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
```

In [2]:

```
#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 [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_inde
x',
       '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(['indicator_link'],axis=1,inplace=True)
df_final_test.drop(['indicator_link'],axis=1,inplace=True)
```

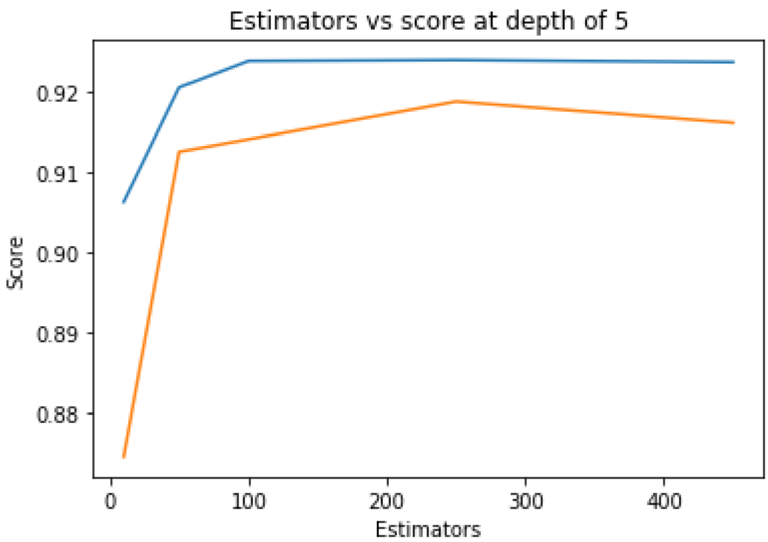
In [6]:

```
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,ver
bose=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 0.8745605278006
858
Estimators =  50 Train Score 0.9205725512208812 test Score 0.9125653355634
538
Estimators =  100 Train Score 0.9238690848446947 test Score 0.914119971415
3599
Estimators =  250 Train Score 0.9239789348046863 test Score 0.918800723266
4732
Estimators =  450 Train Score 0.9237190618658074 test Score 0.916150768582
8595
```

Out[6]:

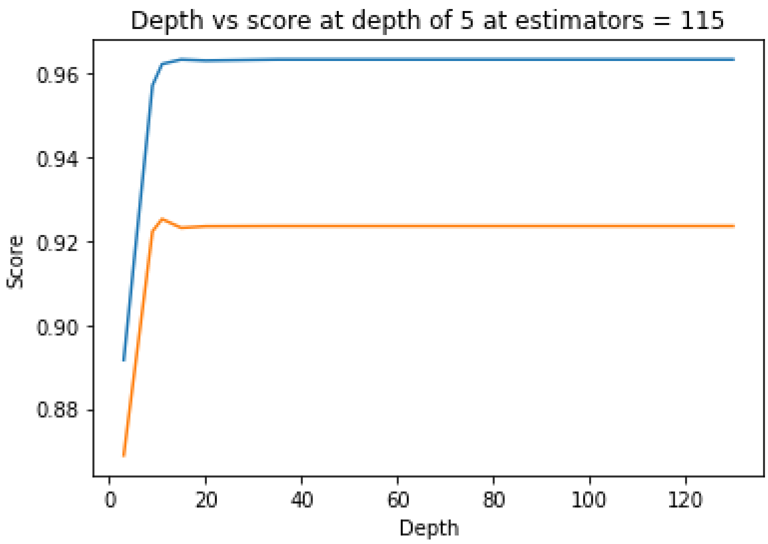
```
Text(0.5, 1.0, 'Estimators vs score at depth of 5')
```



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,v
erbose=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 [30]:

```
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,refit=True,n_iter=5,
cv=10,scoring='f1',random_state=25)

rf_random.fit(df_final_train,y_train)
```

Out[30]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=RandomForestClassifier(bootstrap=True,
                                                    ccp_alpha=0.0,
                                                    class_weight=None,
                                                    criterion='gini',
                                                    max_depth=None,
                                                    max_features='auto',
                                                    max_leaf_nodes=None,
                                                    max_samples=None,
                                                    min_impurity_decrease=
0.0,
                                                    min_impurity_split=Non
e,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
                                                    min_weight_fraction_le
af=0.0,
                                                    n_estimators=100, n_jo
b...
                                        'min_samples_leaf': <scipy.stats._
distn_infrastructure.rv_frozen object at 0x0000028D19C816D8>,
                                        'min_samples_split': <scipy.stats.
_distn_infrastructure.rv_frozen object at 0x0000028D1429F7B8>,
                                        'n_estimators': <scipy.stats._dist
n_infrastructure.rv_frozen object at 0x0000028D1429F898>},
                   pre_dispatch='2*n_jobs', random_state=25, refit=True,
                   return_train_score=False, scoring='f1', verbose=0)
```

In [32]:

```
#print('mean train scores',rf_random.cv_results_['mean_train_score'])
print('mean test scores',rf_random.cv_results_['mean_test_score'])
```

```
mean test scores [0.96225042 0.96215492 0.9605708  0.96194014 0.96330005]
```

In [33]:

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

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=14, max_features='aut
o',
                       max_leaf_nodes=None, max_samples=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, verbos
e=0,
                       warm_start=False)
```

In [34]:

```
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 [35]:

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

In [36]:

```
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
```

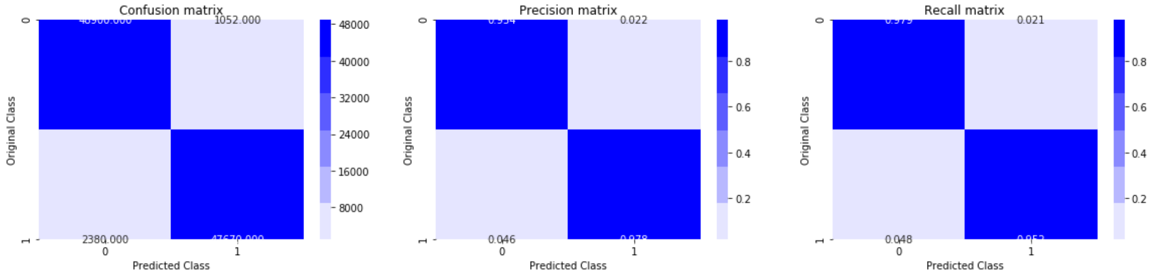
In [19]:

```
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=la
bels)
    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=la
bels)
    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=la
bels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()
```

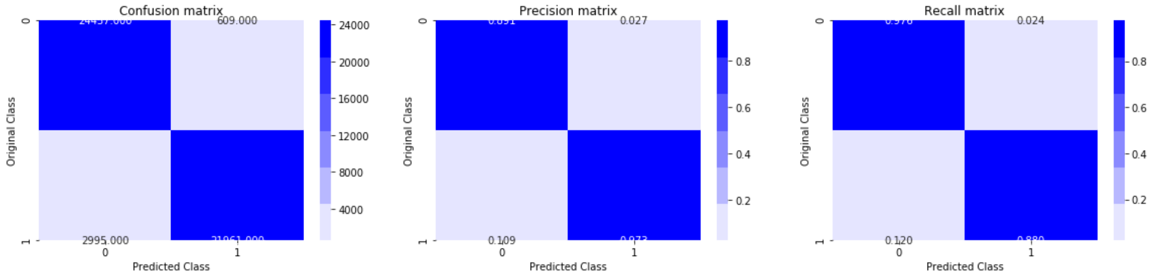
In [38]:

```python
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

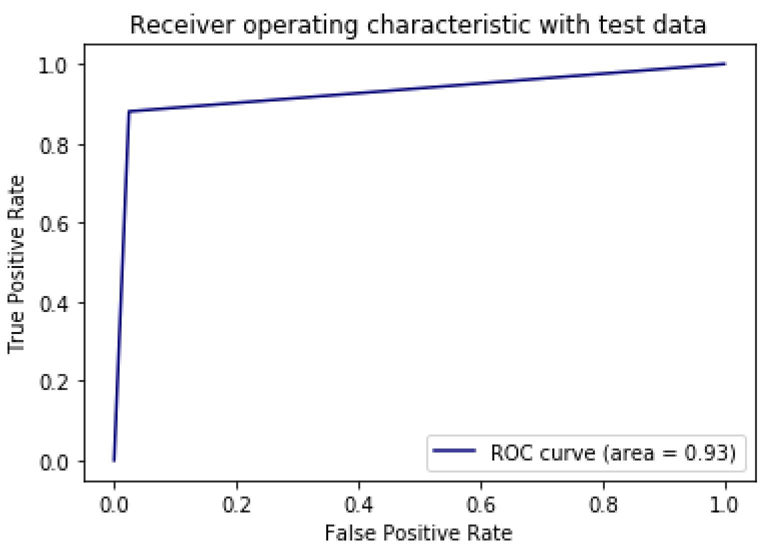


Test confusion_matrix



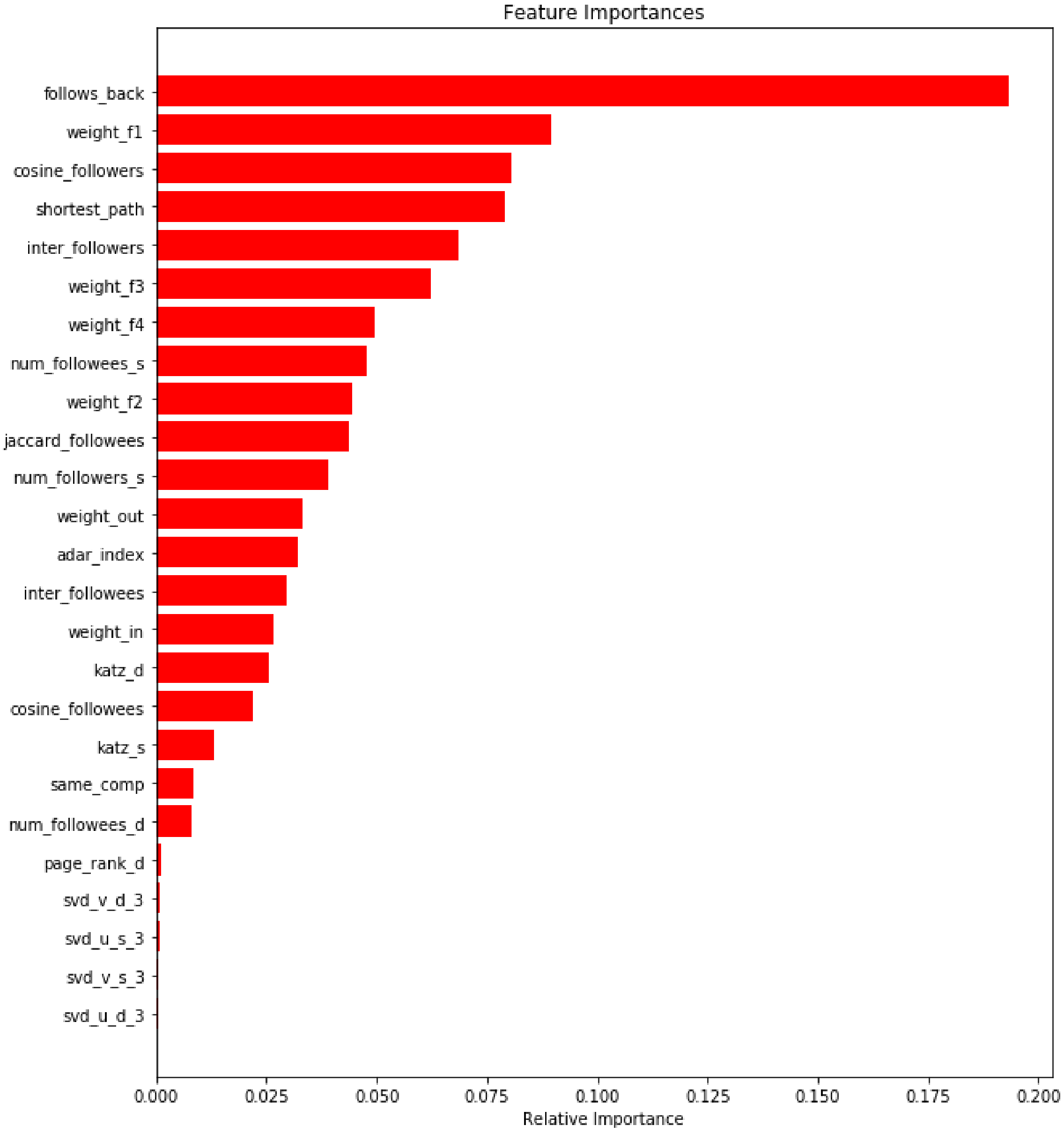
In [39]:

```python
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 [40]:

```
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.

ADDING MORE FEATURES:

In [6]:

```
df_final_train.head()
```

Out[6]:

	source_node	destination_node	jaccard_followers	jaccard_followees	cosine_followers	co
0	273084	1505602	0	0.000000	0.000000	
1	832016	1543415	0	0.187135	0.028382	
2	1325247	760242	0	0.369565	0.156957	
3	1368400	1006992	0	0.000000	0.000000	
4	140165	1708748	0	0.000000	0.000000	

5 rows × 53 columns

In [7]:

```
df_final_train.columns
```

Out[7]:

```
Index(['source_node', 'destination_node', '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_f
1',
       '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 [8]:

```
def followee_preferential_attachment(user1,user2):
    try:
        user_1 = len(set(train_graph.successors(user1)))
        user_2 = len(set(train_graph.successors(user2)))
        return(user_1*user_2)
    except:
        return(0)

def follower_preferential_attachment(user1,user2):
    try:
        user_1 = len(set(train_graph.predecessors(user1)))
        user_2 = len(set(train_graph.predecessors(user2)))
        return(user_1*user_2)
    except:
        return(0)
```

In [9]:

```
df_final_train['followee_preferential_attachment'] = df_final_train.apply(lambda row: f
ollowee_preferential_attachment(row['source_node'],row['destination_node']),axis=1)
df_final_test['followee_preferential_attachment'] = df_final_test.apply(lambda row: fol
lowee_preferential_attachment(row['source_node'],row['destination_node']),axis=1)

df_final_train['follower_preferential_attachment'] = df_final_train.apply(lambda row: f
ollower_preferential_attachment(row['source_node'],row['destination_node']),axis=1)
df_final_test['follower_preferential_attachment'] = df_final_test.apply(lambda row: fol
lower_preferential_attachment(row['source_node'],row['destination_node']),axis=1)
```

In [10]:

```
df_final_train.head()
```

Out[10]:

	source_node	destination_node	jaccard_followers	jaccard_followees	cosine_followers	cos
0	273084	1505602	0	0.000000	0.000000	
1	832016	1543415	0	0.187135	0.028382	
2	1325247	760242	0	0.369565	0.156957	
3	1368400	1006992	0	0.000000	0.000000	
4	140165	1708748	0	0.000000	0.000000	

5 rows × 55 columns

In [11]:

```
df_final_test.head()
```

Out[11]:

	source_node	destination_node	jaccard_followers	jaccard_followees	cosine_followers	cos
0	848424	784690	0	0.0	0.029161	
1	483294	1255532	0	0.0	0.000000	
2	626190	1729265	0	0.0	0.000000	
3	947219	425228	0	0.0	0.000000	
4	991374	975044	0	0.2	0.042767	

5 rows × 55 columns

Add feature called svd_dot. you can calculate svd_dot as Dot product between sourse node svd and destination node svd features.

In [12]:

```
#for train data
u1 = list(df_final_train['svd_u_s_1'])
u2 = list(df_final_train['svd_u_s_2'])
u3 = list(df_final_train['svd_u_s_3'])
u4 = list(df_final_train['svd_u_s_4'])
u5 = list(df_final_train['svd_u_s_5'])
u6 = list(df_final_train['svd_u_s_6'])

u7 = list(df_final_train['svd_u_d_1'])
u8 = list(df_final_train['svd_u_d_2'])
u9 = list(df_final_train['svd_u_d_3'])
u10 = list(df_final_train['svd_u_d_4'])
u11 = list(df_final_train['svd_u_d_5'])
u12 = list(df_final_train['svd_u_d_6'])

y1 = list(df_final_train['svd_v_s_1'])
y2 = list(df_final_train['svd_v_s_2'])
y3 = list(df_final_train['svd_v_s_3'])
y4 = list(df_final_train['svd_v_s_4'])
y5 = list(df_final_train['svd_v_s_5'])
y6 = list(df_final_train['svd_v_s_6'])

y7 = list(df_final_train['svd_v_d_1'])
y8 = list(df_final_train['svd_v_d_2'])
y9 = list(df_final_train['svd_v_d_3'])
y10 = list(df_final_train['svd_v_d_4'])
y11 = list(df_final_train['svd_v_d_5'])
y12 = list(df_final_train['svd_v_d_6'])

print(np.shape(u1))
print(np.shape(u2))
print(np.shape(u3))
print(np.shape(u4))
print(np.shape(u5))
print(np.shape(u6))
print(np.shape(u7))
print(np.shape(u8))
print(np.shape(u9))
print(np.shape(u10))
print(np.shape(u11))
print(np.shape(u12))

print(np.shape(y1))
print(np.shape(y2))
print(np.shape(y3))
print(np.shape(y4))
print(np.shape(y5))
print(np.shape(y6))
print(np.shape(y7))
print(np.shape(y8))
print(np.shape(y9))
print(np.shape(y10))
print(np.shape(y11))
print(np.shape(y12))

train_u_source = []
train_u_destination = []
train_v_source = []
train_v_destination = []
```

```
train_u_s_dot = []
train_u_d_dot = []

for i in range(0,len(u1)):
    train_u_source.append(u1[i])
    train_u_source.append(u2[i])
    train_u_source.append(u3[i])
    train_u_source.append(u4[i])
    train_u_source.append(u5[i])
    train_u_source.append(u6[i])
    train_u_destination.append(u7[i])
    train_u_destination.append(u8[i])
    train_u_destination.append(u9[i])
    train_u_destination.append(u10[i])
    train_u_destination.append(u11[i])
    train_u_destination.append(u12[i])

    dot_product = np.dot(train_u_source[i],train_u_destination[i])
    train_u_s_dot.append(dot_product)


for j in range(0,len(y1)):
    train_v_source.append(y1[j])
    train_v_source.append(y2[j])
    train_v_source.append(y3[j])
    train_v_source.append(y4[j])
    train_v_source.append(y5[j])
    train_v_source.append(y6[j])

    train_v_destination.append(y7[j])
    train_v_destination.append(y8[j])
    train_v_destination.append(y9[j])
    train_v_destination.append(y10[j])
    train_v_destination.append(y11[j])
    train_v_destination.append(y12[j])

    dot_product = np.dot(train_v_source[j],train_v_destination[j])
    train_u_d_dot.append(dot_product)

print("********************************************")
print(np.shape(train_u_s_dot))
print(np.shape(train_u_d_dot))
```

```
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
*******************************************
(100002,)
(100002,)
```

In [13]:

```
#for test data
u1 = list(df_final_test['svd_u_s_1'])
u2 = list(df_final_test['svd_u_s_2'])
u3 = list(df_final_test['svd_u_s_3'])
u4 = list(df_final_test['svd_u_s_4'])
u5 = list(df_final_test['svd_u_s_5'])
u6 = list(df_final_test['svd_u_s_6'])

u7 = list(df_final_test['svd_u_d_1'])
u8 = list(df_final_test['svd_u_d_2'])
u9 = list(df_final_test['svd_u_d_3'])
u10 = list(df_final_test['svd_u_d_4'])
u11 = list(df_final_test['svd_u_d_5'])
u12 = list(df_final_test['svd_u_d_6'])

y1 = list(df_final_test['svd_v_s_1'])
y2 = list(df_final_test['svd_v_s_2'])
y3 = list(df_final_test['svd_v_s_3'])
y4 = list(df_final_test['svd_v_s_4'])
y5 = list(df_final_test['svd_v_s_5'])
y6 = list(df_final_test['svd_v_s_6'])

y7 = list(df_final_test['svd_v_d_1'])
y8 = list(df_final_test['svd_v_d_2'])
y9 = list(df_final_test['svd_v_d_3'])
y10 = list(df_final_test['svd_v_d_4'])
y11 = list(df_final_test['svd_v_d_5'])
y12 = list(df_final_test['svd_v_d_6'])

print(np.shape(u1))
print(np.shape(u2))
print(np.shape(u3))
print(np.shape(u4))
print(np.shape(u5))
print(np.shape(u6))
print(np.shape(u7))
print(np.shape(u8))
print(np.shape(u9))
print(np.shape(u10))
print(np.shape(u11))
print(np.shape(u12))

print(np.shape(y1))
print(np.shape(y2))
print(np.shape(y3))
print(np.shape(y4))
print(np.shape(y5))
print(np.shape(y6))
print(np.shape(y7))
print(np.shape(y8))
print(np.shape(y9))
print(np.shape(y10))
print(np.shape(y11))
print(np.shape(y12))

test_u_source = []
test_u_destination = []
test_v_source = []
test_v_destination = []
```

```
test_u_s_dot = []
test_u_d_dot = []

for i in range(0,len(u1)):
    test_u_source.append(u1[i])
    test_u_source.append(u2[i])
    test_u_source.append(u3[i])
    test_u_source.append(u4[i])
    test_u_source.append(u5[i])
    test_u_source.append(u6[i])
    test_u_destination.append(u7[i])
    test_u_destination.append(u8[i])
    test_u_destination.append(u9[i])
    test_u_destination.append(u10[i])
    test_u_destination.append(u11[i])
    test_u_destination.append(u12[i])

    dot_product = np.dot(test_u_source[i],test_u_destination[i])
    test_u_s_dot.append(dot_product)


for j in range(0,len(y1)):
    test_v_source.append(y1[j])
    test_v_source.append(y2[j])
    test_v_source.append(y3[j])
    test_v_source.append(y4[j])
    test_v_source.append(y5[j])
    test_v_source.append(y6[j])

    test_v_destination.append(y7[j])
    test_v_destination.append(y8[j])
    test_v_destination.append(y9[j])
    test_v_destination.append(y10[j])
    test_v_destination.append(y11[j])
    test_v_destination.append(y12[j])

    dot_product = np.dot(test_v_source[j],test_v_destination[j])
    test_u_d_dot.append(dot_product)

print("********************************************")
print(np.shape(test_u_s_dot))
print(np.shape(test_u_d_dot))
```

```
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
*******************************************
(50002,)
(50002,)
```

In [14]:

```
df_final_train['source_svd'] = np.array(train_u_s_dot)
df_final_train['dest_svd'] = np.array(train_u_d_dot)
df_final_test['source_svd'] = np.array(test_u_s_dot)
df_final_test['dest_svd'] = np.array(test_u_d_dot)
```

In [15]:

```
df_final_train[0:5]
```

Out[15]:

	source_node	destination_node	jaccard_followers	jaccard_followees	cosine_followers	cos
0	273084	1505602	0	0.000000	0.000000	
1	832016	1543415	0	0.187135	0.028382	
2	1325247	760242	0	0.369565	0.156957	
3	1368400	1006992	0	0.000000	0.000000	
4	140165	1708748	0	0.000000	0.000000	

5 rows × 57 columns

APPLYING XGBOOST ON ALL THESE FEATURES

In [16]:

```
df_final_train = df_final_train.drop(['source_node','destination_node'],axis=1)
df_final_test = df_final_test.drop(['source_node','destination_node'],axis=1)
```

In [17]:

```
print(df_final_train.shape,y_train.shape)
print(df_final_test.shape,y_test.shape)
```

```
(100002, 55) (100002,)
(50002, 55) (50002,)
```

HYPERPARAMETER TUNING OF XGBOOST

In [69]:

```
import xgboost as xgb
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import f1_score,make_scorer

params = {
         'min_child_weight':[1,3,4,6],
         'max_depth':[2,3,5,7,9],
         'n_estimators':[50,100,150,200,250,300],
         'learning_rate':[0.1,0.2,0.3]
         }

clf = xgb.XGBClassifier()
model = RandomizedSearchCV(clf, params, cv = 3)
model.fit(df_final_train,y_train)
print(model.best_estimator_)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.2, max_delta_step=0, max_depth=5,
              min_child_weight=3, missing=None, n_estimators=300, n_jobs=
1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

In [18]:

```
XGB = xgb.XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.2, max_delta_step=0, max_depth=5,
              min_child_weight=3, missing=None, n_estimators=300, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
XGB.fit(df_final_train,y_train)
y_train_pred = XGB.predict(df_final_train)
y_test_pred = XGB.predict(df_final_test)

train_f1 = f1_score(y_train,y_train_pred)
test_f1 = f1_score(y_test,y_test_pred)

print(' Train f1 Score :',train_f1)
print(' Test f1 Score :',test_f1)
```

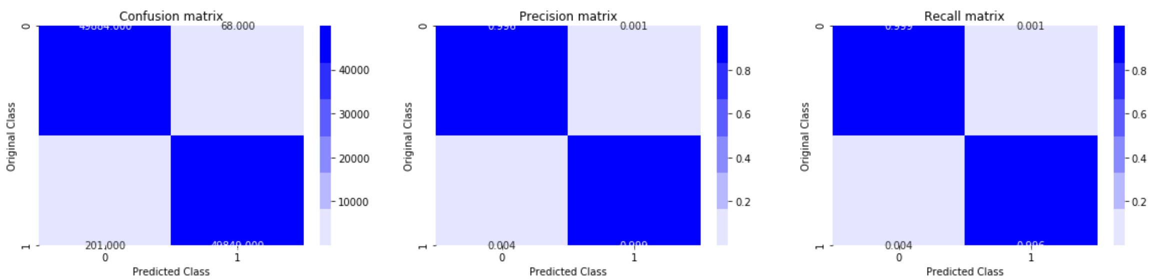
```
 Train f1 Score : 0.9973091120069623
 Test f1 Score : 0.9158473230003006
```

CONFUSION MATRICES

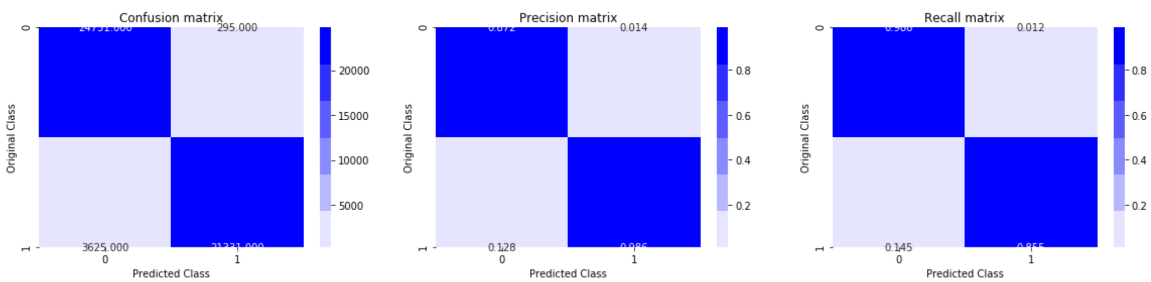
In [20]:

```
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
```



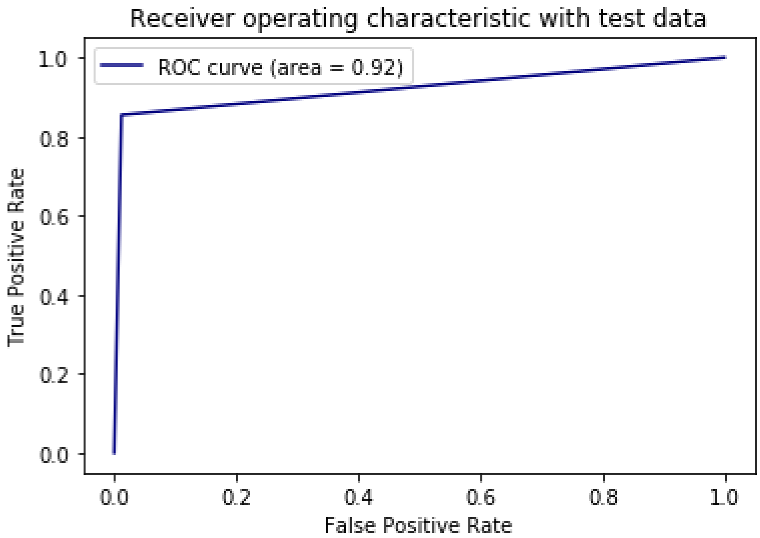

```
Test confusion_matrix
```



AUC CURVE

In [21]:

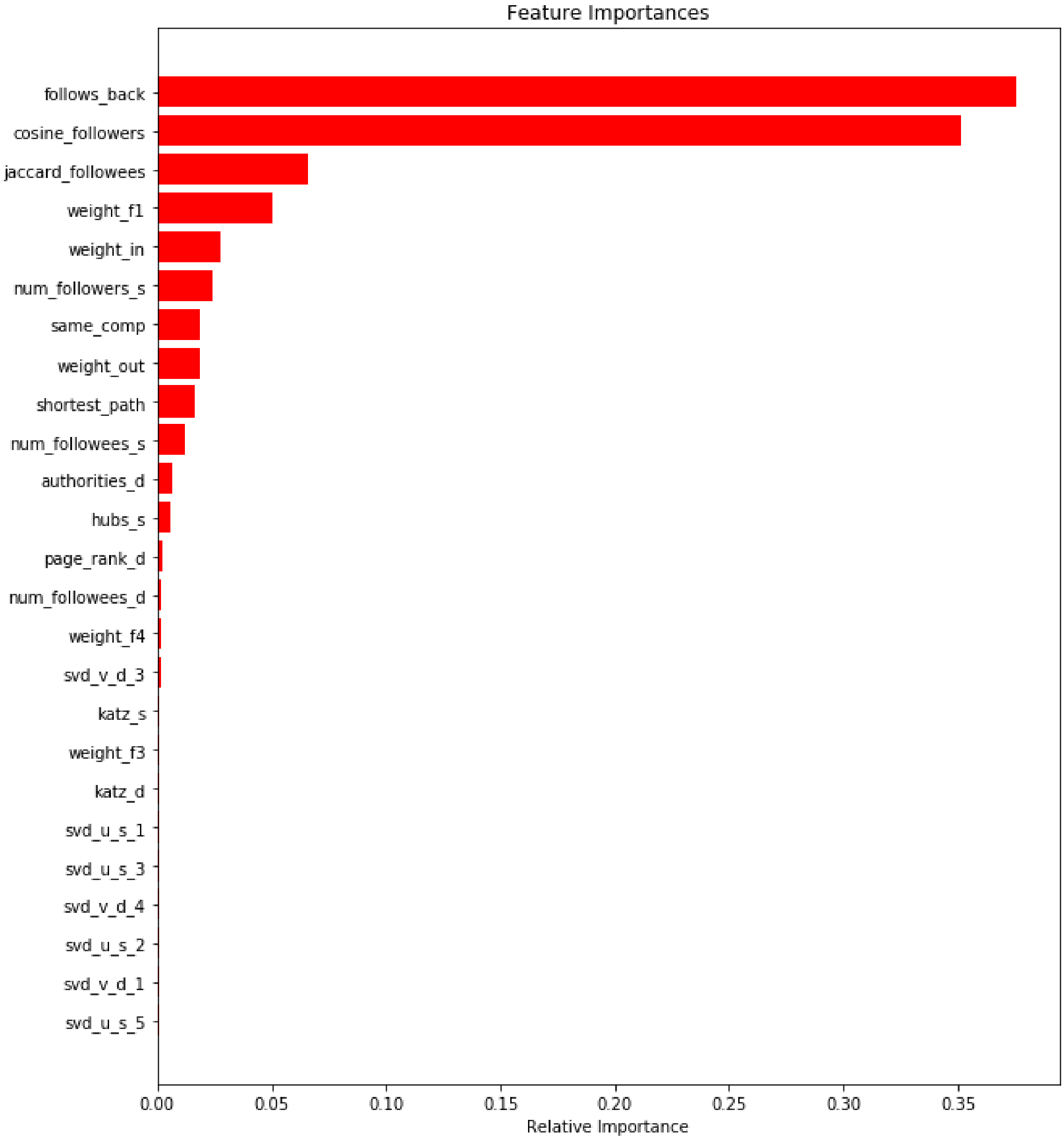
```
from sklearn.metrics import roc_curve, auc
fpr,tpr,ths = roc_curve(y_test,y_test_pred)
auc_sc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='navy',label='ROC curve (area = %0.2f)' % auc_sc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic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



FEATURE IMPORTANCE

In [22]:

```
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
jaccard_followees
weight_f1
weight_in
num_followers_s
same_comp
weight_out
shortest_path
num_followees_s
authorities_d
hubs_s
page_rank_d
num_followees_d
weight_f4
svd_v_d_3
katz_s
weight_f3
katz_d
svd_u_s_1
svd_u_s_3
svd_v_d_4
svd_u_s_2
svd_v_d_1
svd_u_s_5
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
Relative Importance

CONCLUSION

We added two more features preferential attachment with followers and followes and dot product between svd of source and destination nodes.we hyperparameter tuned the xgboost model and we got train f1 score 0.997 and test f1 score as 0.915. we got test auc of 0.92.

In [ ]:

```

```