

```
from sklearn.metrics import roc_curve, auc
fpr,tpr,ths = roc_curve(y_test,y_pred_test)
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 [53]:
```
features = df_final_train.columns
importances = xgb.feature_importances_
indices = (np.argsort(importances))[-25:]
plt.figure(figsize=(10,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='r', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```


Feature Importances
follows_back
cosine_followers
weight_f1
jaccard_followees
shortest_path
authorities_d
same_comp
hubs_s
weight_out
num_followers_s
num_followees_s
weight_in
weight_f4
weight_f3
page_rank_d
weight_f2
svd_u_s_3
katz_d
perferntial_attach
svd_u_s_5
svd_v_d_3
katz_s
num_followees_d
svd_v_d_5
svd_dot_dest
0.0
0.1
0.2
0.3
0.4
0.5
Relative Importance

In [54]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model", "n_estimators", "max_depth", "Train_f1_Score","Test_f1_Score"]
x.add_row(['Random Forest',n_estimator_rf,max_depth_rf,Train_f1_Score_rf,Test_f1_Score_rf])
x.add_row(['XGBOOST',n_estimator_xgb,max_depth_xgb,Train_f1_Score_xgb,Test_f1_Score_xgb])
print(x)
```

```
+---------------+--------------+-----------+--------------------+--------------------+
|     Model     | n_estimators | max_depth |   Train_f1_Score   |   Test_f1_Score    |
+---------------+--------------+-----------+--------------------+--------------------+
| Random Forest |     114      |     14    | 0.9720793459917744 | 0.9188237291012475 |
|    XGBOOST    |     118      |     12    | 0.9981790895447723 | 0.9263278611931106 |
+---------------+--------------+-----------+--------------------+--------------------+
```

Conclusion :

- Facebook Friend Recommendation Case study is one of its kind as most of the Internet Companies like Facebook, Instagram, redit, Github etc has graph based features. Understanding various Graph feature is important and all through a fun ride.
- In EDA we made use of np.percentile and saw how we can even look at 99.1 percentile
- Explored different Graph Based Feature Engineering in which **Follow back & cosine followers** were important one
- Added new features as part of Assignment
- XGboost performed better than Random Forest

In [ ]:
```

```