Social network Graph Link Prediction - Facebook Challenge

In [1]:
```
#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
```

```
C:\Users\Himanshu Pc\Anaconda3\lib\site-packages\sklearn\ensemble\weight_boosting.py:29: DeprecationWarning: numpy.core.umath_tests is an internal Nu
mPy module and should not be imported. It will be removed in a future NumPy release.
  from numpy.core.umath_tests import inner1d
```

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

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

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

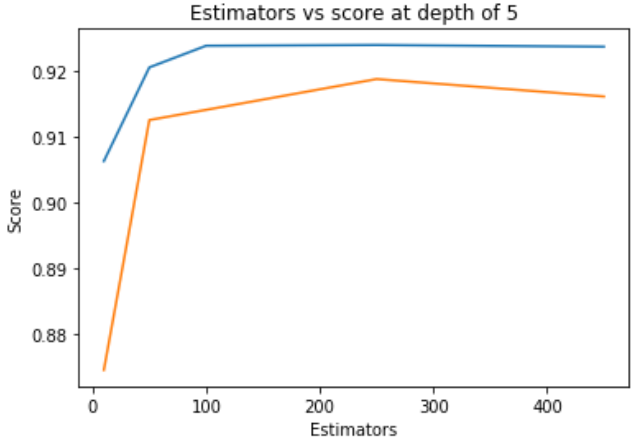
In [4]:
```
y_train = df_final_train.indicator_link
y_test = df_final_test.indicator_link
```

In [5]:
```
df_final_train.drop(['source_node', 'destination_node','indicator_link'],axis=1,inplace=True)
df_final_test.drop(['source_node', 'destination_node','indicator_link'],axis=1,inplace=True)
```

In [6]:
```
estimators = [10,50,100,250,450]
train_scores = []
test_scores = []
for i in estimators:
    clf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=5, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=52, min_samples_split=120,
            min_weight_fraction_leaf=0.0, n_estimators=i, n_jobs=-1,random_state=25,verbose=0,warm_start=False)
    clf.fit(df_final_train,y_train)
    train_sc = f1_score(y_train,clf.predict(df_final_train))
    test_sc = f1_score(y_test,clf.predict(df_final_test))
    test_scores.append(test_sc)
    train_scores.append(train_sc)
    print('Estimators = ',i,'Train Score',train_sc,'test Score',test_sc)
plt.plot(estimators,train_scores,label='Train Score')
plt.plot(estimators,test_scores,label='Test Score')
plt.xlabel('Estimators')
plt.ylabel('Score')
plt.title('Estimators vs score at depth of 5')
```

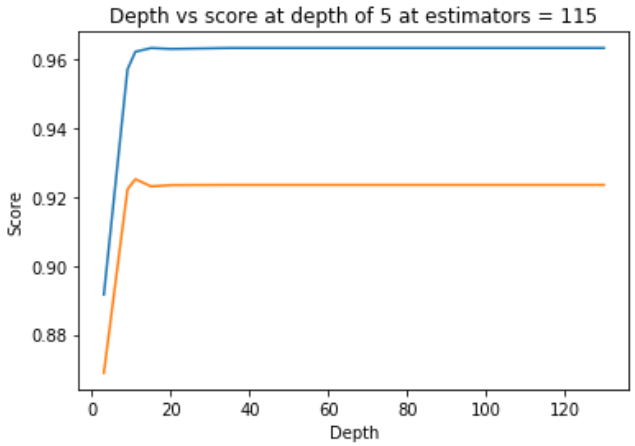
```
Estimators =  10 Train Score 0.9063252121775113 test Score 0.8745605278006858
Estimators =  50 Train Score 0.9205725512208812 test Score 0.9125653355634538
Estimators =  100 Train Score 0.9238690848446947 test Score 0.9141199714153599
Estimators =  250 Train Score 0.9239789348046863 test Score 0.9188007232664732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

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



In [7]:
```
depths = [3,9,11,15,20,35,50,70,130]
train_scores = []
test_scores = []
for i in depths:
    clf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=i, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=52, min_samples_split=120,
            min_weight_fraction_leaf=0.0, n_estimators=115, n_jobs=-1,random_state=25,verbose=0,warm_start=False)
    clf.fit(df_final_train,y_train)
    train_sc = f1_score(y_train,clf.predict(df_final_train))
    test_sc = f1_score(y_test,clf.predict(df_final_test))
    test_scores.append(test_sc)
    train_scores.append(train_sc)
    print('depth = ',i,'Train Score',train_sc,'test Score',test_sc)
plt.plot(depths,train_scores,label='Train Score')
plt.plot(depths,test_scores,label='Test Score')
plt.xlabel('Depth')
plt.ylabel('Score')
plt.title('Depth vs score at depth of 5 at estimators = 115')
plt.show()
```

```
depth =  3 Train Score 0.8916120853581238 test Score 0.8687934859875491
depth =  9 Train Score 0.9572226298198419 test Score 0.9222953031452904
depth =  11 Train Score 0.9623451340902863 test Score 0.9252318758281279
depth =  15 Train Score 0.9634267621927706 test Score 0.9231288356496615
depth =  20 Train Score 0.9631629153051491 test Score 0.9235051024711141
depth =  35 Train Score 0.9634333127085721 test Score 0.9235601652753184
depth =  50 Train Score 0.9634333127085721 test Score 0.9235601652753184
depth =  70 Train Score 0.9634333127085721 test Score 0.9235601652753184
depth =  130 Train Score 0.9634333127085721 test Score 0.9235601652753184
```



In [8]:
```
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 [9]:
```
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 [10]:
```
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 [11]:
```
clf.fit(df_final_train,y_train)
y_train_pred = clf.predict(df_final_train)
y_test_pred = clf.predict(df_final_test)
```

In [12]:
```
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))
```

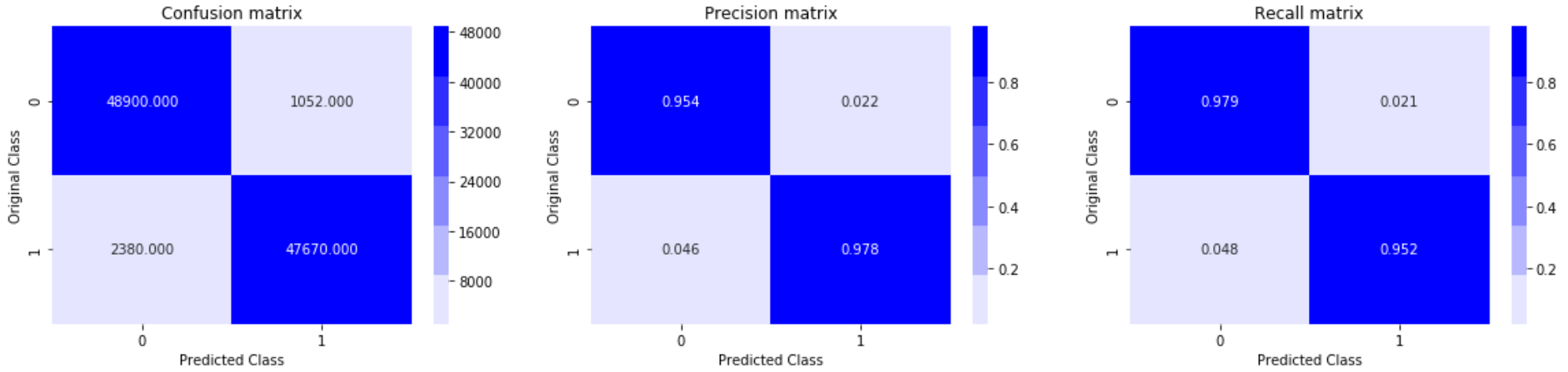
```
Train f1 score 0.9652533106548414
Test f1 score 0.9241678239279553
```

In [13]:

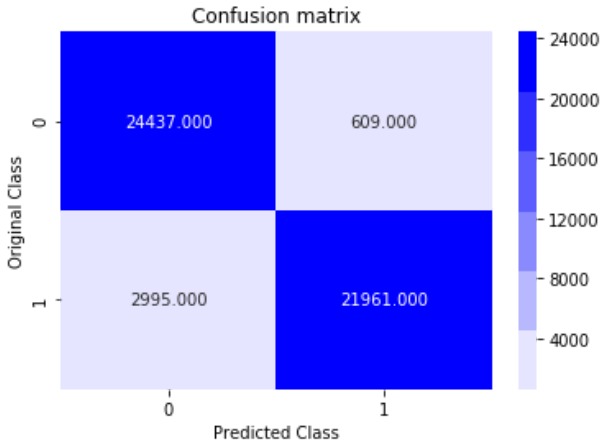
```
from sklearn.metrics import confusion_matrix
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)

    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()
```

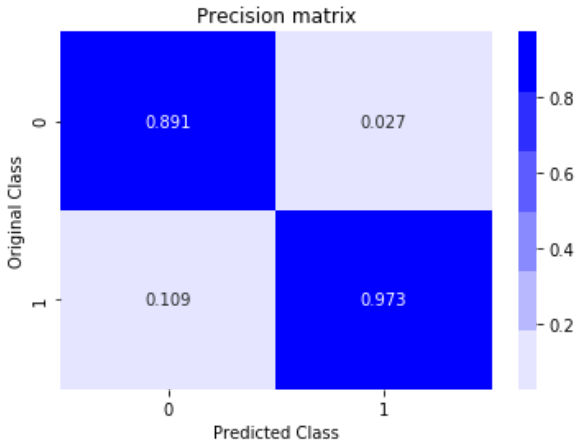
In [14]:
```
print('Train confusion_matrix')
plot_confusion_matrix(y_train,y_train_pred)
print('Test confusion_matrix')
plot_confusion_matrix(y_test,y_test_pred)
```

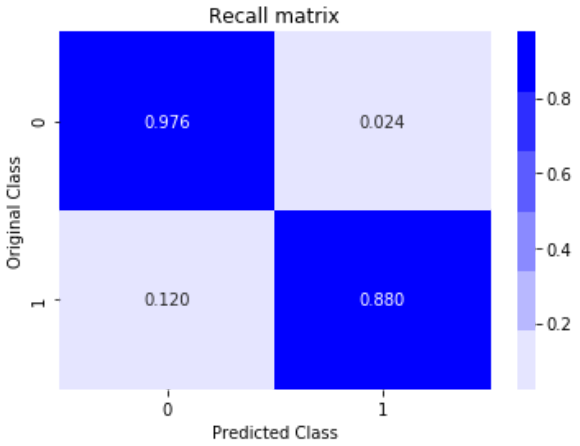
```
Train confusion_matrix
```



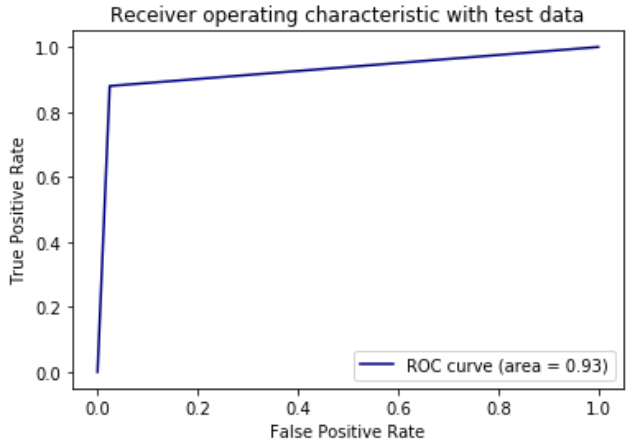

```
Test confusion_matrix
```


Confusion matrix
24437.000
609.000
2995.000
21961.000
Original Class
Predicted Class
0
1
24000
20000
16000
12000
8000
4000

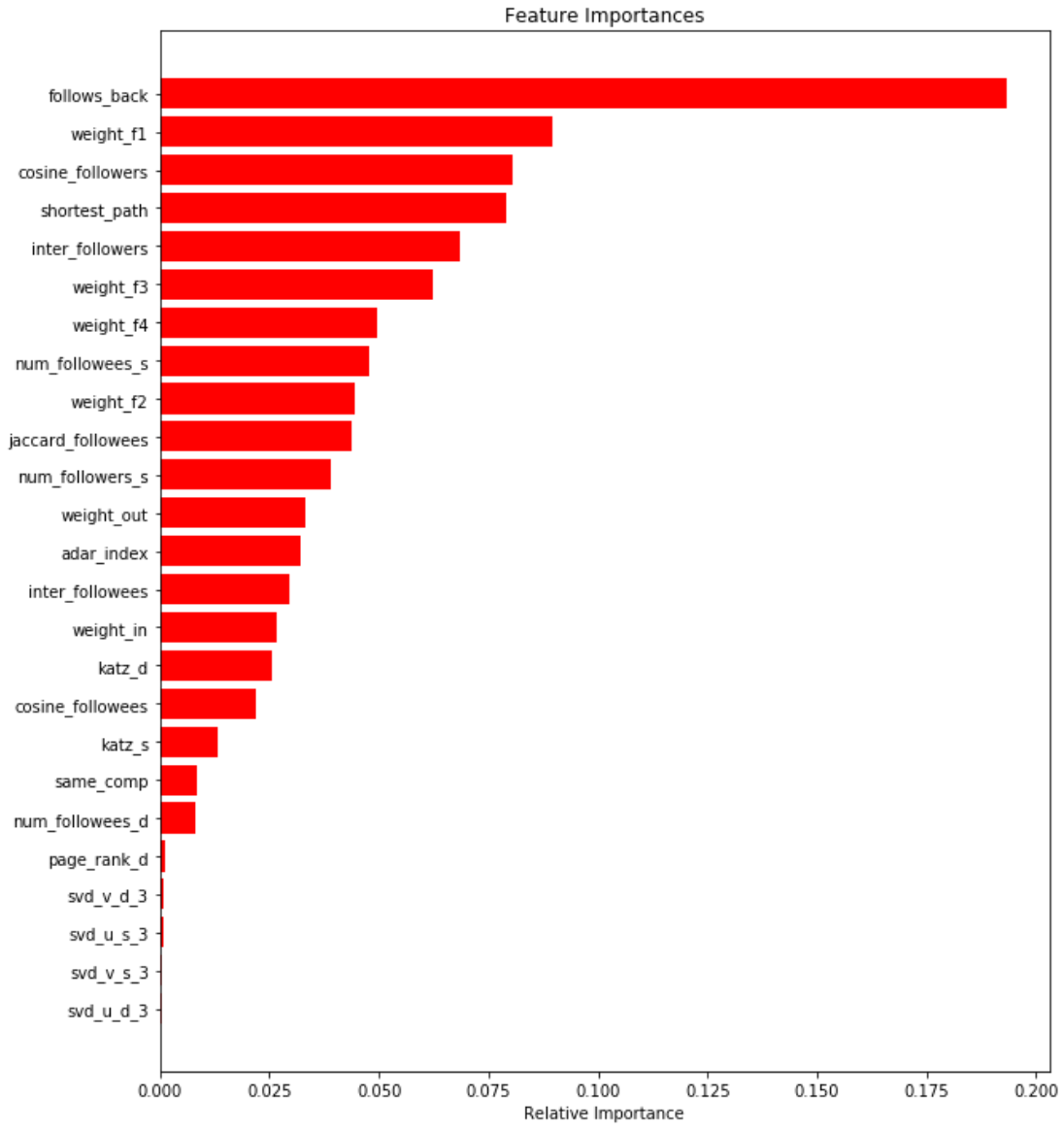

Precision matrix
0.891
0.027
0.109
0.973
Original Class
Predicted Class
0
1
0.8
0.6
0.4
0.2


Recall matrix
0.976
0.024
0.120
0.880
Original Class
Predicted Class
0
1
0.8
0.6
0.4
0.2

In [15]:
```
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 [16]:
```
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 [17]:
```
#from pandas import read_hdf
final_xtr = read_hdf('data/fea_sample/storage_sample_stage4.h5', 'train_df',mode='r')
final_xte = read_hdf('data/fea_sample/storage_sample_stage4.h5', 'test_df',mode='r')
```

In [18]:
```
# Reading list of edges
if os.path.isfile('data/after_eda/train_pos_after_eda.csv'):
    train_graph=nx.read_edgelist('data/after_eda/train_pos_after_eda.csv',delimiter=',',create_using=nx.DiGraph(),nodetype=int)
    print(nx.info(train_graph))
else:
    print("please run the FB_EDA.ipynb or download the files from drive")
```

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

In [19]:
```
in_degree = dict(train_graph.in_degree())
out_degree = dict(train_graph.out_degree())
```

In [20]:
```
def Prefer (x,y,typ):
    try:
        return typ.get(x)*typ.get(y)
    except:
        return 0
```

In [21]:
```
if not os.path.isfile('data/fea_sample/storage_sample_stage5.h5'):
    #mapping preferential followers to train and test data
    final_xtr['pref_followers'] = final_xtr.apply(lambda row:
                                            Prefer(row['source_node'],row['destination_node'],out_degree),axis=1)
    final_xte['pref_followers'] = final_xte.apply(lambda row:
                                            Prefer(row['source_node'],row['destination_node'],out_degree),axis=1)

    # mapping preferential followees to train and test data
    final_xtr['pref_followees'] = final_xtr.apply(lambda row:
                                            Prefer(row['source_node'],row['destination_node'],in_degree),axis=1)
    final_xte['pref_followees'] = final_xte.apply(lambda row:
                                            Prefer(row['source_node'],row['destination_node'],in_degree),axis=1)


    hdf = HDFStore('data/fea_sample/storage_sample_stage5.h5')
    hdf.put('train_df',final_xtr, format='table', data_columns=True)
    hdf.put('test_df',final_xte, format='table', data_columns=True)
    hdf.close()
else:
    final_xtr = read_hdf('data/fea_sample/storage_sample_stage5.h5', 'train_df',mode='r')
    final_xte = read_hdf('data/fea_sample/storage_sample_stage5.h5', 'test_df',mode='r')
```

SVD Dot

In [24]:
```
#For source & destination in the training set
sou_u=final_xtr[['svd_u_s_1','svd_u_s_2','svd_u_s_3','svd_u_s_4','svd_u_s_5','svd_u_s_6']]
sou_v=final_xtr[['svd_v_s_1','svd_v_s_2','svd_v_s_3','svd_v_s_4','svd_v_s_5','svd_v_s_6']]
des_u=final_xtr[['svd_u_d_1','svd_u_d_2','svd_u_d_3','svd_u_d_4','svd_u_d_5','svd_u_d_6']]
des_v=final_xtr[['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 [26]:
```
svd_dot_u=[]
svd_dot_v=[]
for i in range(len(sou_u)):
    svd_dot_u.append(np.dot(sou_u.values[i],des_u.values[i]))
    svd_dot_v.append(np.dot(sou_v.values[i],des_v.values[i]))
final_xtr['svd_dot_u']=svd_dot_u
final_xtr['svd_dot_v']=svd_dot_v
```

In [27]:
```
sou_u=final_xte[['svd_u_s_1','svd_u_s_2','svd_u_s_3','svd_u_s_4','svd_u_s_5','svd_u_s_6']]
sou_v=final_xte[['svd_v_s_1','svd_v_s_2','svd_v_s_3','svd_v_s_4','svd_v_s_5','svd_v_s_6']]
des_u=final_xte[['svd_u_d_1','svd_u_d_2','svd_u_d_3','svd_u_d_4','svd_u_d_5','svd_u_d_6']]
des_v=final_xte[['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 [28]:
```
svd_dot_u=[]
svd_dot_v=[]
for i in range(len(sou_u)):
    svd_dot_u.append(np.dot(sou_u.values[i],des_u.values[i]))
    svd_dot_v.append(np.dot(sou_v.values[i],des_v.values[i]))
final_xte['svd_dot_u']=svd_dot_u
final_xte['svd_dot_v']=svd_dot_v
```

In [29]:
```
final_xtr.head()
```

Out[29]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	num_followers_s	num_followees_s	num_followees_d	...	sv
0	273084	1505602	1	0	0.000000	0.000000	0.000000	6	15	8	...	-1.3
1	832016	1543415	1	0	0.187135	0.028382	0.343828	94	61	142	...	1.24
2	1325247	760242	1	0	0.369565	0.156957	0.566038	28	41	22	...	-1.3
3	1368400	1006992	1	0	0.000000	0.000000	0.000000	11	5	7	...	-9.8
4	140165	1708748	1	0	0.000000	0.000000	0.000000	1	11	3	...	0.000

5 rows × 58 columns

In [30]:
```
final_xte.head()
```

Out[30]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	num_followers_s	num_followees_s	num_followees_d	...	svd_
0	848424	784690	1	0	0.0	0.029161	0.000000	14	6	9	...	-9.99
1	483294	1255532	1	0	0.0	0.000000	0.000000	17	1	19	...	-9.36
2	626190	1729265	1	0	0.0	0.000000	0.000000	10	16	9	...	-4.25
3	947219	425228	1	0	0.0	0.000000	0.000000	37	10	34	...	-2.16
4	991374	975044	1	0	0.2	0.042767	0.347833	27	15	27	...	-8.74

5 rows × 58 columns

In [31]:
```
if not os.path.isfile('data/fea_sample/storage_sample_stage6.h5'):
    #SVD_dot for U
    final_xtr['svd_dot_U'] = final_xtr.apply(lambda row: svd_dot(row['source_node'], row['destination_node']), axis=1)
    final_xte['svd_dot_U'] = final_xte.apply(lambda row: svd_dot(row['svd_u_s_1'], row['svd_u_d_1']), axis=1)

    #SVD_dot for V
    final_xtr['svd_dot_2'] = final_xte.apply(lambda row: svd_dot(row['svd_u_s_2'], row['svd_u_d_2']), axis=1)
    final_xte['svd_dot_3'] = final_xte.apply(lambda row: svd_dot(row['svd_u_s_3'], row['svd_u_d_3']), axis=1)

    hdf = HDFStore('data/fea_sample/storage_sample_stage6.h5')
    hdf.put('train_df',final_xtr, format='table', data_columns=True)
    hdf.put('test_df',final_xte, format='table', data_columns=True)
    hdf.close()
else:
    final_xtr = read_hdf('data/fea_sample/storage_sample_stage6.h5', 'train_df',mode='r')
    final_xte = read_hdf('data/fea_sample/storage_sample_stage6.h5', 'test_df',mode='r')
```

In [32]:
```
final_xtr.head()
```

Out[32]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	num_followers_s	num_followees_s	num_followees_d	...	sv
0	273084	1505602	1	0	0.000000	0.000000	0.000000	6	15	8	...	9.77
1	832016	1543415	1	0	0.187135	0.028382	0.343828	94	61	142	...	2.60
2	1325247	760242	1	0	0.369565	0.156957	0.566038	28	41	22	...	1.62
3	1368400	1006992	1	0	0.000000	0.000000	0.000000	11	5	7	...	3.04
4	140165	1708748	1	0	0.000000	0.000000	0.000000	1	11	3	...	0.000

5 rows × 62 columns

In [33]:
```
y_train = final_xtr.indicator_link
y_test = final_xte.indicator_link
```

In [34]:
```
final_xtr.drop(['source_node', 'destination_node','indicator_link'],axis=1,inplace=True)
final_xte.drop(['source_node', 'destination_node','indicator_link'],axis=1,inplace=True)
```

Applying XGBoost with all above features

In [35]:
```
from sklearn.model_selection import RandomizedSearchCV
param_grid = {"max_depth":[1, 3, 5, 7, 10],
              "n_estimators":[10, 50, 100, 200, 500]}
model = RandomizedSearchCV(xgb.XGBClassifier(n_jobs=-1,random_state=92), param_distributions=param_grid,n_iter=10,scoring='f1',cv=3,n_jobs=-1)
model.fit(final_xtr, y_train)
model.best_params_
```

Out[35]:
```
{'n_estimators': 500, 'max_depth': 7}
```

In [36]:
```
train_auc = model.cv_results_['mean_train_score']
train_auc_std = model.cv_results_['std_train_score']
cv_auc = model.cv_results_['mean_test_score']
cv_auc_std = model.cv_results_['std_test_score']
```

In [37]:
```
#Results of grid Search
best_params = model.best_params_
print(model.best_score_)
print(model.best_params_)
```

```
0.9824502700489093
{'n_estimators': 500, 'max_depth': 7}
```

In [38]:
```
model.cv_results_
```

Out[38]:
```
{'mean_fit_time': array([ 15.6524872 ,  25.3045462 , 185.03030229,   2.58176525,
          1.83110611,  66.9736588 ,  51.02794528,  75.58331362,
         33.50311955,  31.35652343]),
 'std_fit_time': array([1.99229989, 0.05207163, 2.22103943, 0.0326367 , 0.0342305 ,
        1.44286924, 1.80848166, 0.61451051, 0.97423891, 1.77058118]),
 'mean_score_time': array([0.18085011, 0.20312381, 0.42951918, 0.15857633, 0.1486028 ,
        0.22174096, 0.21243231, 0.27193999, 0.19281777, 0.11967993]),
 'std_score_time': array([0.02785769, 0.02200701, 0.01186606, 0.02850114, 0.01494909,
        0.02450158, 0.01000664, 0.00204947, 0.0223655 , 0.02447071]),
 'param_n_estimators': masked_array(data=[50, 100, 500, 10, 10, 200, 100, 500, 500, 100],
              mask=[False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'param_max_depth': masked_array(data=[7, 5, 7, 5, 3, 7, 10, 3, 1, 7],
              mask=[False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'params': [{'n_estimators': 50, 'max_depth': 7},
  {'n_estimators': 100, 'max_depth': 5},
  {'n_estimators': 500, 'max_depth': 7},
  {'n_estimators': 10, 'max_depth': 5},
  {'n_estimators': 10, 'max_depth': 3},
  {'n_estimators': 200, 'max_depth': 7},
  {'n_estimators': 100, 'max_depth': 10},
  {'n_estimators': 500, 'max_depth': 3},
  {'n_estimators': 500, 'max_depth': 1},
  {'n_estimators': 100, 'max_depth': 7}],
 'split0_test_score': array([0.97489209, 0.97649877, 0.98334187, 0.92986368, 0.92160478,
        0.98158252, 0.98030449, 0.98062413, 0.97026633, 0.97859087]),
 'split1_test_score': array([0.97413741, 0.97544988, 0.98181709, 0.93065428, 0.91810193,
        0.97979889, 0.97837985, 0.97863196, 0.96870823, 0.97707242]),
 'split2_test_score': array([0.97369061, 0.97448515, 0.98219182, 0.92984895, 0.92152109,
        0.9800489 , 0.97843031, 0.97908481, 0.9689275 , 0.97662448]),
 'mean_test_score': array([0.97424005, 0.97547795, 0.98245027, 0.9301223 , 0.92040927,
        0.98047678, 0.97903824, 0.97944698, 0.9693007 , 0.97742927]),
 'std_test_score': array([0.00049584, 0.0008223 , 0.00064876, 0.00037621, 0.00163189,
        0.00078852, 0.00089564, 0.00085267, 0.00068866, 0.0008415 ]),
 'rank_test_score': array([ 7,  6,  1,  9, 10,  2,  4,  3,  8,  5]),
 'split0_train_score': array([0.97426381, 0.977172  , 1.        , 0.93098842, 0.92287905,
        0.99414941, 0.99163136, 0.98431011, 0.96923521, 0.98192935]),
 'split1_train_score': array([0.97502774, 0.97740439, 1.        , 0.92887771, 0.9174488 ,
        0.99419584, 0.99254462, 0.98416197, 0.96956357, 0.98309493]),
 'split2_train_score': array([0.9757446 , 0.97715274, 1.        , 0.93056959, 0.92167531,
        0.99384768, 0.99332962, 0.98468879, 0.96976508, 0.98242749]),
 'mean_train_score': array([0.97501205, 0.97724304, 1.        , 0.93014524, 0.92066772,
        0.99406431, 0.99250187, 0.98438696, 0.96952129, 0.98248392]),
```

```
'std_train_score': array([0.00060463, 0.00011436, 0.        , 0.00091245, 0.00232857,
       0.00015435, 0.00069397, 0.00022183, 0.00021837, 0.00047752])}
```

In [39]:
```
#https://towardsdatascience.com/using-3d-visualizations-to-tune-hyperparameters-of-ml-models-with-python-b
#https://github.com/xoelop/Medium-posts/blob/master/3d%20cross%20validation/ML%206%20-%20Gridsearch%20visu
#https://qiita.com/bmj0114/items/8009f282c99b77780563
#Saving the obtained results from gridsearch in two dimensional array as dataframe
results = pd.DataFrame(model.cv_results_)
results.head()
```

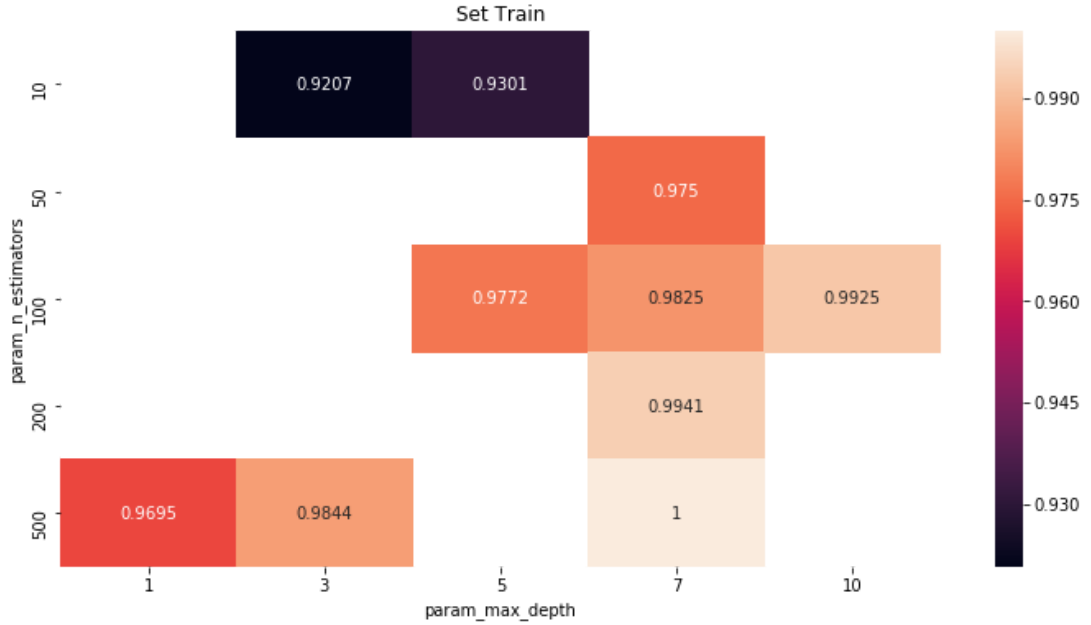
Out[39]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_estimators	param_max_depth	params	split0_test_score	split1_test_score	split2_test_score	mean_test_
0	15.652487	1.992300	0.180850	0.027858	50	7	{'n_estimators': 50, 'max_depth': 7}	0.974892	0.974137	0.973691	0.97
1	25.304546	0.052072	0.203124	0.022007	100	5	{'n_estimators': 100, 'max_depth': 5}	0.976499	0.975450	0.974485	0.97
2	185.030302	2.221039	0.429519	0.011866	500	7	{'n_estimators': 500, 'max_depth': 7}	0.983342	0.981817	0.982192	0.98
3	2.581765	0.032637	0.158576	0.028501	10	5	{'n_estimators': 10, 'max_depth': 5}	0.929864	0.930654	0.929849	0.93
4	1.831106	0.034231	0.148603	0.014949	10	3	{'n_estimators': 10, 'max_depth': 3}	0.921605	0.918102	0.921521	0.92

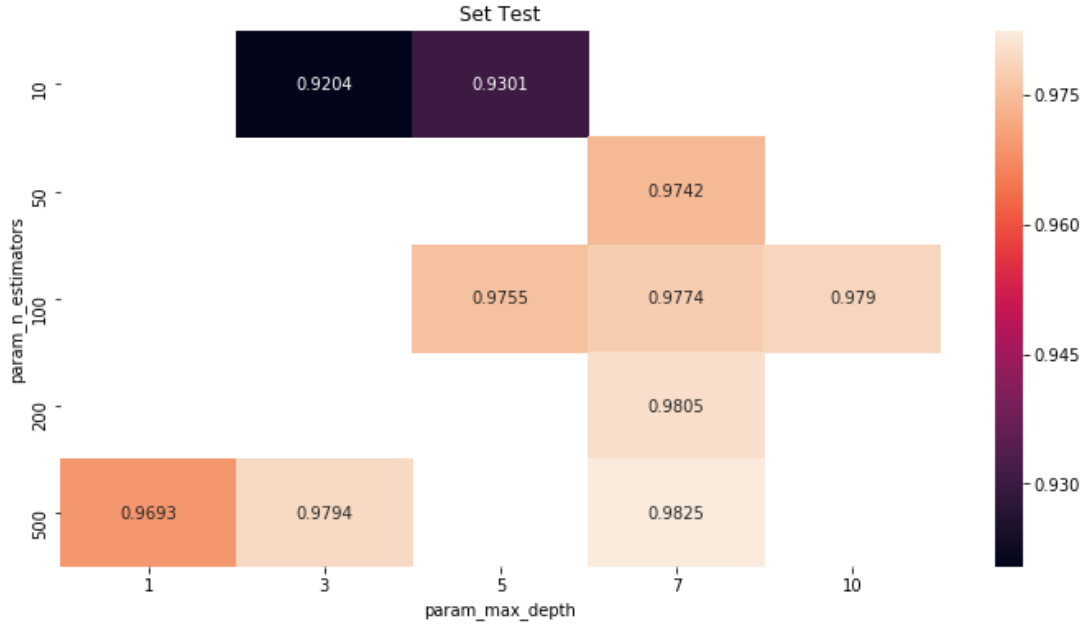
In [40]:
```
#https://github.com/xoelop/Medium-posts/blob/master/3d%20cross%20validation/ML%206%20-%20Gridsearch%20visulizations%20.ipynb
best_scores = results.groupby(['param_n_estimators', 'param_max_depth']).max().unstack()[['mean_test_score','mean_train_score']]
print(best_scores)
```

```
                   mean_test_score                                        \
param_max_depth                  1         3         5         7         10
param_n_estimators
10                             NaN  0.920409  0.930122       NaN       NaN
50                             NaN       NaN       NaN  0.974240       NaN
100                            NaN       NaN  0.975478  0.977429  0.979038
200                            NaN       NaN       NaN  0.980477       NaN
500                       0.969301  0.979447       NaN  0.982450       NaN

                   mean_train_score
param_max_depth                  1         3         5         7         10
param_n_estimators
10                             NaN  0.920668  0.930145       NaN       NaN
50                             NaN       NaN       NaN  0.975012       NaN
100                            NaN       NaN  0.977243  0.982484  0.992502
200                            NaN       NaN       NaN  0.994064       NaN
500                       0.969521  0.984387       NaN  1.000000       NaN
```

In [41]:
```
#https://github.com/xoelop/Medium-posts/blob/master/3d%20cross%20validation/ML%206%20-%20Gridsearch%20visulizations%20.ipynb
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (12, 6)
title = 'Set Train'
fmt = 'png'
sns.heatmap(best_scores.mean_train_score, annot=True, fmt='.4g');
plt.title(title);
```



In [42]:
```
#https://github.com/xoelop/Medium-posts/blob/master/3d%20cross%20validation/ML%206%20-%20Gridsearch%20visulizations%20.ipynb
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (12, 6)
title = 'Set Test'
fmt = 'png'
sns.heatmap(best_scores.mean_test_score, annot=True, fmt='.4g');
plt.title(title);
```



In [47]:
```
clf=xgb.XGBClassifier(n_jobs=-1,random_state=42,max_depth=7,n_estimators=50)
clf.fit(final_xtr, y_train)
y_pred_test=clf.predict(final_xte)
y_pred_train=clf.predict(final_xtr)
```

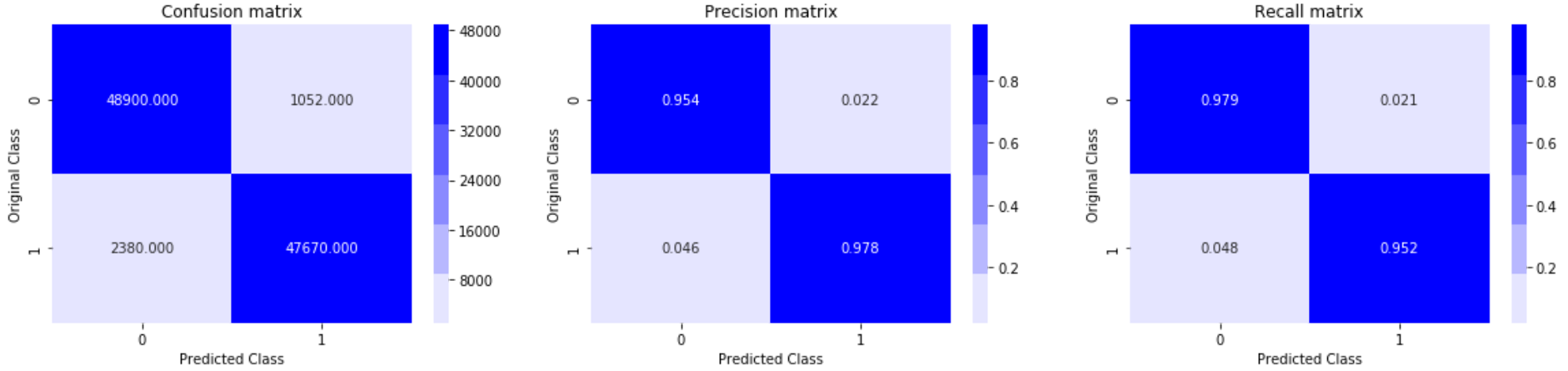
```
C:\Users\Himanshu Pc\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:151: DeprecationWarning: The truth value of an empty array is ambiguo
us. Returning False, but in future this will result in an error. Use `array.size > 0` to check that an array is not empty.
  if diff:
C:\Users\Himanshu Pc\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:151: DeprecationWarning: The truth value of an empty array is ambiguo
us. Returning False, but in future this will result in an error. Use `array.size > 0` to check that an array is not empty.
  if diff:
```

In [48]:
```
from sklearn.metrics import f1_score
print('Train f1 score',f1_score(y_train,y_pred_train))
print('Test f1 score',f1_score(y_test,y_pred_test))
```

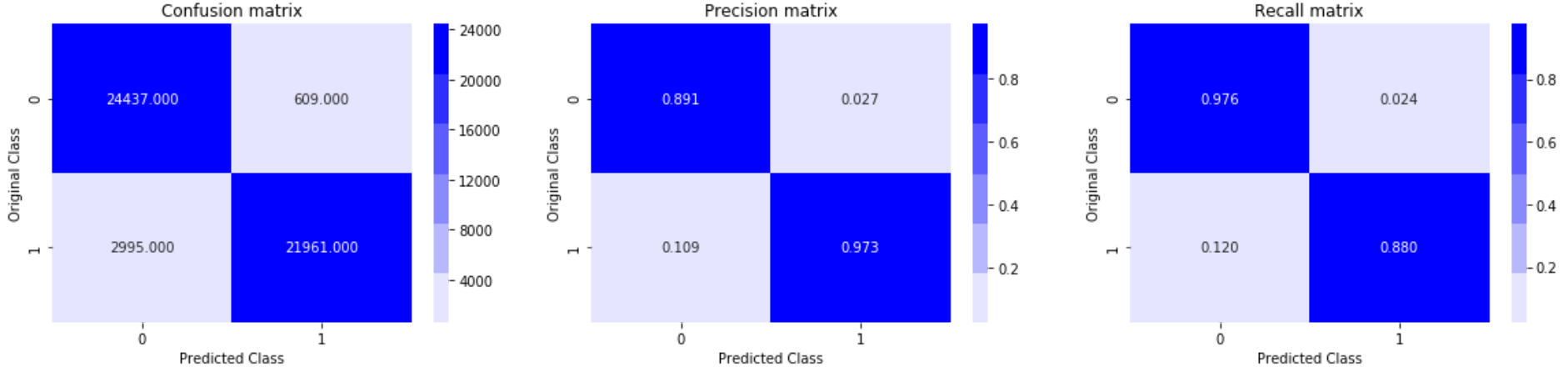
```
Train f1 score 0.9747258760818014
Test f1 score 0.9314958296286888
```

In [49]:
```
print('Train confusion_matrix')
plot_confusion_matrix(y_train,y_train_pred)
print('Test confusion_matrix')
plot_confusion_matrix(y_test,y_test_pred)
```

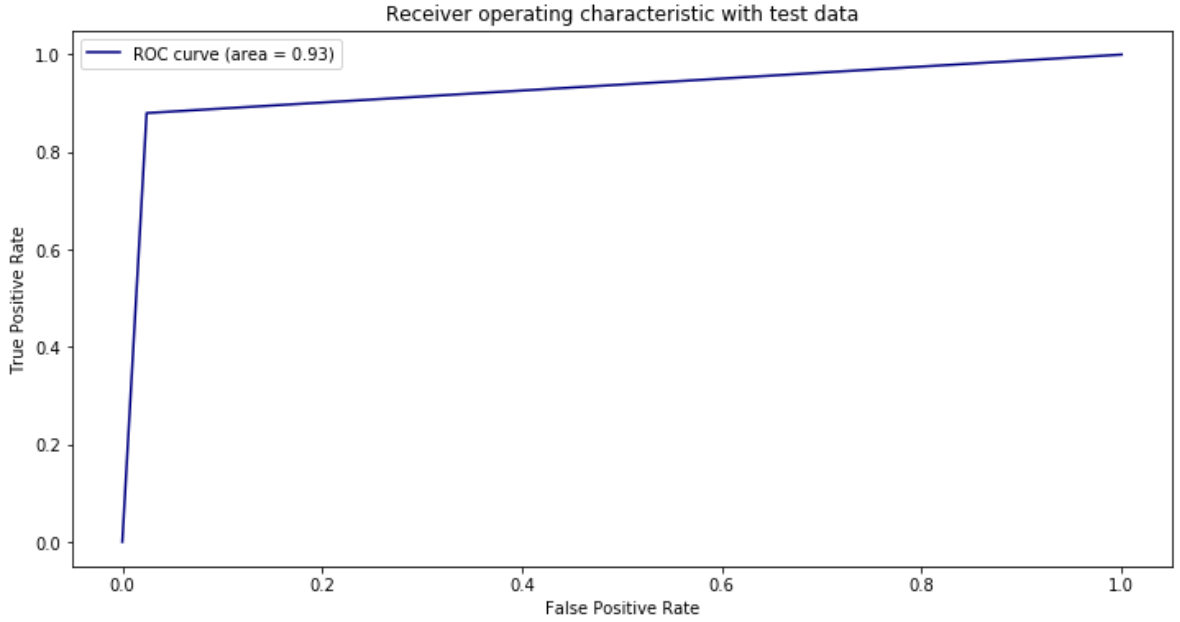
```
Train confusion_matrix
```



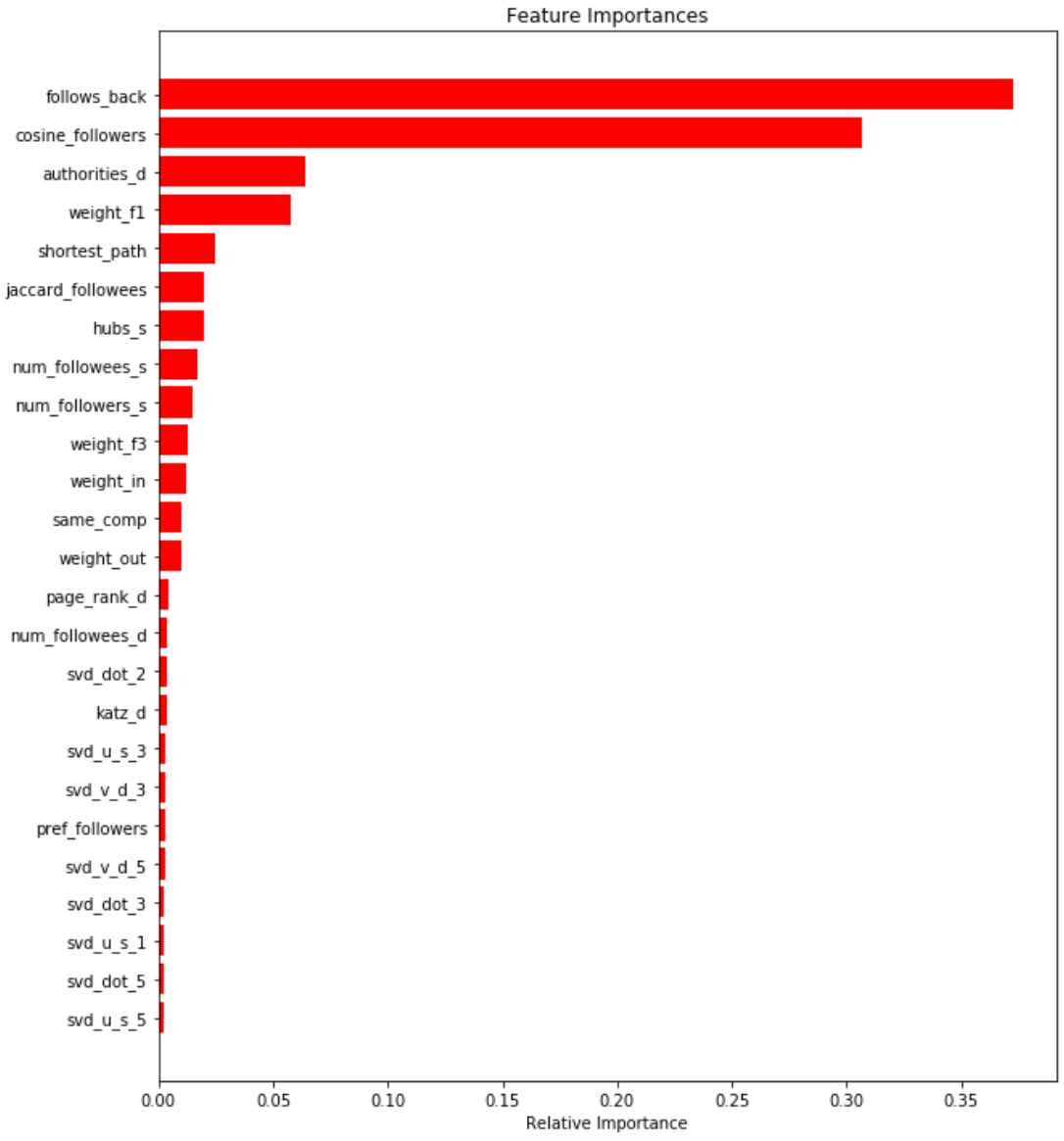

```
Test confusion_matrix
```



In [50]:
```
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 [51]:
```
features = final_xtr.columns
importances = clf.feature_importances_
indices = (np.argsort(importances))[-25:]
plt.figure(figsize=(10,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='r', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```


Feature Importances
follows_back
cosine_followers
authorities_d
weight_f1
shortest_path
jaccard_followees
hubs_s
num_followees_s
num_followers_s
weight_f3
weight_in
same_comp
weight_out
page_rank_d
num_followees_d
svd_dot_2
katz_d
svd_u_s_3
svd_v_d_3
pref_followers
svd_v_d_5
svd_dot_3
svd_u_s_1
svd_dot_5
svd_u_s_5
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
Relative Importance

In [52]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Sr. No.", "Model Name", "n_estimators", "max_depth", "Train F1 Score", "Test F1 Score"]
x.add_row(["1", 'XGBOOST', '50', '7', '0.974', '0.931'])
print(x)
```

```
+---------+------------+--------------+-----------+----------------+---------------+
| Sr. No. | Model Name | n_estimators | max_depth | Train F1 Score | Test F1 Score |
+---------+------------+--------------+-----------+----------------+---------------+
|    1    |  XGBOOST   |      50      |     7     |     0.974      |     0.931     |
+---------+------------+--------------+-----------+----------------+---------------+
```

Observations :-

1. We started with just two features in our dataset & then we performed numerous feature engineering on the given dataset to get more features.
2. We obtained features like Jaccard distance, Katz similarity, shortest distance, page rank etc.
3. Later we also obtained some advanced features like preferential attachment & svd dot.
4. On inculcating these features in our dataset we were able to improve our model slightly.
5. On applying Hyperparameter Tuned XGBoost along with the above features we were able to get a F1-score of 0.974 & 0.931 on our train & test data respectively.

In [ ]: