Social network Graph Link Prediction - Facebook Challenge

In [49]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

In [0]:

```
#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 [52]:

```
!wget --header="Host: doc-00-b0-docs.googleusercontent.com" --header="User-Agent: Mozilla/5.0
(Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.122
Safari/537.36" --header="Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/s
d-exchange;v=b3;q=0.9" --header="Accept-Language: en-IN,en-GB;q=0.9,en-US;q=0.8,en;q=0.7" --header
="Referer: https://drive.google.com/drive/u/0/folders/1qYtDPghLMT6rv3xd7NmQUSUKWwCS5375" --header=
"Cookie: AUTH_smvf2o367eja801lv3hmgenqovh1tbcl_nonce=6nf1kt0o0r09g;
_ga=GA1.2.1476194893.1587472682" --header="Connection: keep-alive" "https://doc-00-b0-
docs.googleusercontent.com/docs/securesc/9csvdbmvo9gt489glsl99tqs7subbnk6/lg1ijt1tj91quojbv5ou9gr9e
lrf/1587882075000/06629147635963609455/01088116874641946513/1fDJptlCFEWNV5UNGPc4geTykgFI3PDCV?e=do
wnload&authuser=0&nonce=6nf1kt0o0r09g&user=01088116874641946513&hash=cgmiss8vte9i9camaovfh36fosqcfr
-c -O 'storage_sample_stage4.h5'
```

```
--2020-04-26 06:22:48--  https://doc-00-b0-
docs.googleusercontent.com/docs/securesc/9csvdbmvo9gt489glsl99tqs7subbnk6/lg1ijt1tj91quojbv5ou9gr9e
lrf/1587882075000/06629147635963609455/01088116874641946513/1fDJptlCFEWNV5UNGPc4geTykgFI3PDCV?e=do
wnload&authuser=0&nonce=6nf1kt0o0r09g&user=01088116874641946513&hash=cgmiss8vte9i9camaovfh36fosqcfr

Resolving doc-00-b0-docs.googleusercontent.com (doc-00-b0-docs.googleusercontent.com)...
172.217.203.132, 2607:f8b0:400c:c07::84
```

```
Connecting to doc-00-b0-docs.googleusercontent.com (doc-00-b0-
docs.googleusercontent.com)|172.217.203.132|:443... connected.
HTTP request sent, awaiting response... 416 Requested range not satisfiable

    The file is already fully retrieved; nothing to do.
```

In [0]:

```
#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 [54]:

```
df_final_train.columns
```

Out[54]:

```
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 [55]:

```
print(type(df_final_train))
df_final_train.head(5)
```

```
<class 'pandas.core.frame.DataFrame'>
```

Out[55]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	num_followe
0	273084	1505602	1	0	0.000000	0.000000	0.000000	
1	832016	1543415	1	0	0.187135	0.028382	0.343828	
2	1325247	760242	1	0	0.369565	0.156957	0.566038	
3	1368400	1006992	1	0	0.000000	0.000000	0.000000	
4	140165	1708748	1	0	0.000000	0.000000	0.000000	

In [56]:

```
df_final_train.shape
```

Out[56]:

```
(100002, 54)
```

In [57]:

```
!wget --header="Host: doc-0g-b0-docs.googleusercontent.com" --header="User-Agent: Mozilla/5.0
(Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.122
Safari/537.36" --header="Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/s
d-exchange;v=b3;q=0.9" --header="Accept-Language: en-IN,en-GB;q=0.9,en-US;q=0.8,en;q=0.7" --header
="Referer: https://drive.google.com/drive/u/0/folders/1ONG0P5YAMlkzyKB7VugPKtLs9fXcdmRx" --header=
"Cookie:
AUTH_smvf2o367eja801lv3hmgenqovh1tbcl=01088116874641946513|1587882075000|t7t6ugl59of4q3drpglo141hc
re; _ga=GA1.2.1476194893.1587472682" --header="Connection: keep-alive" "https://doc-0g-b0-
docs.googleusercontent.com/docs/securesc/9csvdbmvo9gt489glsl99tqs7subbnk6/t4fvtrbq94anhq5ja4k3ui3qk
0ce/1587882150000/06629147635963609455/01088116874641946513/1XLHsIRXKLx9TA9nuC1SS7JDkLyRVmo69?e=do
wnload&authuser=0" -c -O 'train_pos_after_eda.csv'
```

```
--2020-04-26 06:23:56--  https://doc-0g-b0-
docs.googleusercontent.com/docs/securesc/9csvdbmvo9gt489glsl99tqs7subbnk6/t4fvtrbq94anhq5ja4k3ui3qk
0ce/1587882150000/06629147635963609455/01088116874641946513/1XLHsIRXKLx9TA9nuC1SS7JDkLyRVmo69?e=do
wnload&authuser=0
Resolving doc-0g-b0-docs.googleusercontent.com (doc-0g-b0-docs.googleusercontent.com)...
172.217.203.132, 2607:f8b0:400c:c07::84
Connecting to doc-0g-b0-docs.googleusercontent.com (doc-0g-b0-
docs.googleusercontent.com)|172.217.203.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/csv]
Saving to: ‘train_pos_after_eda.csv’

train_pos_after_eda     [      <=>           ] 113.80M  56.2MB/s    in 2.0s

2020-04-26 06:23:59 (56.2 MB/s) - ‘train_pos_after_eda.csv’ saved [119333798]
```

In [58]:

```
train_graph=nx.read_edgelist('train_pos_after_eda.csv',delimiter=',',create_using=nx.DiGraph(),nod
etype=int)
print(nx.info(train_graph))
```

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

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 [0]:

```
def  preferntial_attachment_followers(a,b):
    try:

        if len(set(train_graph.predecessors(a))) == 0  | len(set(train_graph.predecessors(b))) == 0
:
            return 0
        sim = (len(set(train_graph.predecessors(a))))  *  (len(set(train_graph.predecessors(b))))
        return sim
    except:
        return 0
```

In [0]:

```
#mapping preferntial attachment followers to train and test data
df_final_train['preferntial_attachment_followers'] = df_final_train.apply(lambda row:
                                            preferntial_attachment_followers(row['source_node'],row
'destination_node']),axis=1)
```

```
df_final_test['preferntial_attachment_followers'] = df_final_test.apply(lambda row:
                                            preferntial_attachment_followers(row['source_node'],rov
'destination_node']),axis=1)
```

In [0]:

```
def  preferntial_attachment_followees(a,b):
    try:
        
        if len(set(train_graph.successors(a))) == 0  | len(set(train_graph.successors(b))) == 0:
            return 0
        sim = (len(set(train_graph.successors(a))))   *   (len(set(train_graph.successors(b))))
        return sim
    except:
        return 0
```

In [0]:

```
#mapping preferntial attachment followess to train and test data
df_final_train['preferntial_attachment_followees'] = df_final_train.apply(lambda row:
                                            preferntial_attachment_followees(row['source_node'],rov
'destination_node']),axis=1)
df_final_test['preferntial_attachment_followees'] = df_final_test.apply(lambda row:
                                            preferntial_attachment_followees(row['source_node'],rov
'destination_node']),axis=1)
```

SVD_dot feature

SVD_dot feature is product of source node and destination for both u and v.T matrix

In [0]:

```
for i in range(len(df_final_train)):
    df_final_train["svd_dot_u"]=((df_final_train["svd_u_s_1"][i] *df_final_train["svd_u_d_1"][i]))+
((df_final_train["svd_u_s_2"][i] *df_final_train["svd_u_d_2"][i]))+\
                                ((df_final_train["svd_u_s_3"][i] *df_final_train["svd_u_d_3"][i]))+
(df_final_train["svd_u_s_4"][i] *df_final_train["svd_u_d_4"][i])) +\
                                ((df_final_train["svd_u_s_5"][i] *df_final_train["svd_u_d_5"][i]))+
(df_final_train["svd_u_s_6"][i] *df_final_train["svd_u_d_6"][i]))
    df_final_train["svd_dot_v"]=((df_final_train["svd_v_s_1"][i] *df_final_train["svd_v_d_1"][i]))+
((df_final_train["svd_v_s_2"][i] *df_final_train["svd_v_d_2"][i]) )+\
                                ((df_final_train["svd_v_s_3"][i] *df_final_train["svd_v_d_3"][i]))+
(df_final_train["svd_v_s_4"][i] *df_final_train["svd_v_d_4"][i]) )+\
                                ((df_final_train["svd_v_s_5"][i] *df_final_train["svd_v_d_5"][i]))+
(df_final_train["svd_v_s_6"][i] *df_final_train["svd_v_d_6"][i]))
```

In [0]:

```
for i in range(len(df_final_test)):
    df_final_test["svd_dot_u"]=((df_final_test["svd_u_s_1"][i] *df_final_test["svd_u_d_1"][i]))+((d
f_final_test["svd_u_s_2"][i] *df_final_test["svd_u_d_2"][i])) +\
                                (df_final_test["svd_u_s_3"][i] *df_final_test["svd_u_d_3"][i])+(df_
inal_test["svd_u_s_4"][i] *df_final_test["svd_u_d_4"][i]) +\
                                (df_final_test["svd_u_s_5"][i] *df_final_test["svd_u_d_5"][i])+(df_
inal_test["svd_u_s_6"][i] *df_final_test["svd_u_d_6"][i])
    df_final_test["svd_dot_v"]=(df_final_test["svd_v_s_1"][i] *df_final_test["svd_v_d_1"][i])+(df_f
inal_test["svd_v_s_2"][i] *df_final_test["svd_v_d_2"][i]) +\
                                (df_final_test["svd_v_s_3"][i] *df_final_test["svd_v_d_3"][i])+(df_
inal_test["svd_v_s_4"][i] *df_final_test["svd_v_d_4"][i]) +\
                                (df_final_test["svd_v_s_5"][i] *df_final_test["svd_v_d_5"][i])+(df_
inal_test["svd_v_s_6"][i] *df_final_test["svd_v_d_6"][i])
```

In [65]:

```
df_final_train.head(5)
```

Out[65]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	num_followe
0	273084	1505602	1	0	0.000000	0.000000	0.000000	
1	832016	1543415	1	0	0.187135	0.028382	0.343828	
2	1325247	760242	1	0	0.369565	0.156957	0.566038	
3	1368400	1006992	1	0	0.000000	0.000000	0.000000	
4	140165	1708748	1	0	0.000000	0.000000	0.000000	

In [0]:

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

In [0]:

```
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)
```

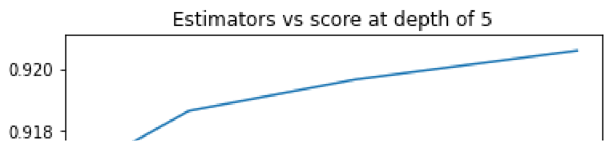
In [68]:

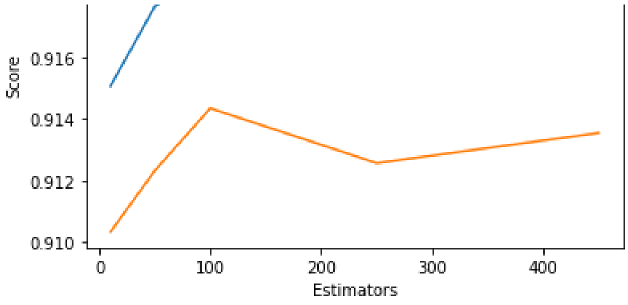
```
estimators = [10,50,100,250,450]
train_scores = []
test_scores = []
for i in estimators:
    clf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=5, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            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 0.9150545972795674 test Score 0.9103378634240753
Estimators =  50 Train Score 0.9176359931023672 test Score 0.9123226727104843
Estimators =  100 Train Score 0.9186409434634156 test Score 0.9143397415694925
Estimators =  250 Train Score 0.9196526766729861 test Score 0.9125661180536531
Estimators =  450 Train Score 0.9205880814105583 test Score 0.9135376325980805
```

Out[68]:

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

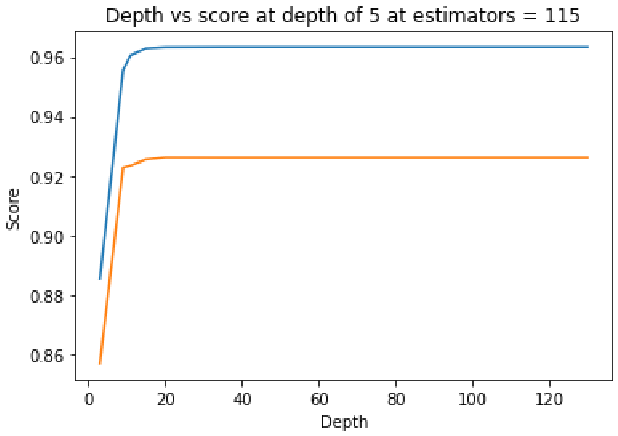




In [69]:

```
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,war
m_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 0.8853267055838093 test Score 0.8569288068838563
depth =  9 Train Score 0.9556087187666136 test Score 0.9228074618046763
depth =  11 Train Score 0.9606332905434317 test Score 0.9235351273710447
depth =  15 Train Score 0.9629192325309176 test Score 0.9256264645654331
depth =  20 Train Score 0.9633234445482993 test Score 0.926322444678609
depth =  35 Train Score 0.9633746152048686 test Score 0.926296710013278
depth =  50 Train Score 0.9633746152048686 test Score 0.926296710013278
depth =  70 Train Score 0.9633746152048686 test Score 0.926296710013278
depth =  130 Train Score 0.9633746152048686 test Score 0.926296710013278
```



In [70]:

```
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)
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,return_train_score=Tr
ue)

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 [0.9618273  0.96235223 0.96054857 0.96106605 0.96299828]
mean train scores [0.96296072 0.96296808 0.96082918 0.96228931 0.96418863]
```

In [71]:

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

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=14, max_features='auto',
                       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, 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 [74]:

```
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 0.9638921025391219
Test f1 score 0.9257162102427011
```

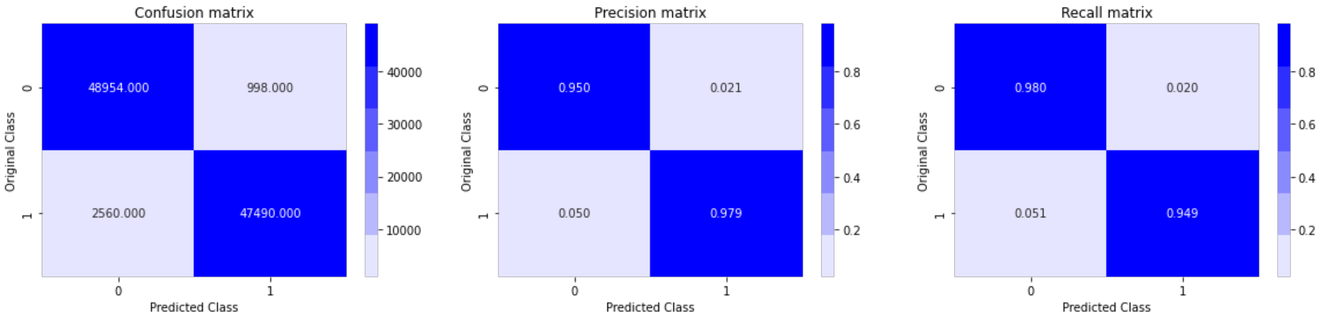
In [0]:

```
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)
    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()
```

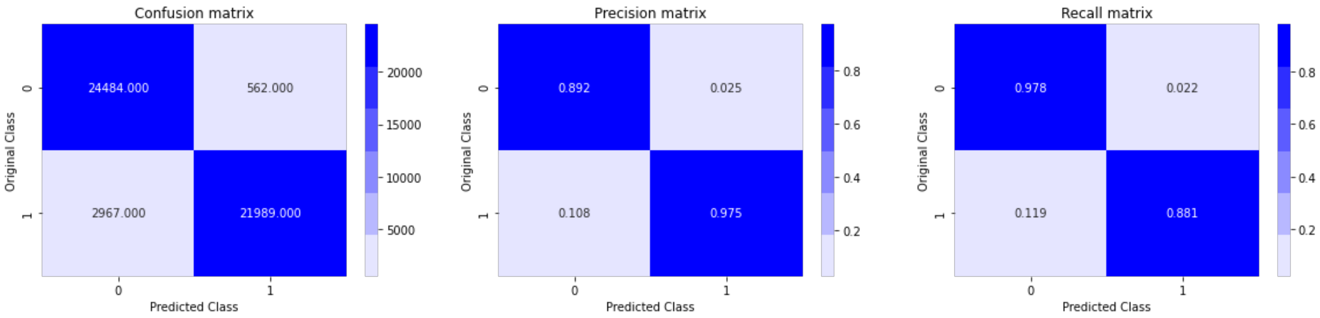
In [76]:

```
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
```

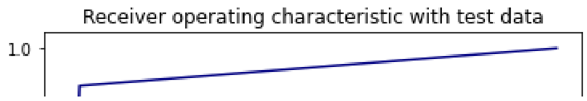


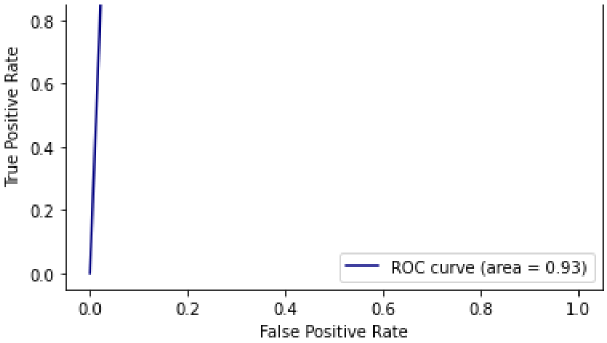
```
Test confusion_matrix
```



In [77]:

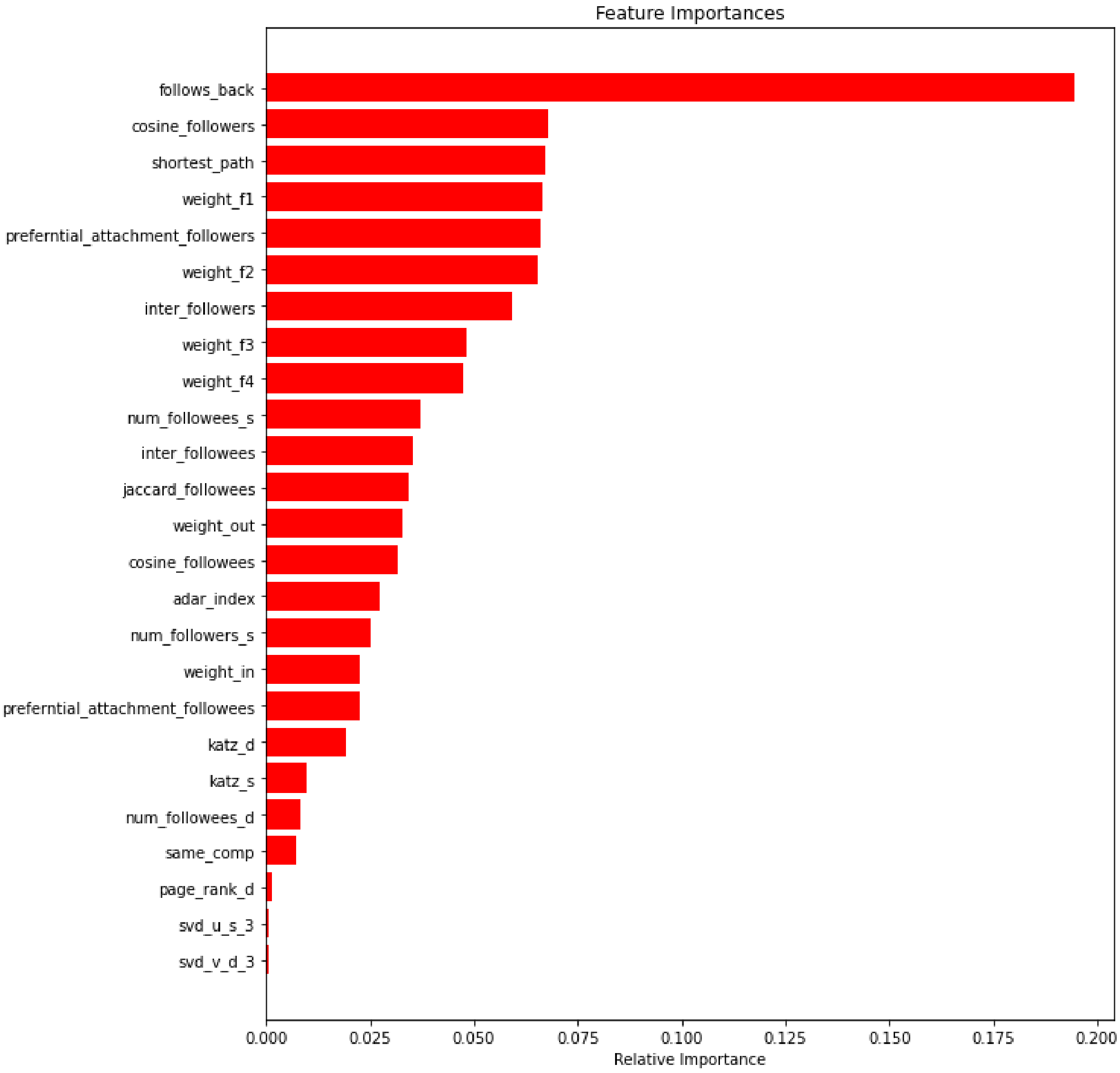
```
from sklearn.metrics import roc_curve, auc
fpr,tpr,ths = roc_curve(y_test,y_test_pred)
auc_sc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='navy',label='ROC curve (area = %0.2f)' % auc_sc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic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





In [78]:

```
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()
```



XGBOOOST MODEL

in xgboost there are mainly three hyperparameter

1.learning rate (nu): it shows how fast we want to move towards predicted value

2.n_estimator: no of decision tree . in boosting you want high bias and low variance so as no of decision tree increases your chances of overfitting increase.

3.max_depth: in boosing you need high bias and low variance and we achived this by having shallow or decision tree with less depth

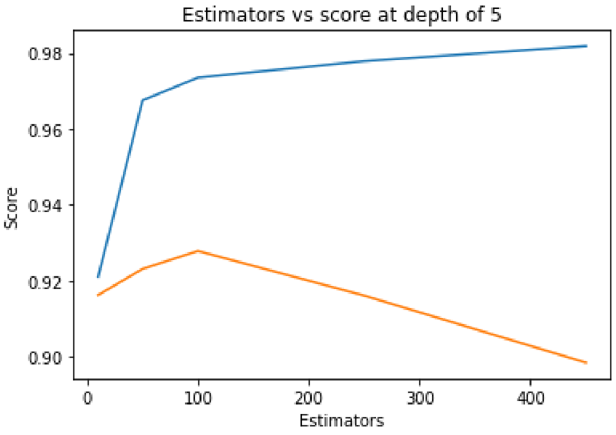
In [104]:

```
estimators = [10,50,100,250,450]
train_scores = []
test_scores = []
for i in estimators:
    clf=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=3,
              min_child_weight=1, missing=None, n_estimators=i, 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)

    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 0.9210492696844526 test Score 0.9162413689582708
Estimators =  50 Train Score 0.9675354922332259 test Score 0.9231354642313546
Estimators =  100 Train Score 0.9735434729762581 test Score 0.927852099985159
Estimators =  250 Train Score 0.9778734238603298 test Score 0.9161052225035404
Estimators =  450 Train Score 0.9818456578682249 test Score 0.8984685195689166
```

Out[104]:

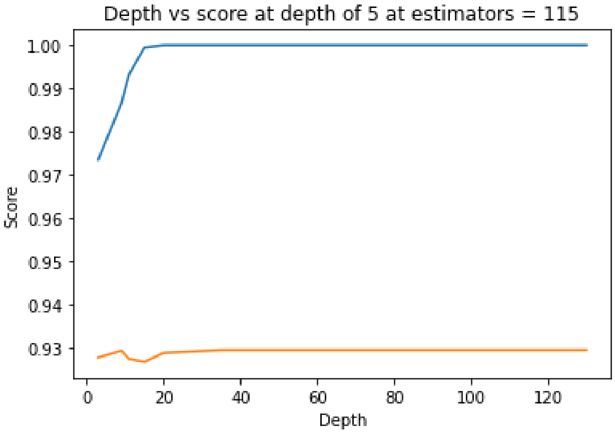
```
Text(0.5, 1.0, 'Estimators vs score at depth of 5')
```



In [105]:

```
depths = [3,9,11,15,20,35,50,70,130]
train_scores = []
test_scores = []
for i in depths:
    clf=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=i,
              min_child_weight=1, missing=None, n_estimators=100, 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)
    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 0.9735434729762581 test Score 0.927852099985159
depth =  9 Train Score 0.9865335453612191 test Score 0.9294234803059517
depth =  11 Train Score 0.9932370058197872 test Score 0.9275122510023548
depth =  15 Train Score 0.999450247388675 test Score 0.926811671087533
depth =  20 Train Score 1.0 test Score 0.9289161727349704
depth =  35 Train Score 1.0 test Score 0.929567923449838
depth =  50 Train Score 1.0 test Score 0.929567923449838
depth =  70 Train Score 1.0 test Score 0.929567923449838
depth =  130 Train Score 1.0 test Score 0.929567923449838
```



in boosting (adaboost,gradientboost,xgboost) we need our model to be have high bias and high bias mean high training error .

high bias == high training error

so we will select our hyperparameter n_estimator and max depth which have high training error

In [0]:

```
best_n_estimator=10
best_max_depth=3
```

In [0]:

```
clf1 =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=best_max_depth,
              min_child_weight=1, missing=None, n_estimators=best_n_estimator, 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 [0]:

```
clf1.fit(df_final_train,y_train)
y_train_pred1 = clf1.predict(df_final_train)
y_test_pred1= clf1.predict(df_final_test)
```

In [109]:

```
from sklearn.metrics import f1_score
```

```
print('Train f1 score',f1_score(y_train,y_train_pred1))
print('Test f1 score',f1_score(y_test,y_test_pred1))
```

```
Train f1 score 0.9210492696844526
Test f1 score 0.9162413689582708
```

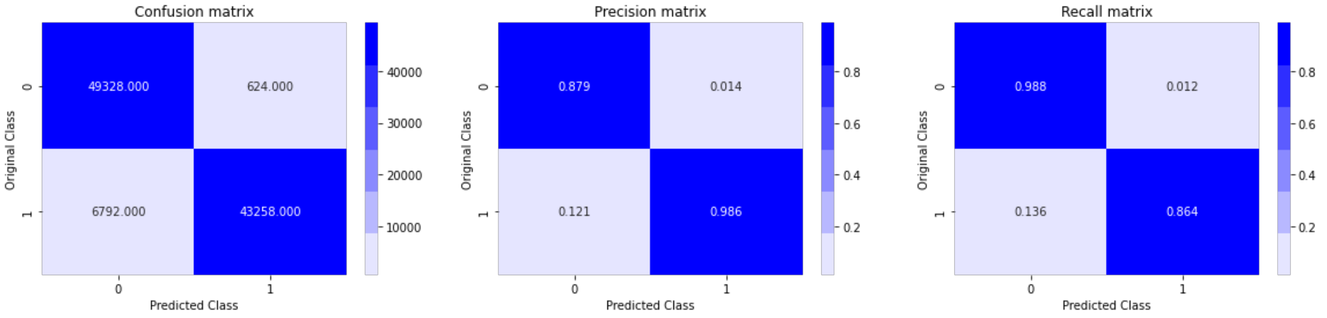
In [0]:

```
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()
```

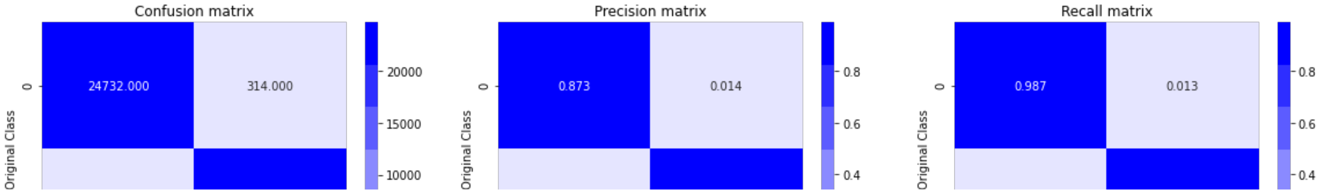
In [111]:

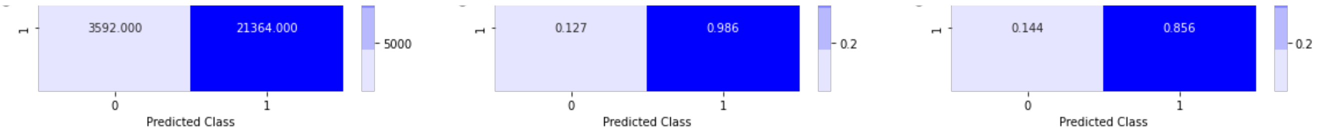
```
print('Train confusion_matrix')
plot_confusion_matrix(y_train,y_train_pred1)
print('Test confusion_matrix')
plot_confusion_matrix(y_test,y_test_pred1)
```

```
Train confusion_matrix
```



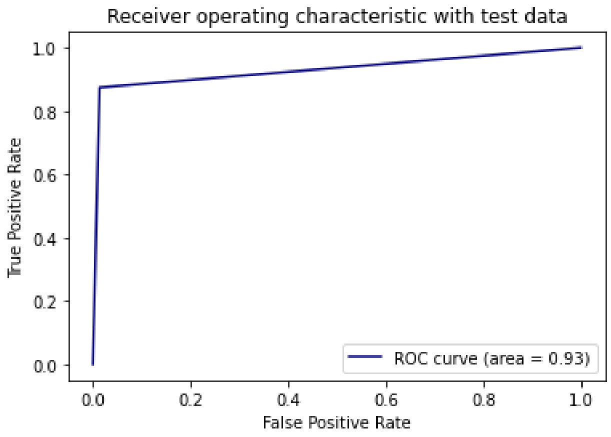
```
Test confusion_matrix
```





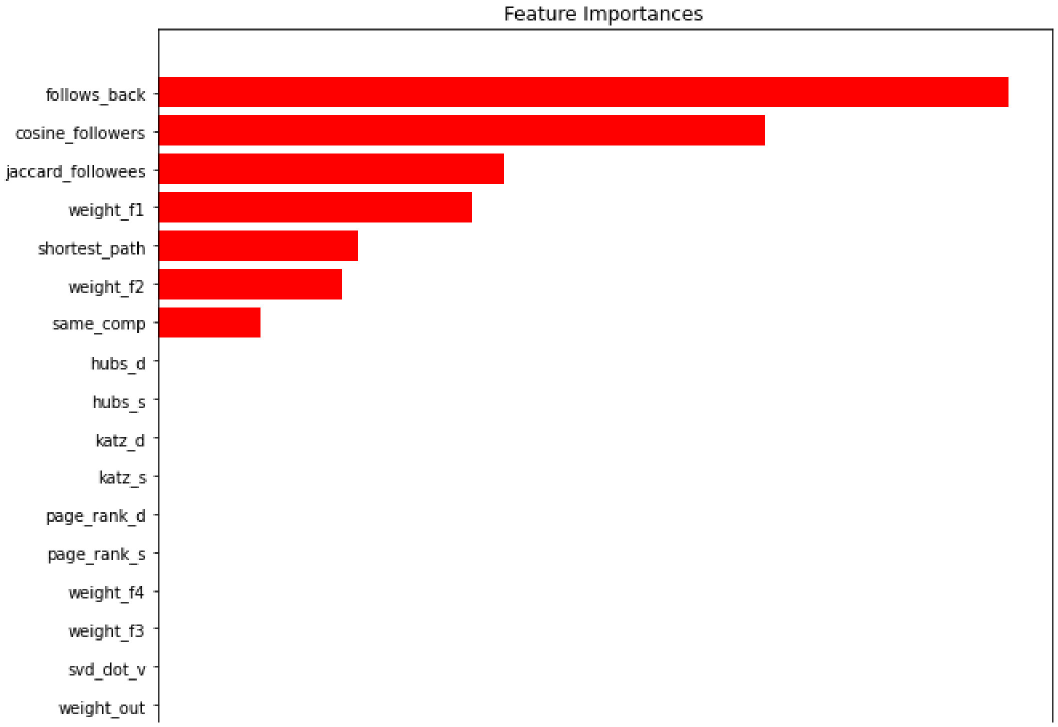
In [112]:

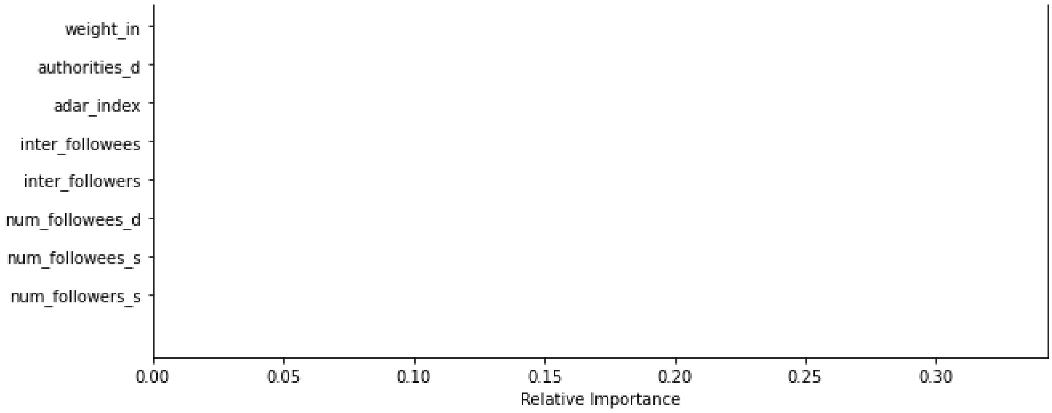
```
from sklearn.metrics import roc_curve, auc
fpr,tpr,ths = roc_curve(y_test,y_test_pred)
auc_sc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='navy',label='ROC curve (area = %0.2f)' % auc_sc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic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



In [113]:

```
features = df_final_train.columns
importances = clf1.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()
```





In [115]:

```
from prettytable import PrettyTable
ptable = PrettyTable()
ptable.field_names = ["Model", "n_estimators", "max_depth", "Train f1-Score","Test f1-Score"]
ptable.add_row(['Random Forest','121','14','0.963892','0.925716'])
ptable.add_row(['XGBOOST','10','3','0.921049','0.916241'])
print(ptable)
```

```
+---------------+--------------+-----------+----------------+---------------+
|     Model     | n_estimators | max_depth | Train f1-Score | Test f1-Score |
+---------------+--------------+-----------+----------------+---------------+
| Random Forest |     121      |     14    |    0.963892    |    0.925716   |
|    XGBOOST    |      10      |     3     |    0.921049    |    0.916241   |
+---------------+--------------+-----------+----------------+---------------+
```

In [0]:

In [0]: