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 [3]:

```
#reading
from pandas import read_hdf
df_final_train = read_hdf('/content/storage_sample_stage4.h5', 'tra
df_final_test = read_hdf('/content/storage_sample_stage4.h5', 'test
```

In [4]:
```
df_final_train.columns
```

Out[4]:
```
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',
       'num_followers_d', 'adar_index', 'follows_back', 'same_comp
',
       'shortest_path', 'weight_in', 'weight_out', 'weight_f1', 'w
eight_f2',
       'weight_f3', 'weight_f4', 'page_rank_s', 'page_rank_d', 'ka
tz_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',
       'pref_attach_for_followers', 'pref_attach_for_followees', '
svd_dot'],
      dtype='object')
```

In [5]:
```
y_train = df_final_train.indicator_link
y_test = df_final_test.indicator_link
```

In [6]:
```
df_final_train.drop(['source_node', 'destination_node','indicator_l
df_final_test.drop(['source_node', 'destination_node','indicator_li
```

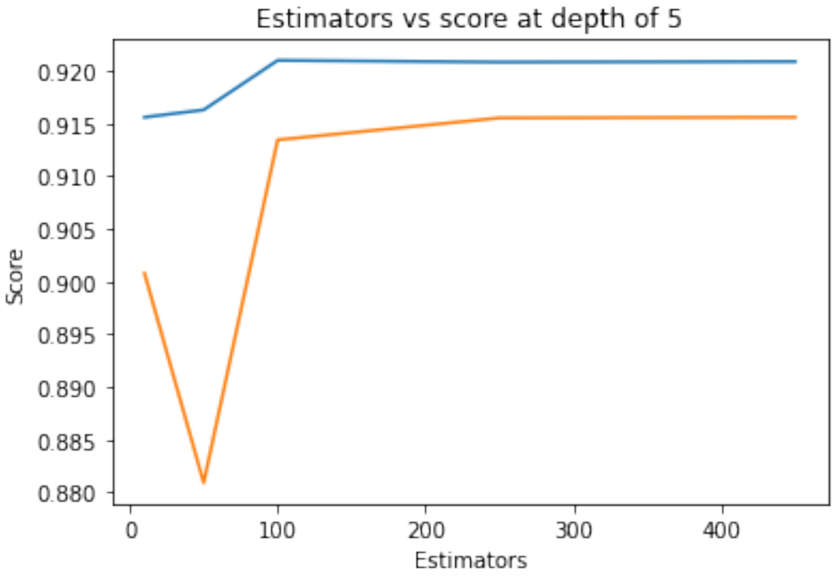
In [7]:

```
estimators = [10,50,100,250,450]
train_scores = []
test_scores = []
for i in estimators:
    clf = RandomForestClassifier(bootstrap=True, class_weight=None,
            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
    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',tes
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 0.9155990957045969 test Score 0.90078
24593396806
Estimators =  50 Train Score 0.9162977107245525 test Score 0.88090
80275527444
Estimators =  100 Train Score 0.9209946424655102 test Score 0.9134
520380920347
Estimators =  250 Train Score 0.920833071952826 test Score 0.91553
31654072209
Estimators =  450 Train Score 0.9208840587282512 test Score 0.9155
999496369664
```

Out[7]:

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

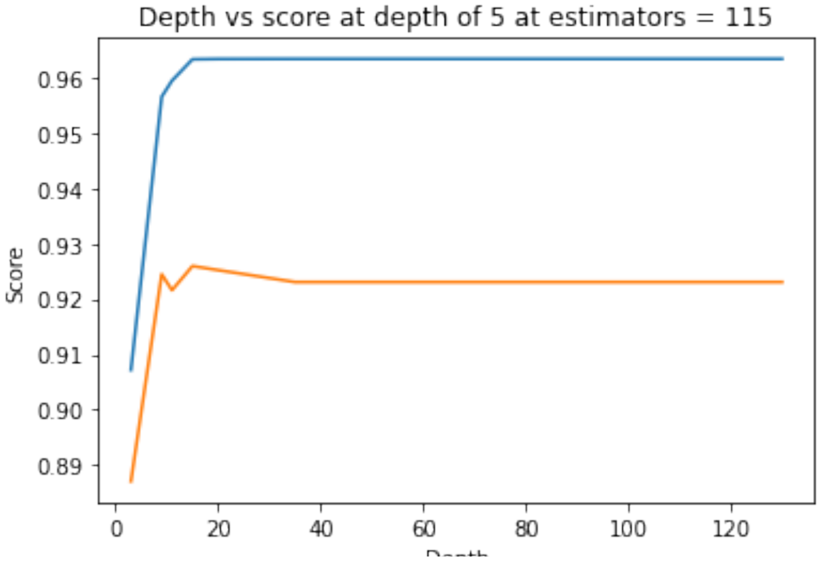


In [8]:

```
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,
            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=
    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 0.9071967945754353 test Score 0.88709913603
74871
depth =  9 Train Score 0.9565048821050697 test Score 0.92446353738
91802
depth =  11 Train Score 0.9594020136275806 test Score 0.9215835232
548325
depth =  15 Train Score 0.9632633200875842 test Score 0.9259415106
248684
depth =  20 Train Score 0.963321127273463 test Score 0.92520891949
77533
depth =  35 Train Score 0.9633355259828973 test Score 0.9230574857
864813
depth =  50 Train Score 0.9633355259828973 test Score 0.9230574857
864813
depth =  70 Train Score 0.9633355259828973 test Score 0.9230574857
864813
depth =  130 Train Score 0.9633355259828973 test Score 0.923057485
7864813
```



Depth

In [12]:
```
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',rand

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 [0.9613815  0.96028819 0.95883605 0.96086253 0.96
25841 ]
mean train scores [0.96228379 0.96095398 0.9593384  0.96169181 0.9
6400173]
```

In [13]:
```
print(rf_random.best_estimator_)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight
=None,
                       criterion='gini', max_depth=14, max_feature
s='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_spl
it=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 [14]:
```
clf = RandomForestClassifier(bootstrap=True, class_weight=None, cri
            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=
            oob_score=False, random_state=25, verbose=0, warm_start
```

In [15]:

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

In [16]:

```
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 0.9643103378350976
Test f1 score 0.9236383580137536
```

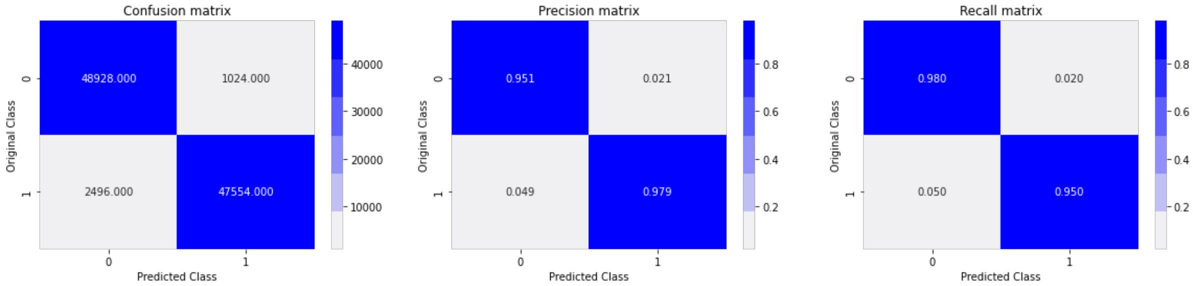
In [17]:

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

    plt.show()
```

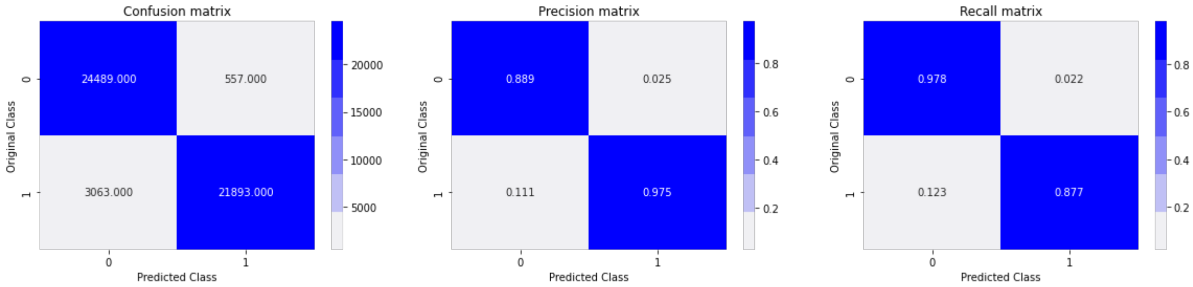
In [18]:

```
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
```

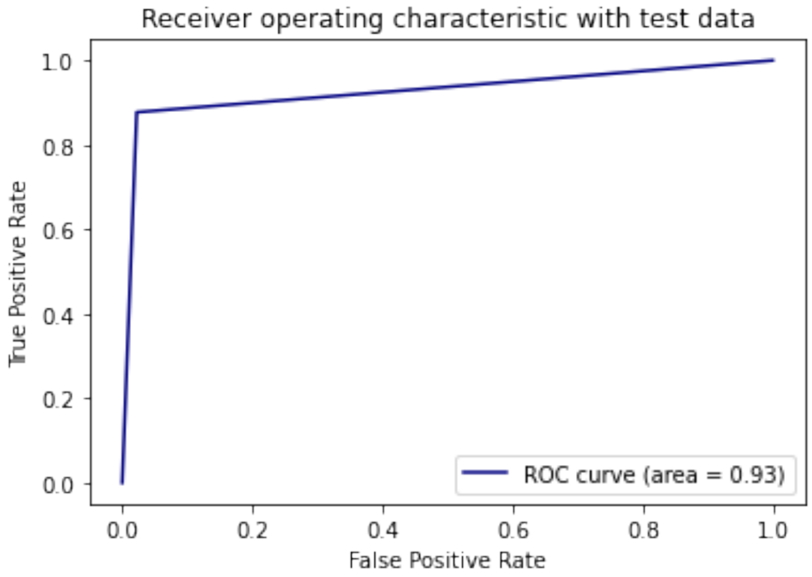


```
Test confusion_matrix
```

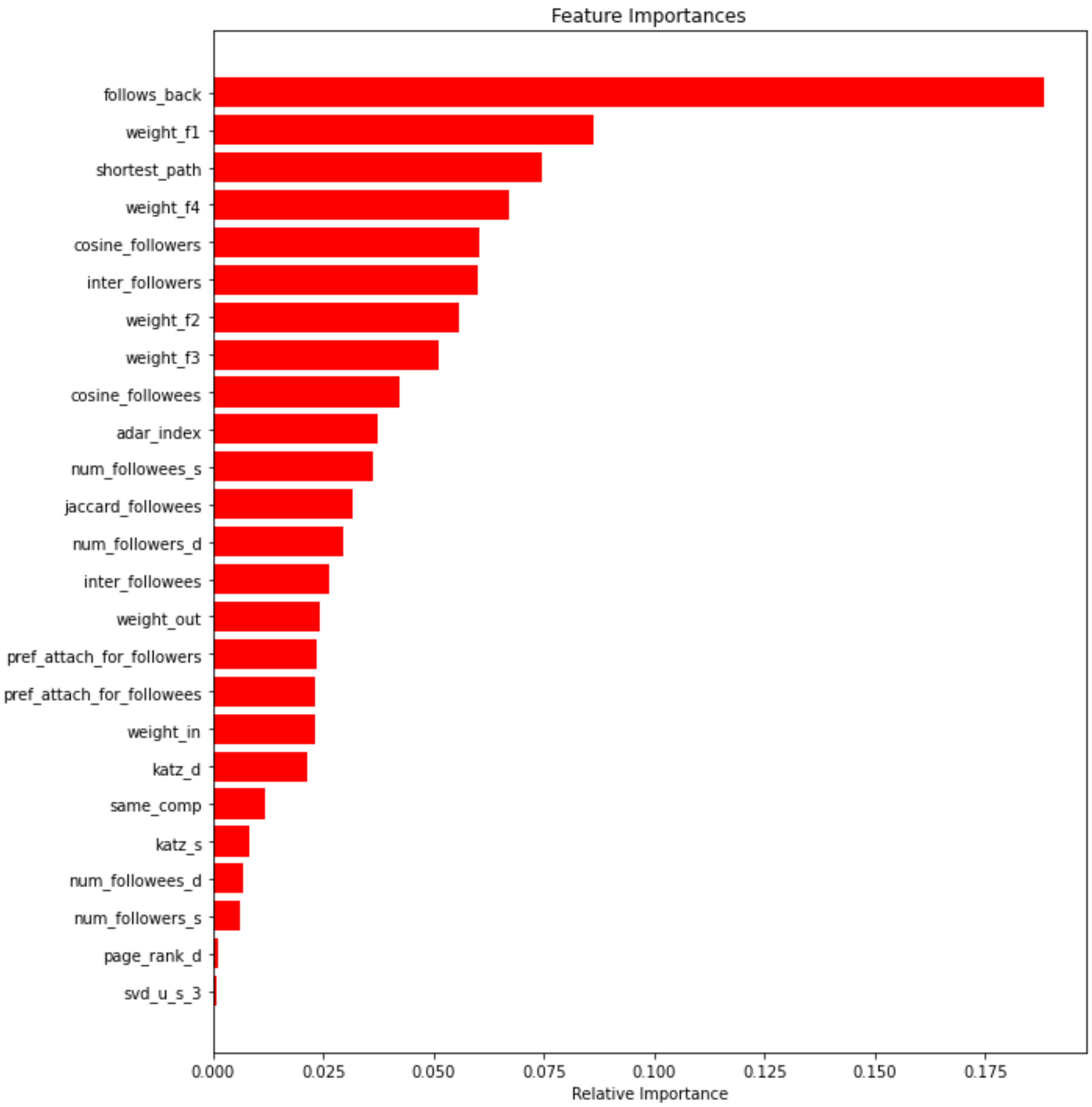


In [19]:

```
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)' %
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic with test data')
plt.legend()
plt.show()
```



In [20]:
```
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', alig
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.

Applying XGBoost

In [28]:

```
clf = xgb.XGBClassifier()
param_dist = {
    "n_estimators" :sp_randint(50,125),
    "max_depth": sp_randint(10,20)
}

model = RandomizedSearchCV(clf, param_distributions=param_dist, n_i
model.fit(df_final_train, y_train)

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

```
mean train scores [0.9998851  0.99925026 0.98984567 0.99765717 0.9
9997003]
mean test scores [0.9788322  0.97784751 0.97645552 0.97772971 0.97
893365]
```

In [29]:

```
print(model.best_estimator_)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=
1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=14,
              min_child_weight=1, missing=None, n_estimators=119,
n_jobs=1,
              nthread=None, objective='binary:logistic', random_st
ate=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=
None,
              silent=None, subsample=1, verbosity=1)
```

In [30]:
```
clf = xgb.XGBClassifier(base_score=0.5, booster='gbtree', colsample
              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=119, n
              nthread=None, objective='binary:logistic', random_sta
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=N
              silent=None, subsample=1, verbosity=1)
```

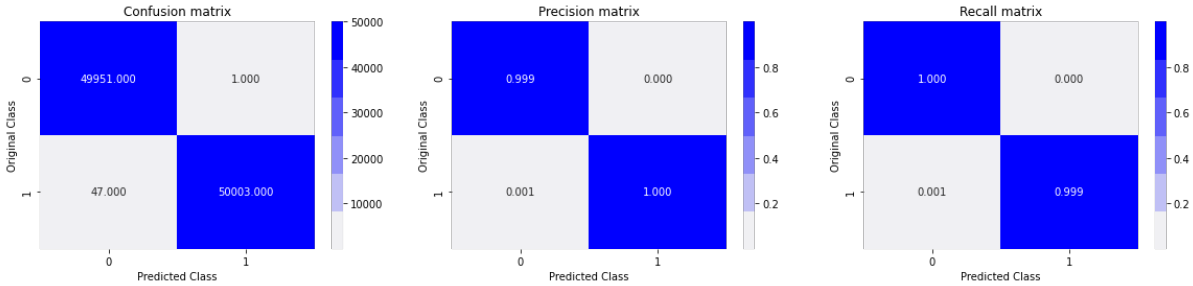
In [31]:
```
clf.fit(df_final_train, y_train)
y_train_pred = clf.predict(df_final_train)
y_test_pred = clf.predict(df_final_test)
```

In [32]:
```
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))
```

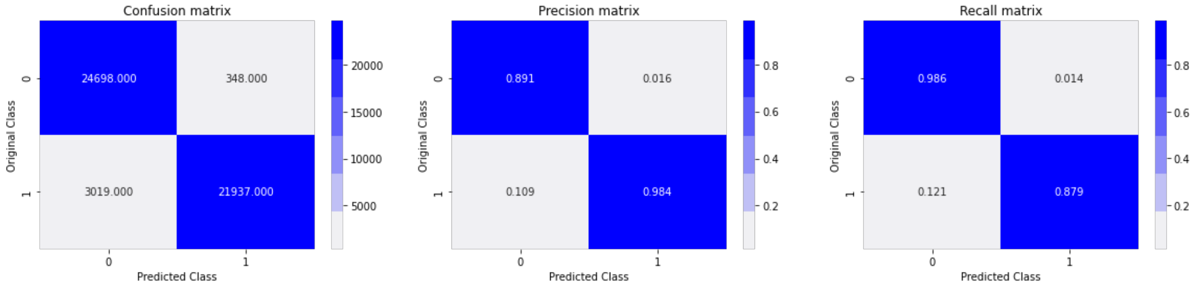
```
Train f1 score 0.9995202590601077
Test f1 score 0.9287271649626384
```

In [33]:
```
print('Train confusion_matrix')
plot_confusion_matrix(y_train,y_train_pred)
print('Test confusion_matrix')
plot_confusion_matrix(y_test,y_test_pred)
```

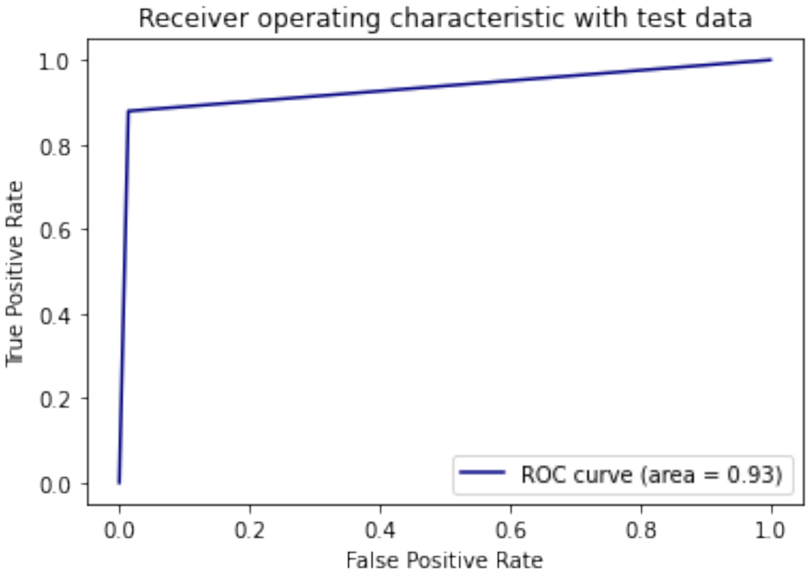
```
Train confusion_matrix
```



```
Test confusion_matrix
```

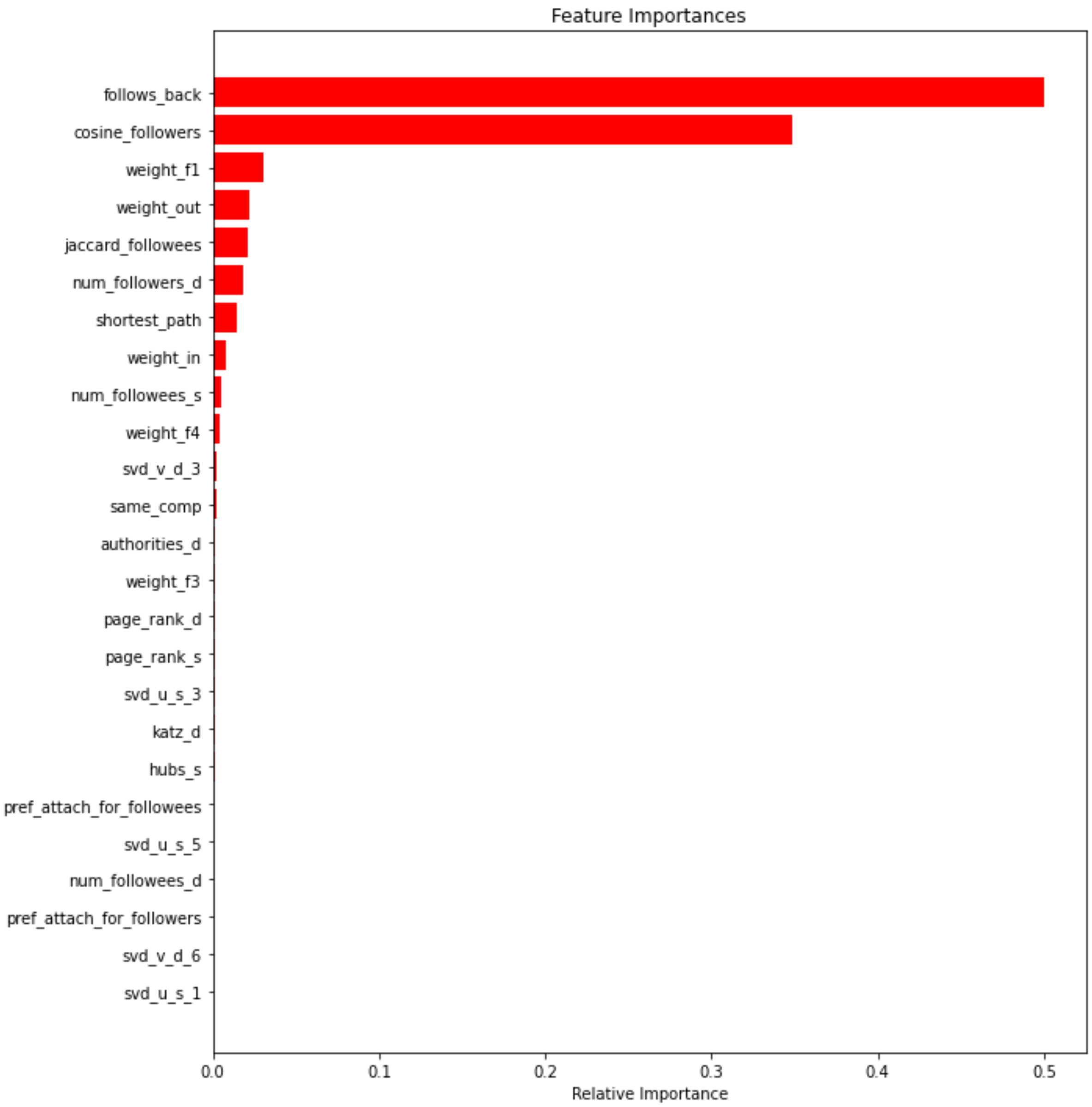


In [34]:
```
from sklearn.metrics import roc_curve, auc
fpr,tpr,ths = roc_curve(y_test,y_test_pred)
auc_sc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='navy',label='ROC curve (area = %0.2f)' %
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic with test data')
plt.legend()
plt.show()
```



In [35]:

```
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', alig
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



Observation

In [37]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model", "n_estimators", "max_depth", "Train f1-Sc
x.add_row(['Random Forest','121','14','0.964','0.924'])
x.add_row(['XGBoost','119','14','0.999','0.928'])
print(x)
```

```
+---------------+--------------+-----------+----------------+-----
----------+
|     Model     | n_estimators | max_depth | Train f1-Score | Test
f1-Score |
+---------------+--------------+-----------+----------------+-----
----------+
| Random Forest |     121      |     14    |     0.964      |
0.924     |
|    XGBoost    |     119      |     14    |     0.999      |
0.928     |
+---------------+--------------+-----------+----------------+-----
----------+
```