Social network Graph Link Prediction - Facebook Challenge

In [37]:

```
#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
from scipy.stats import randint as sp_randint
from scipy.stats import uniform
```

In [38]:

```
!gdown --id 1fDJptlCFEWNV5UNGPc4geTykgFI3PDCV
```

```
Downloading...
From: https://drive.google.com/uc?id=1fDJptlCFEWNV5UNGPc4geTykgFI3PDCV
(https://drive.google.com/uc?id=1fDJptlCFEWNV5UNGPc4geTykgFI3PDCV)
To: /content/storage_sample_stage4.h5
103MB [00:00, 155MB/s]
```

In [39]:

```
#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 [40]:

```
df_final_train.columns
```

Out[40]:

```
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', 'wei
ght_out',
       'weight_f1', 'weight_f2', 'weight_f3', 'weight_f4', 'page_rank_
s',
       'page_rank_d', 'katz_s', 'katz_d', 'hubs_s', 'hubs_d', 'authori
ties_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 [41]:

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

In [42]:

```
df_final_train.drop(['source_node', 'destination_node','indicator_link'],axis=1,inpl
df_final_test.drop(['source_node', 'destination_node','indicator_link'],axis=1,inpla
```

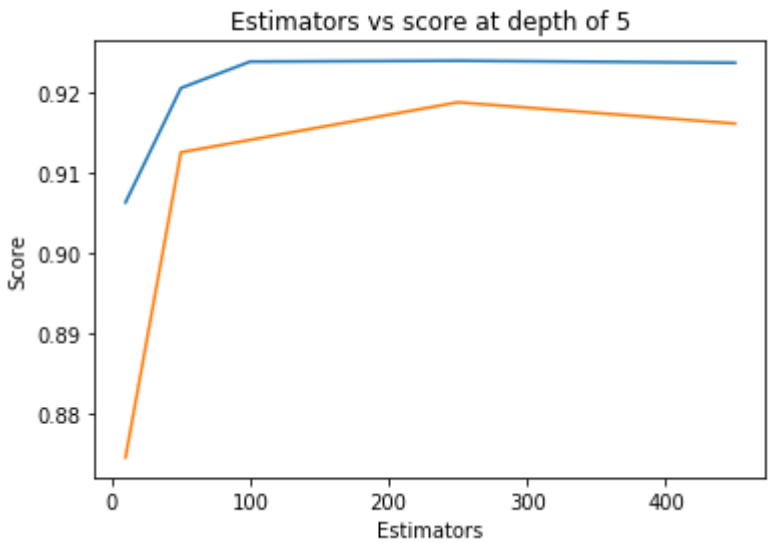
In [ ]:

```
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,
    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 0.874560527
8006858
Estimators =  50 Train Score 0.9205725512208812 test Score 0.912565335
5634538
Estimators =  100 Train Score 0.9238690848446947 test Score 0.91411997
14153599
Estimators =  250 Train Score 0.9239789348046863 test Score 0.91880072
32664732
Estimators =  450 Train Score 0.9237190618658074 test Score 0.91615076
85828595
```

Out[6]:

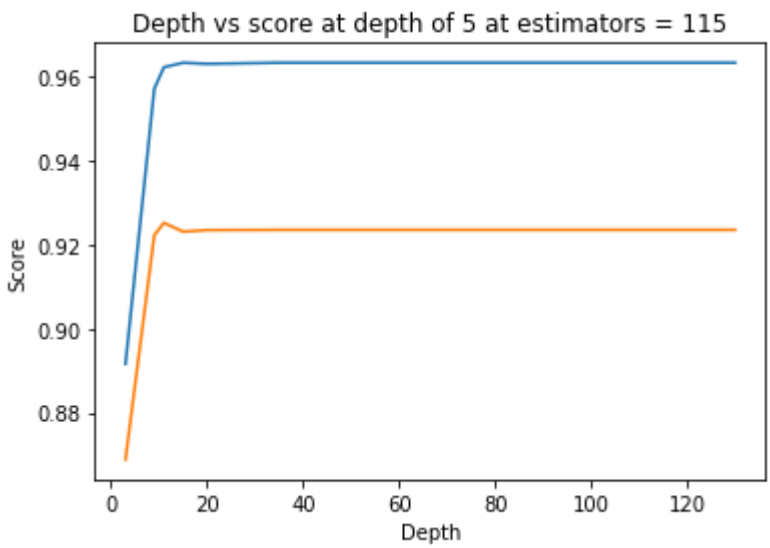
```
Text(0.5,1,'Estimators vs score at depth of 5')
```



In [ ]:

```
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=2
    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 0.868793485987549
1
depth =  9 Train Score 0.9572226298198419 test Score 0.922295303145290
4
depth =  11 Train Score 0.9623451340902863 test Score 0.92523187582812
79
depth =  15 Train Score 0.9634267621927706 test Score 0.92312883564966
15
depth =  20 Train Score 0.9631629153051491 test Score 0.92350510247111
41
depth =  35 Train Score 0.9634333127085721 test Score 0.92356016527531
84
depth =  50 Train Score 0.9634333127085721 test Score 0.92356016527531
84
depth =  70 Train Score 0.9634333127085721 test Score 0.92356016527531
84
depth =  130 Train Score 0.9634333127085721 test Score 0.9235601652753
184
```



In [ ]:

```
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 0.963300
05]
mean train scores [0.96294922 0.96266735 0.96115674 0.96263457 0.96430
539]
```

In [ ]:

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

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='g
ini',
            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=Fa
lse)
```

In [ ]:

```
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 [ ]:

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

In [ ]:

```
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
```

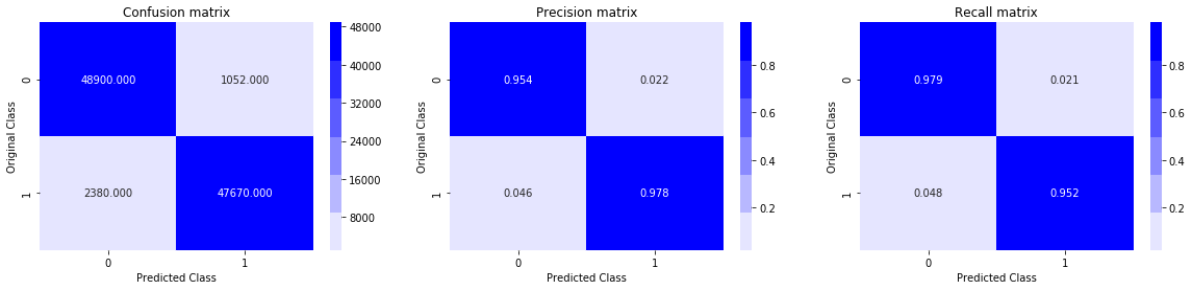
In [55]:

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

    plt.show()
```

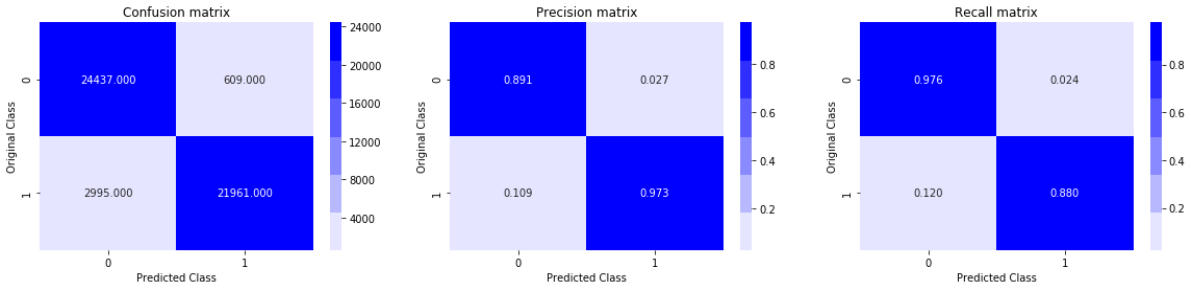
In [ ]:

```
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
```

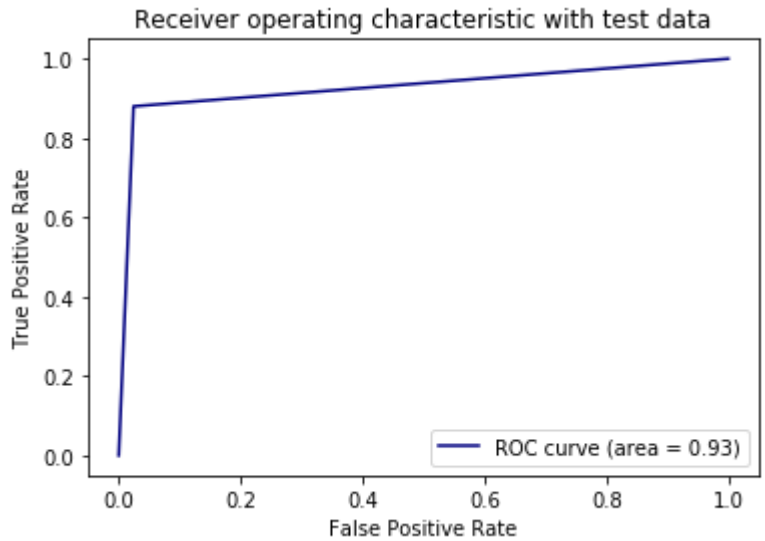



```
Test confusion_matrix
```



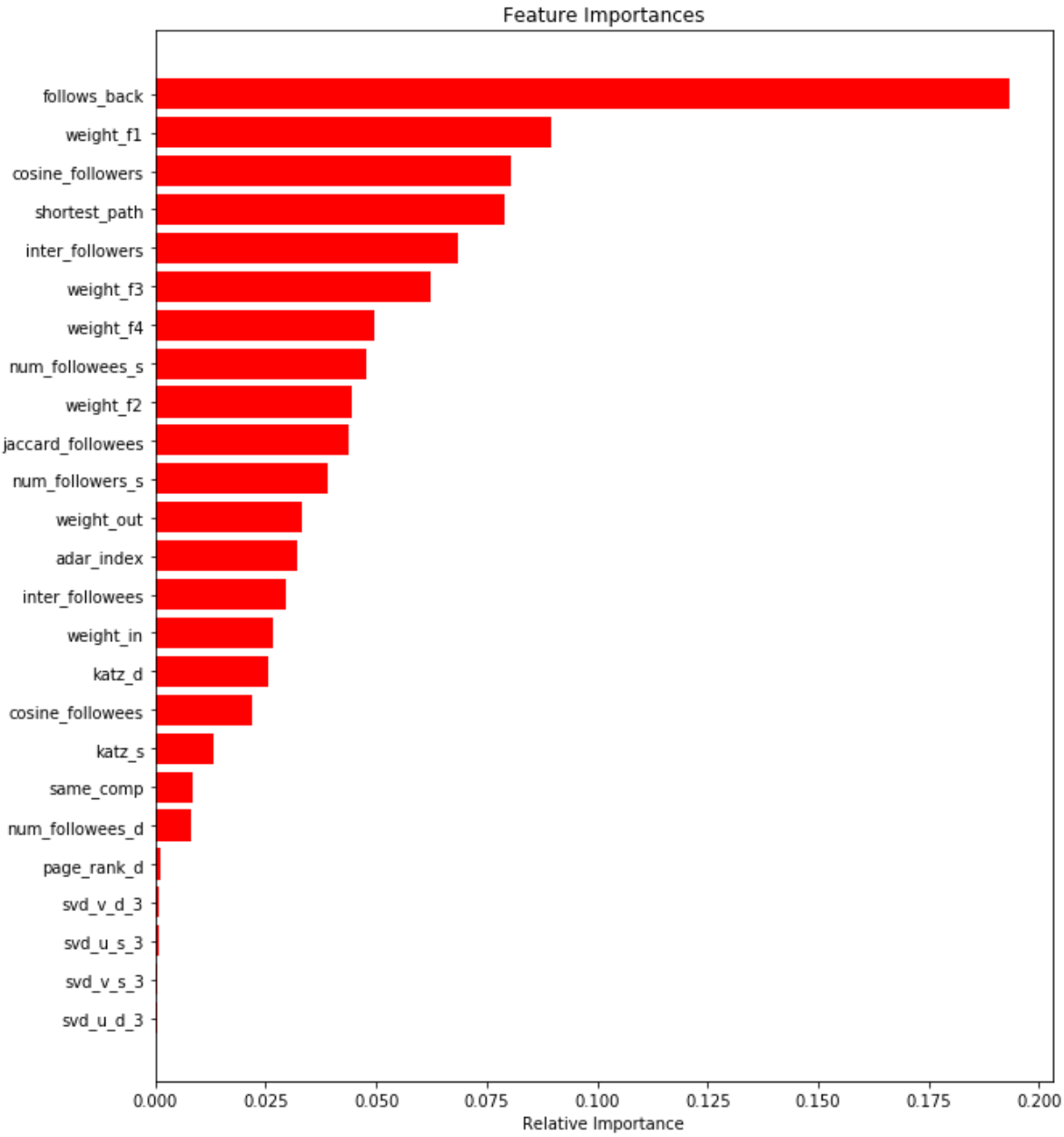
In [ ]:

```
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 [ ]:

```
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.

In [19]:

```
df_final_train.columns
```

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', 'wei
ght_out',
       'weight_f1', 'weight_f2', 'weight_f3', 'weight_f4', 'page_rank_
s',
       'page_rank_d', 'katz_s', 'katz_d', 'hubs_s', 'hubs_d', 'authori
ties_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',
       'FB_FOLLWER_FEATURE_1', 'FB_FOLLWER_FEATURE_2'],
      dtype='object')
```

In [43]:

```
#for train data
df_final_train['FB_FOLLWER_FEATURE_1']=df_final_train['num_followers_s']*df_final_tr
df_final_train['FB_FOLLWEES_FEATURE_2']=df_final_train['num_followees_s']*df_final_t
```

In [44]:

```
#for test data

df_final_test['FB_FOLLWER_FEATURE_1']=df_final_test['num_followers_s']*df_final_test
df_final_test['FB_FOLLWEES_FEATURE_2']=df_final_test['num_followees_s']*df_final_tes
```

In [45]:

```
df_final_train['svd_u_s_1'].shape, df_final_train['svd_u_s_2'].shape,df_final_train[
```

Out[45]:

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

In [28]:

```
#for train data
total_fields = df_final_train['svd_u_s_1'].shape[0]
svd=[]
for ele in range(total_fields):
    svd_v_s_details = []
    for fea in ['svd_u_s_1','svd_u_s_2','svd_u_s_3','svd_u_s_4','svd_u_s_5','svd_u_s
        svd_v_s_details.append(np.array(df_final_train[fea][ele]))
    svd_v_d_details = []
    for fea in ['svd_u_d_1','svd_u_d_2','svd_u_d_3','svd_u_d_4','svd_u_d_5','svd_u_d
        svd_v_d_details.append(np.array(df_final_train[fea][ele]))
    svd.append(np.dot(svd_v_s_details,svd_v_d_details))
df_final_train['SVD_U_S__AND_D_FEATURES']=svd
```

In [29]:

```
#for test data
total_fields = df_final_test['svd_u_s_1'].shape[0]
svd=[]
for ele in range(total_fields):
    svd_v_s_details = []
    for fea in ['svd_u_s_1','svd_u_s_2','svd_u_s_3','svd_u_s_4','svd_u_s_5','svd_u_s
        svd_v_s_details.append(np.array(df_final_train[fea][ele]))
    svd_v_d_details = []
    for fea in ['svd_u_d_1','svd_u_d_2','svd_u_d_3','svd_u_d_4','svd_u_d_5','svd_u_d
        svd_v_d_details.append(np.array(df_final_train[fea][ele]))
    svd.append(np.dot(svd_v_s_details,svd_v_d_details))
df_final_test['SVD_U_S__AND_D_FEATURES']=svd
df_final_test.head()
```

Out[29]:

	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	num_followers_s	nur
0	0	0.0	0.029161	0.000000	14	
1	0	0.0	0.000000	0.000000	17	
2	0	0.0	0.000000	0.000000	10	
3	0	0.0	0.000000	0.000000	37	
4	0	0.2	0.042767	0.347833	27	

In [48]:

```
'''
    Get the best model from RandomizedSearchCV using XGBOOST
'''
fb_model_classifier = xgb.XGBClassifier()
param_dist = {"n_estimators":sp_randint(105,125),"max_depth": sp_randint(10,15)}
fb_model_best_model = RandomizedSearchCV(fb_model_classifier, param_distributions=pa
                                   random_state=25,verbose=10, return_train_score=True,
#fit on best train data
fb_model_best_model.fit(df_final_train,y_train)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent wor
kers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:  2.5min
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:  5.0min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed: 10.8min
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed: 14.8min
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed: 23.2min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 31.2min finished
```

Out[48]:

```
RandomizedSearchCV(cv=3, error_score=nan,
                   estimator=XGBClassifier(base_score=0.5, booster='gb
tree',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=
0,
                                           learning_rate=0.1, max_delt
a_step=0,
                                           max_depth=3, min_child_weig
ht=1,
                                           missing=None, n_estimators=
100,
                                           n_jobs=1, nthread=None,
                                           objective='binary:logisti
c',
                                           random_state=0, reg_alpha=
0,
                                           reg_lambda=1, sc...
                                           seed=None, silent=None, sub
sample=1,
                                           verbosity=1),
                   iid='deprecated', n_iter=10, n_jobs=-1,
                   param_distributions={'max_depth': <scipy.stats._dis
tn_infrastructure.rv_frozen object at 0x7f79dc8de438>,
                                        'n_estimators': <scipy.stats._
distn_infrastructure.rv_frozen object at 0x7f79dc8def60>},
                   pre_dispatch='2*n_jobs', random_state=25, refit=Tru
e,
                   return_train_score=True, scoring='f1', verbose=10)
```

In [49]:

```
#train score
print('mean test scores', fb_model_best_model.cv_results_['mean_test_score'])
#test score
print('mean train scores',fb_model_best_model.cv_results_['mean_train_score'])
#finally print best score
print(fb_model_best_model.best_estimator_)
```

```
mean test scores [0.980249   0.98023047 0.97965035 0.97987899 0.979960
72 0.97981432
 0.98018737 0.97977405 0.98017984 0.98023065]
mean train scores [1.         1.         0.99411493 0.99669868 0.99701
496 0.99939524
 0.99998501 0.99930022 0.99569431 0.99794312]
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=14,
              min_child_weight=1, missing=None, n_estimators=120, n_jo
bs=1,
              nthread=None, objective='binary:logistic', random_state=
0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=Non
e,
              silent=None, subsample=1, verbosity=1)
```

In [59]:

```
#get best model
get_best_model_classifier=fb_model_best_model.best_estimator_
get_best_model_classifier.best_estimator_
```

Out[59]:

```
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=14,
              min_child_weight=1, missing=None, n_estimators=120, n_jo
bs=1,
              nthread=None, objective='binary:logistic', random_state=
0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=Non
e,
              silent=None, subsample=1, verbosity=1)
```

In [53]:

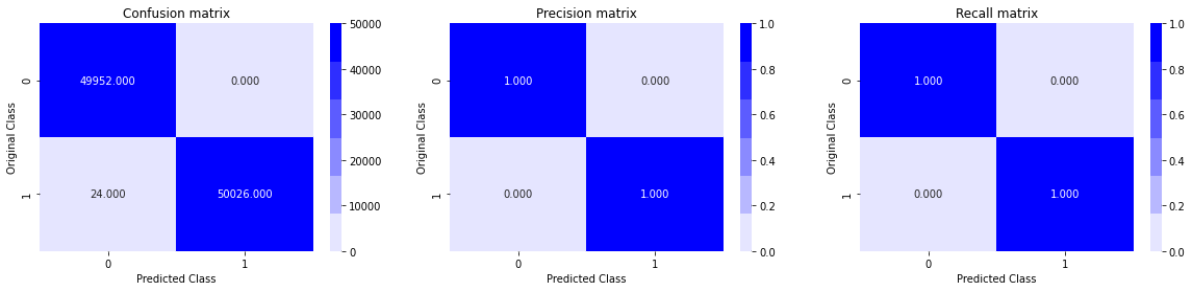
```
#doing model fit
get_best_model_classifierf.fit(df_final_train,y_train)
#predict on train data
y_train_pred = get_best_model_classifier.predict(df_final_train)
#predict on test data
y_test_pred  = get_best_model_classifierf.predict(df_final_test)
#f1 score
from sklearn.metrics import f1_score
print('FB MODEL TRAIN F1 SCORE',f1_score(y_train,y_train_pred))
print('FB MODEL TEST F1 SCORE',f1_score(y_test,y_test_pred))
```

```
FB MODEL TRAIN F1 SCORE 0.9997601822614812
FB MODEL TEST F1 SCORE 0.9268334111851884
```

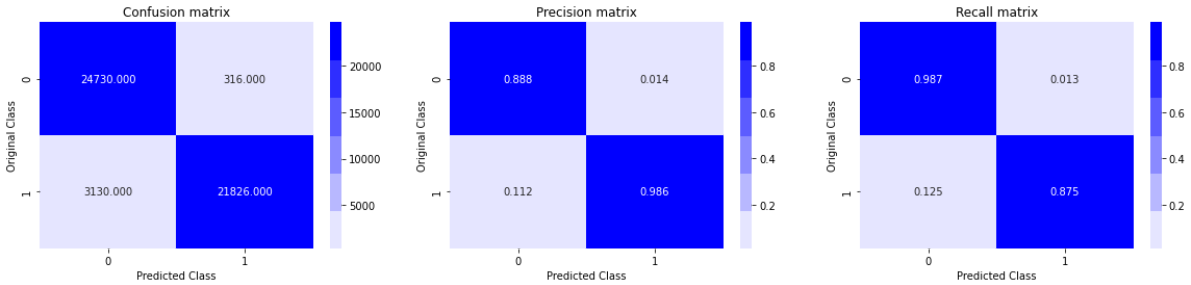
In [56]:

```
print('FB MODEL TRAIN CONFUSION MATRIX')
plot_confusion_matrix(y_train,y_train_pred)
print('FB MODEL TEST CONFUSION MATRIX')
plot_confusion_matrix(y_test,y_test_pred)
```

```
FB MODEL TRAIN CONFUSION MATRIX
```

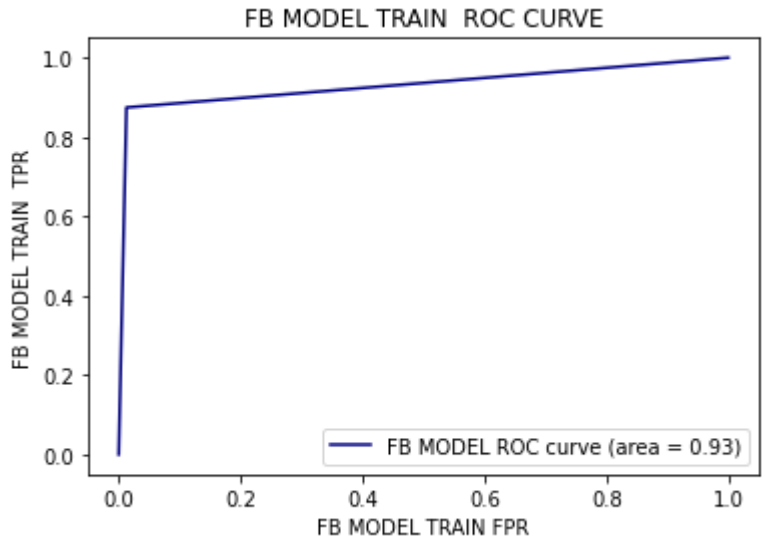



```
FB MODEL TEST CONFUSION MATRIX
```



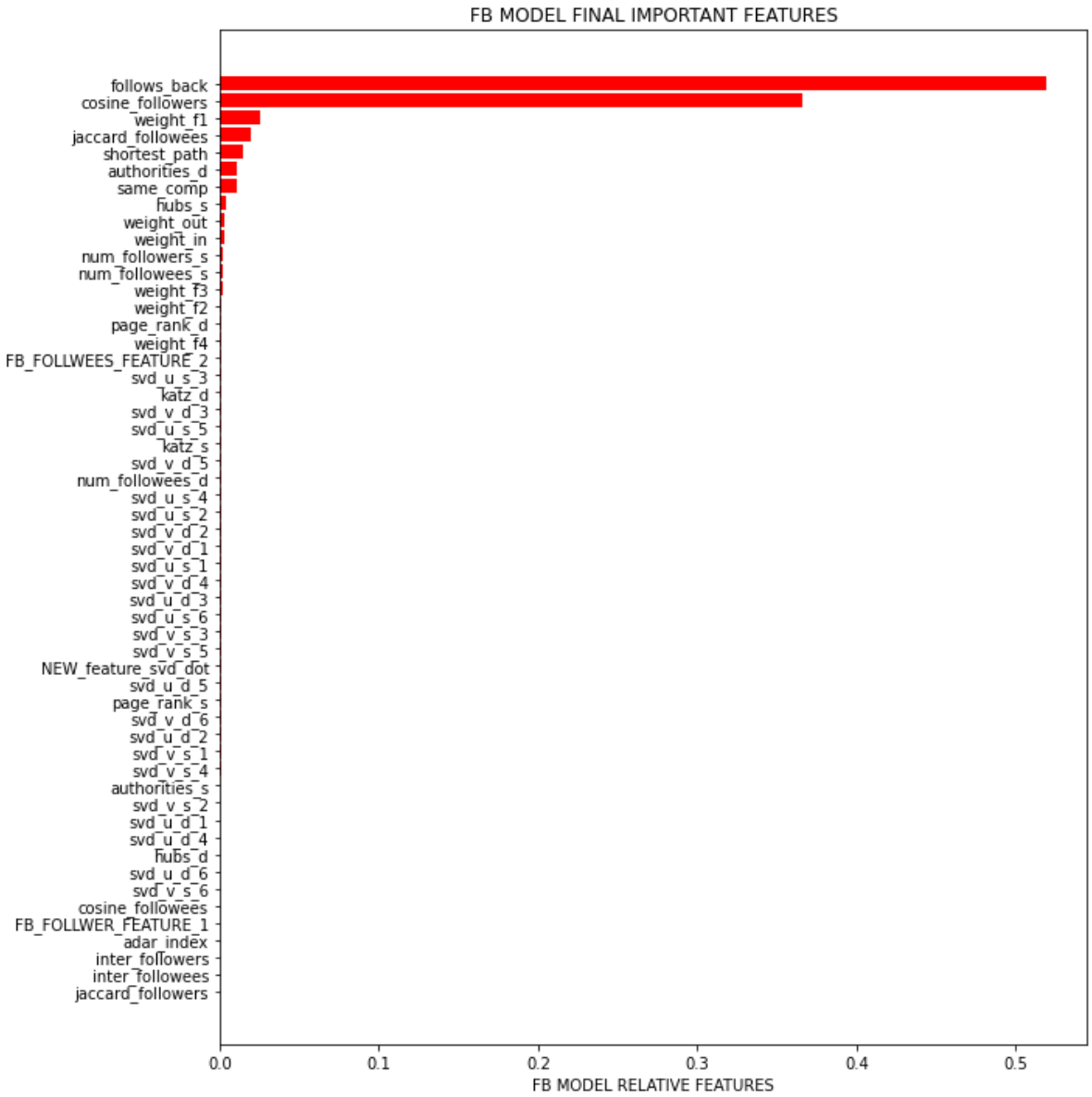
In [57]:

```
from sklearn.metrics import roc_curve, auc
best_model_fpr,best_model_tpr,best_model_ths = roc_curve(y_test,y_test_pred)
best_model_auc_sc = auc(best_model_fpr, best_model_tpr)
plt.plot(best_model_fpr, best_model_tpr, color='navy',label='FB MODEL ROC curve (are
plt.xlabel('FB MODEL TRAIN FPR')
plt.ylabel('FB MODEL TRAIN  TPR')
plt.title('FB MODEL TRAIN  ROC CURVE')
plt.legend()
plt.show()
```



In [58]:

```
best_model_features = df_final_train.columns
best_model_importances = get_best_model_clf.feature_importances_
best_model_indices = (np.argsort(best_model_importances))
plt.figure(figsize=(10,10))
plt.title('FB MODEL FINAL IMPORTANT FEATURES')
plt.barh(range(len(best_model_indices)), best_model_importances[best_model_indices],
plt.yticks(range(len(best_model_indices)), [best_model_features[i] for i in best_mod
plt.xlabel('FB MODEL RELATIVE FEATURES')
plt.show()
```



In [60]:

```
from prettytable import PrettyTable
FB_MODEL = PrettyTable()
FB_MODEL.field_names = ["FB MODEL ", "ESTIMATORS", "MAX_DEPTH", "TRAIN F1 SCORE","TE
FB_MODEL.add_row(['RANDOM FOREST','121','14','0.9652533106548414','0.924167823927955
FB_MODEL.add_row(['XGBClassifier','120','14','0.9997601822614812','0.926833411185188
print(FB_MODEL)
```

```
+---------------+------------+-----------+--------------------+-------
-------------+
|   FB MODEL    | ESTIMATORS | MAX_DEPTH |   TRAIN F1 SCORE   |   TEST
F1 SCORE    |
+---------------+------------+-----------+--------------------+-------
-------------+
| RANDOM FOREST |    121     |     14    | 0.9652533106548414 | 0.9241
678239279553 |
| XGBClassifier |    120     |     14    | 0.9997601822614812 | 0.9268
334111851884 |
+---------------+------------+-----------+--------------------+-------
-------------+
```

Observations:

After adding the follwers and follees of additional feature the model performing better.

References:

- http://cs229.stanford.edu/proj2007/DaniyalzadeLipus-FacebookFriendSuggestion.pdf (http://cs229.stanford.edu/proj2007/DaniyalzadeLipus-FacebookFriendSuggestion.pdf)
- https://www.appliedaicourse.com/course/4/facebook-friend-recommendation-using-graph-mining (https://www.appliedaicourse.com/course/4/facebook-friend-recommendation-using-graph-mining)
- https://ai.facebook.com/blog/dlrm-an-advanced-open-source-deep-learning-recommendation-model/ (https://ai.facebook.com/blog/dlrm-an-advanced-open-source-deep-learning-recommendation-model/)