Social network Graph Link Prediction - Facebook Challenge

In [12]:
```
#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 [ ]:
```
!wget --header="Host: doc-0o-bk-docs.googleusercontent.com" --header="User-Agent: Mozilla/5.0 (Windows NT 10.0; W
```

```
--2021-06-12 16:01:49--  https://doc-0o-bk-docs.googleusercontent.com/docs/securesc/nss2f5s2soorprev6d4t4qp3n5ekp
9nh/evl2j2j4t5hronicnhsbdlsblnbl9qk3/1622116650000/06629147635963609455/13017565264516993811/1fDJptlCFEWNV5UNGPc4
geTykgFI3PDCV?e=download&authuser=0&nonce=iak2ig7rpq664&user=13017565264516993811&hash=fvl5s6dohfnqle6k8q3koe9jr2
mhe6jr
Resolving doc-0o-bk-docs.googleusercontent.com (doc-0o-bk-docs.googleusercontent.com)... 64.233.170.132, 2607:f8b
0:400c:c0d::84
Connecting to doc-0o-bk-docs.googleusercontent.com (doc-0o-bk-docs.googleusercontent.com)|64.233.170.132|:443...
connected.
HTTP request sent, awaiting response... 403 Forbidden
2021-06-12 16:01:49 ERROR 403: Forbidden.
```

In [14]:
```
#reading
from pandas import read_hdf
df_final_train = read_hdf('/content/drive/MyDrive/data fb_rec/storage_sample_stage4.h5', 'train_df',mode='r')
df_final_test = read_hdf('/content/drive/MyDrive/data fb_rec/storage_sample_stage4.h5', 'test_df',mode='r')
```

In [ ]:

In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [15]:
```
df_final_train.columns
```

Out[15]:
```
Index(['source_node', 'destination_node', 'indicator_link', 'num_followers_s',
       'num_followees_s', 'num_followers_d', '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')
```

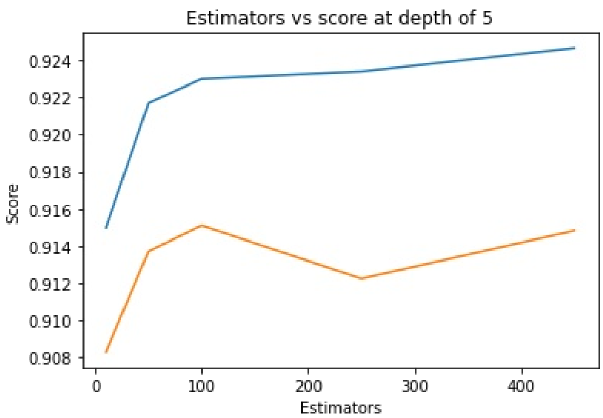
In [16]:
```
y_train = df_final_train.indicator_link
y_test = df_final_test.indicator_link
```

In [17]:
```
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 [19]:
```
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_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 0.9149747651539691 test Score 0.9082928887951045
Estimators =  50 Train Score 0.9216745159602303 test Score 0.9137047235755136
Estimators =  100 Train Score 0.922984999528287 test Score 0.9151047409040793
Estimators =  250 Train Score 0.9233818116998187 test Score 0.9122389075594504
Estimators =  450 Train Score 0.9246385920804526 test Score 0.9148240886135042
```

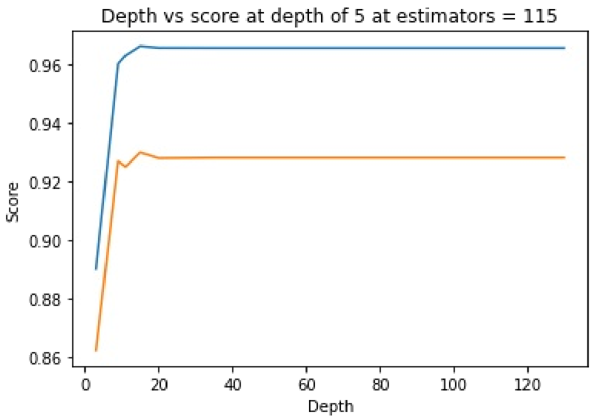
Out[19]:
```
Text(0.5, 1.0, 'Estimators vs score at depth of 5')
```



In [21]:
```
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_samples_leaf=52, min_samples_split=120,
            min_weight_fraction_leaf=0.0, n_estimators=115, n_jobs=-1,random_state=25,verbose=0,warm_start=False)
    clf.fit(df_final_train,y_train)
    train_sc = f1_score(y_train,clf.predict(df_final_train))
    test_sc = f1_score(y_test,clf.predict(df_final_test))
    test_scores.append(test_sc)
    train_scores.append(train_sc)
    print('depth = ',i,'Train Score',train_sc,'test Score',test_sc)
plt.plot(depths,train_scores,label='Train Score')
plt.plot(depths,test_scores,label='Test Score')
plt.xlabel('Depth')
plt.ylabel('Score')
plt.title('Depth vs score at depth of 5 at estimators = 115')
plt.show()
```

```
depth =  3 Train Score 0.8899857316493157 test Score 0.8620764647035797
depth =  9 Train Score 0.9599729325158407 test Score 0.9267817017848814
```

```
depth =  11 Train Score 0.9628765314582246 test Score 0.9248554913294798
depth =  15 Train Score 0.9660836439601851 test Score 0.9297843436916284
depth =  20 Train Score 0.9654988895370854 test Score 0.9278009003652424
depth =  35 Train Score 0.9654769949014067 test Score 0.927964661909616
depth =  50 Train Score 0.9654769949014067 test Score 0.927964661909616
depth =  70 Train Score 0.9654769949014067 test Score 0.927964661909616
depth =  130 Train Score 0.9654769949014067 test Score 0.927964661909616
```



In [22]:

```
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 [0.96402373 0.96317869 0.96192849 0.96366566 0.96565916]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-22-0386232e104c> in <module>()
     18 rf_random.fit(df_final_train,y_train)
     19 print('mean test scores',rf_random.cv_results_['mean_test_score'])
---> 20 print('mean train scores',rf_random.cv_results_['mean_train_score'])

KeyError: 'mean_train_score'
```

In [27]:

```
rf_random.cv_results_
```

Out[27]:

```
{'mean_fit_time': array([18.50797713, 17.14984212, 16.07933903, 16.8669075 , 19.5481878 ]),
 'mean_score_time': array([0.21306698, 0.13201284, 0.12166493, 0.16314764, 0.21214581]),
 'mean_test_score': array([0.96402373, 0.96317869, 0.96192849, 0.96366566, 0.96565916]),
 'param_max_depth': masked_array(data=[14, 12, 11, 13, 14],
              mask=[False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_min_samples_leaf': masked_array(data=[51, 33, 56, 49, 28],
              mask=[False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_min_samples_split': masked_array(data=[125, 138, 179, 165, 111],
              mask=[False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_n_estimators': masked_array(data=[117, 109, 106, 108, 121],
              mask=[False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'max_depth': 14,
```

```
  'min_samples_leaf': 51,
  'min_samples_split': 125,
  'n_estimators': 117},
 {'max_depth': 12,
  'min_samples_leaf': 33,
  'min_samples_split': 138,
  'n_estimators': 109},
 {'max_depth': 11,
  'min_samples_leaf': 56,
  'min_samples_split': 179,
  'n_estimators': 106},
 {'max_depth': 13,
  'min_samples_leaf': 49,
  'min_samples_split': 165,
  'n_estimators': 108},
 {'max_depth': 14,
  'min_samples_leaf': 28,
  'min_samples_split': 111,
  'n_estimators': 121}],
'rank_test_score': array([2, 4, 5, 3, 1], dtype=int32),
'split0_test_score': array([0.96600451, 0.96515182, 0.9644722 , 0.96576392, 0.96575622]),
'split1_test_score': array([0.96684012, 0.96645253, 0.96552428, 0.96688471, 0.96904277]),
'split2_test_score': array([0.96347684, 0.96215221, 0.96265221, 0.96466503, 0.9654326 ]),
'split3_test_score': array([0.96157393, 0.96047512, 0.96008209, 0.96127049, 0.96322027]),
'split4_test_score': array([0.96159754, 0.96208482, 0.96071209, 0.96168818, 0.963668  ]),
'split5_test_score': array([0.96091339, 0.95976124, 0.95844476, 0.96072383, 0.96278783]),
'split6_test_score': array([0.9638009 , 0.96296296, 0.96117005, 0.96210873, 0.96658098]),
'split7_test_score': array([0.96350964, 0.96158587, 0.9600082 , 0.96328205, 0.96558787]),
'split8_test_score': array([0.96550312, 0.96381343, 0.96145544, 0.96412464, 0.96722148]),
'split9_test_score': array([0.96701726, 0.96734694, 0.96476356, 0.96614503, 0.96729354]),
'std_fit_time': array([0.50426135, 0.19035886, 0.08000463, 0.2814729 , 0.1661613 ]),
'std_score_time': array([0.00119622, 0.0398629 , 0.02982303, 0.0506443 , 0.00082804]),
'std_test_score': array([0.00212557, 0.00237088, 0.00222426, 0.00207351, 0.00188769])}
```

In [23]:
```
print(rf_random.best_estimator_)
```

```
RandomForestClassifier(max_depth=14, min_samples_leaf=28, min_samples_split=111,
                       n_estimators=121, n_jobs=-1, random_state=25)
```

In [24]:
```
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_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 [25]:
```
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
```

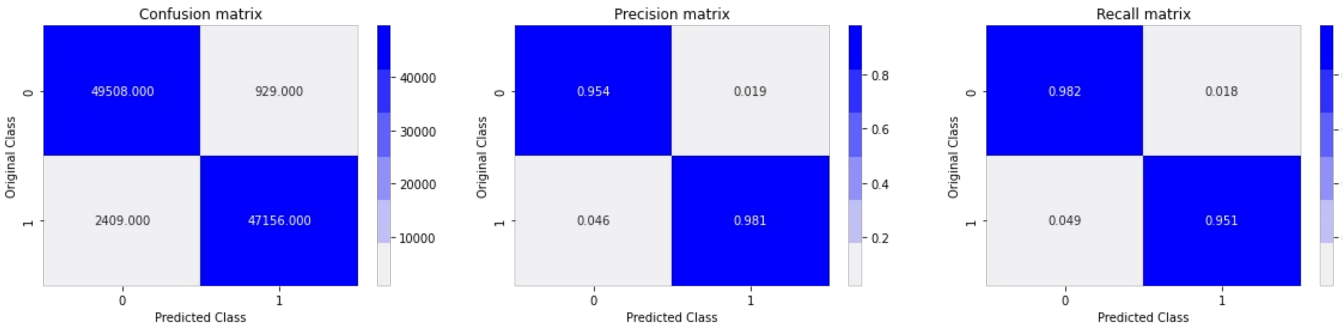
In [28]:
```
from sklearn.metrics import confusion_matrix
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)

    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()
```

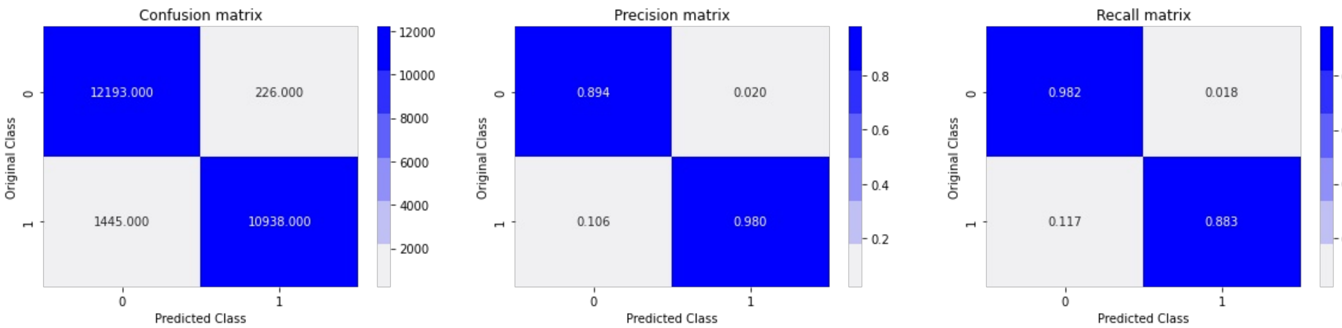
In [29]:

```
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

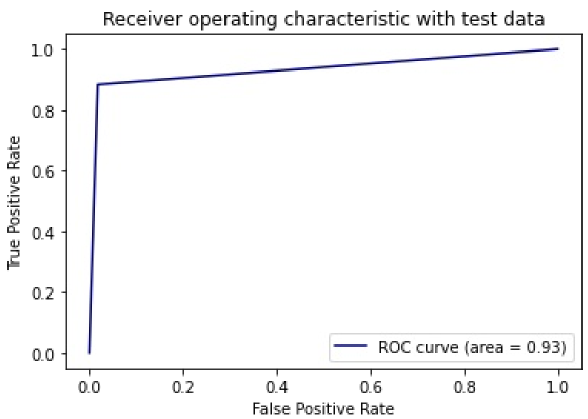


Test confusion_matrix



In [30]:

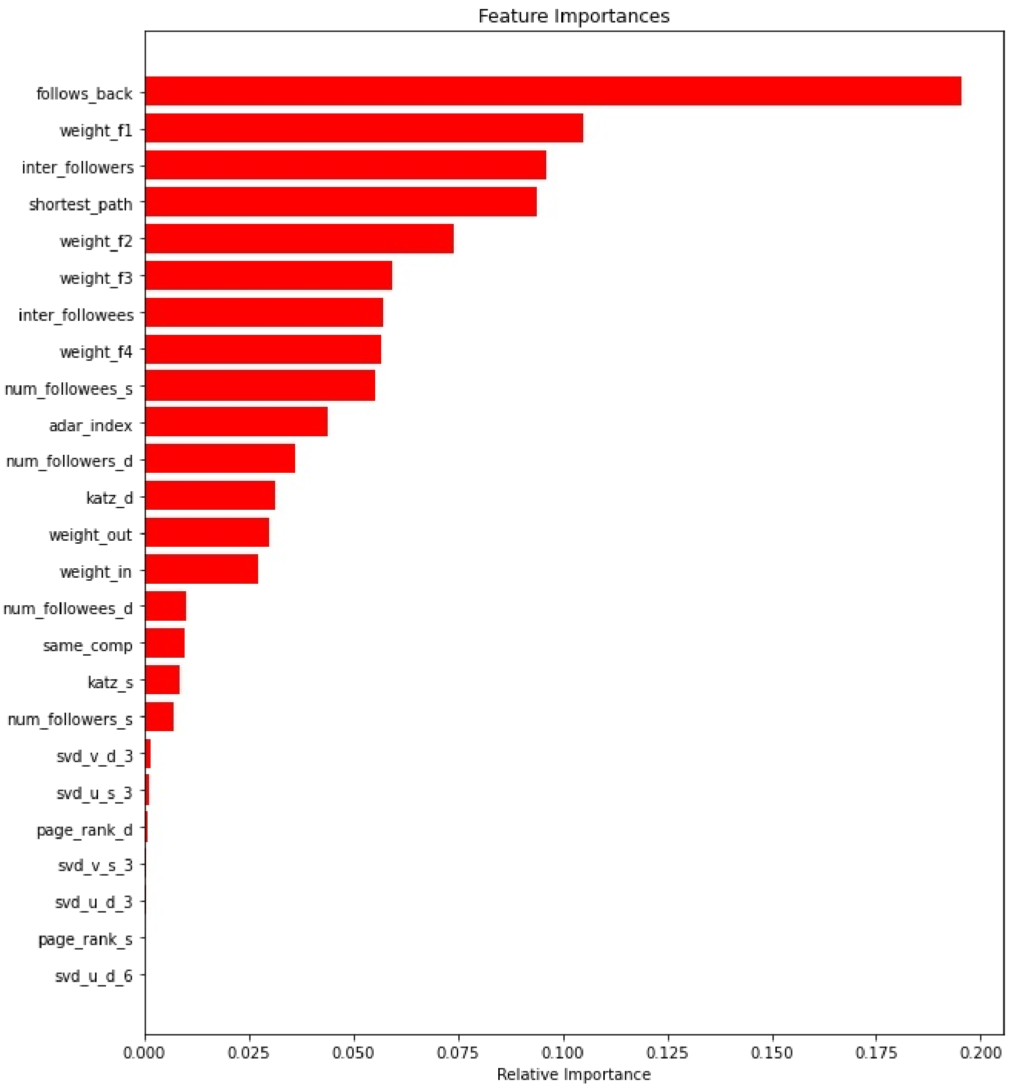
```
from sklearn.metrics import roc_curve, auc
fpr,tpr,ths = roc_curve(y_test,y_test_pred)
auc_sc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='navy',label='ROC curve (area = %0.2f)' % auc_sc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic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



In [31]:

```
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/
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
3. Tune hyperparameters for XG boost with all these features and check the error metric.

In [ ]:

Applying XGBoost on dataframe with newly added features

In [32]:

```
#reading a train and test with additionaly added features
from pandas import read_hdf
df_final_train = read_hdf('/content/drive/MyDrive/data fb_rec/storage_sample_stage5.h5', 'train_df',mode='r')
df_final_test = read_hdf('/content/drive/MyDrive/data fb_rec/storage_sample_stage5.h5', 'test_df',mode='r')
```

In [33]:

```
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 [39]:

```
xgb_clf = XGBClassifier()
params = { 'learning_rate' : [0.0001, 0.001, 0.01], 'n_estimators' : [20, 50, 100], 'max_depth' : [10, 12, 15] }
grid_search = GridSearchCV(xgb_clf, params, cv=3, scoring='f1', return_train_score=True)
grid_search.fit(df_final_train, y_train)
best_params_gridsearch_xgb = grid_search.best_params_
print("Best Params from GridSearchCV with XGB  ", best_params_gridsearch_xgb)
```

```
Best Params from GridSearchCV with XGB for Set s1  {'learning_rate': 0.01, 'max_depth': 15, 'n_estimators': 100}
```

In [41]:
```python
print('mean test scores',grid_search.cv_results_['mean_test_score'])
print('mean train scores',grid_search.cv_results_['mean_train_score'])
```

```
mean test scores [0.97187754 0.97186811 0.97187742 0.97203671 0.9720304  0.97204174
 0.97081741 0.97068493 0.9708362  0.97196541 0.97225502 0.97255809
 0.97205739 0.97265184 0.97333644 0.97104837 0.9716613  0.97239321
 0.97282636 0.97376652 0.97480159 0.97358898 0.97450248 0.97558985
 0.97308888 0.97449188 0.97563148]
mean train scores [0.97644571 0.97645713 0.97646689 0.98089414 0.98085061 0.9809056
 0.98609825 0.98601175 0.98593004 0.97658521 0.97673932 0.97670869
 0.98090414 0.98119781 0.98151554 0.98579379 0.98619726 0.98669611
 0.97682305 0.97802331 0.97969994 0.98188973 0.9833496  0.98496206
 0.98761736 0.98921555 0.99109315]
```

In [42]:
```python
#best params are {'learning_rate': 0.01, 'max_depth': 15, 'n_estimators': 100}
clf = XGBClassifier(max_depth=15, learning_rate=0.01, n_estimators=100)
```

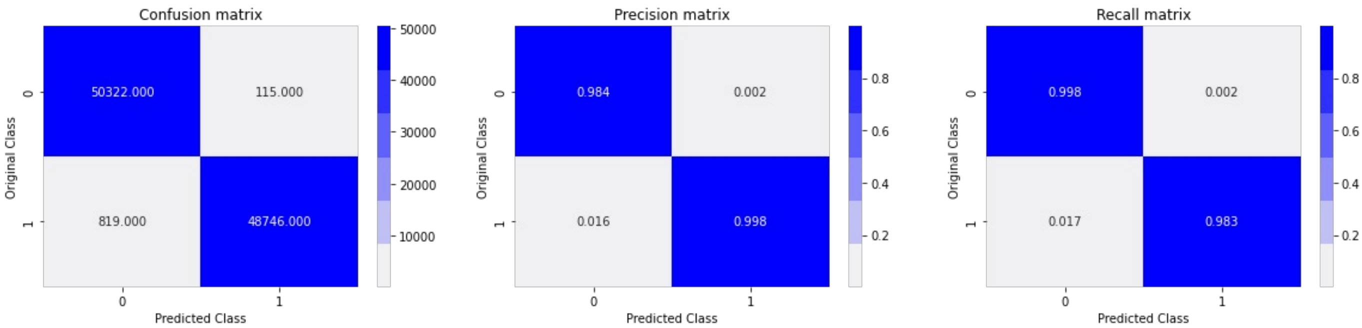
In [43]:
```python
clf.fit(df_final_train, y_train)
y_train_pred = clf.predict(df_final_train)
y_test_pred = clf.predict(df_final_test)
```

In [44]:
```python
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))
```

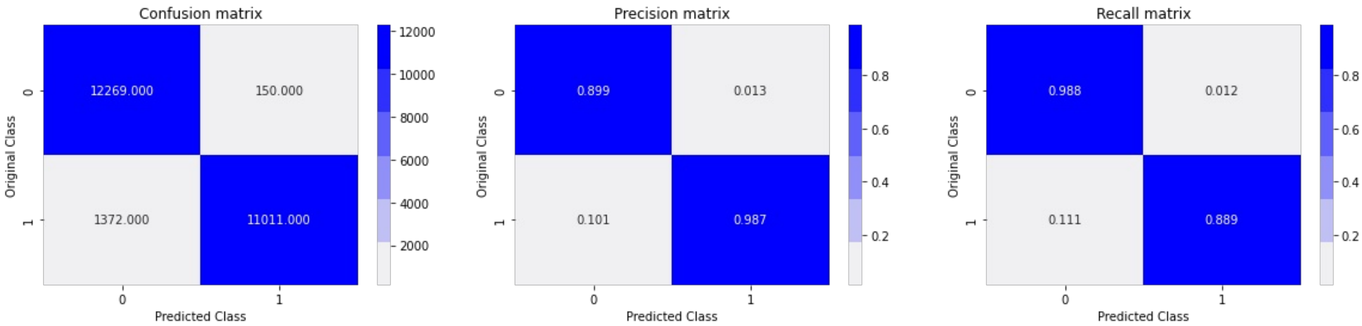
```
Train f1 score 0.9905106374331986
Test f1 score 0.9353550798504927
```

In [45]:
```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)
```

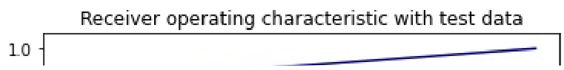
Train confusion_matrix

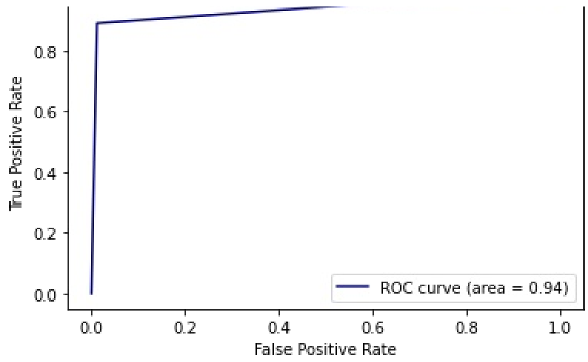


Test confusion_matrix

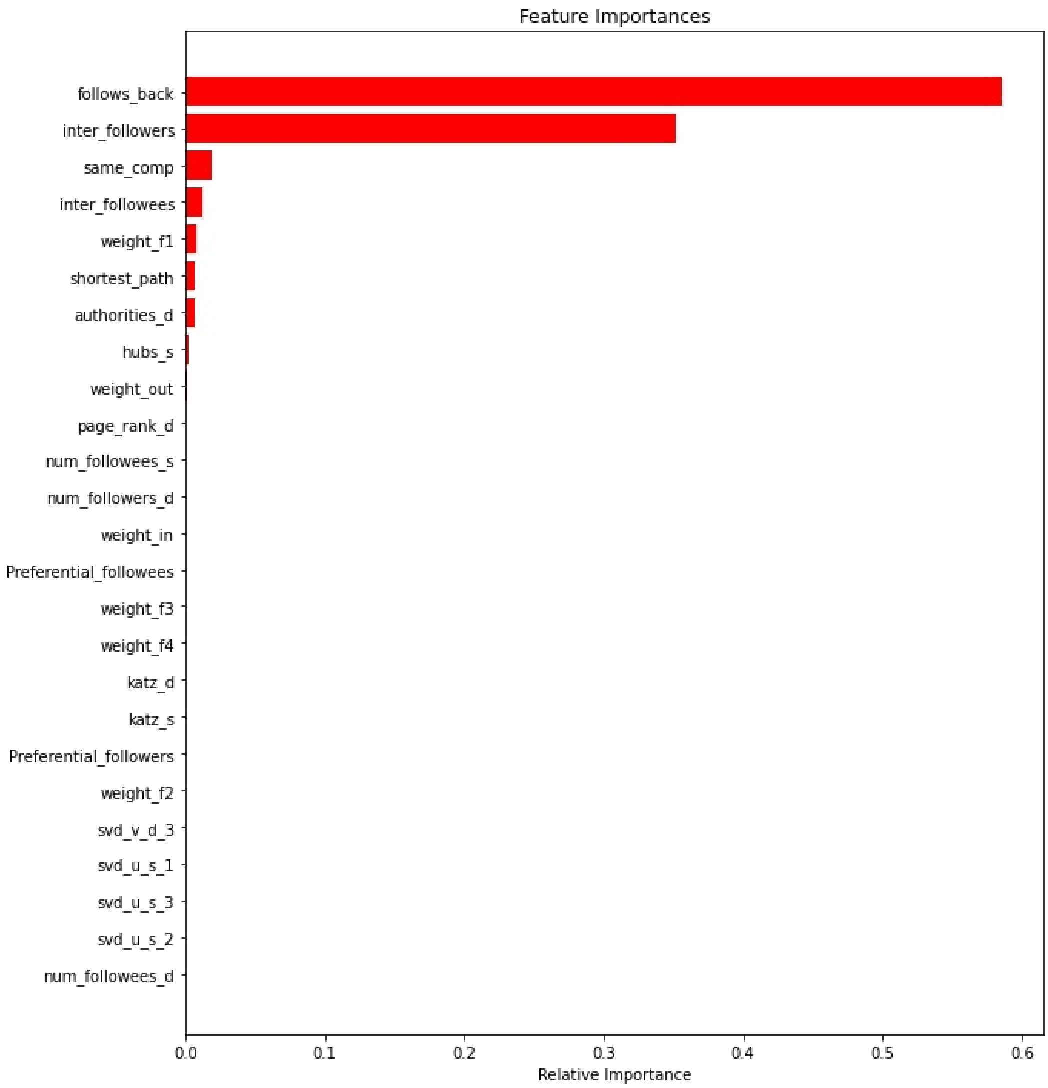


In [46]:
```python
from sklearn.metrics import roc_curve, auc
fpr,tpr,ths = roc_curve(y_test,y_test_pred)
auc_sc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='navy',label='ROC curve (area = %0.2f)' % auc_sc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic with test data')
plt.legend()
plt.show()
```





In [48]:
```
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()
```



Observation

1. Objective of the case study is finding, is link avail between the source and destination nodes which given as two features in the orginal dataset
2. we perform some featurization techniques already given and apply random forest to classify the target variable, find f1 score for train and test and plot the confusion matrix for precision and recall
3. After the random forest we add aditional features Preferential Attachment and dot product of SVD features of source and destination nodes.
4. XGBoost is now applied for dataset with additional features mentioned in step 3 to classify the target variable.
5. Evaluation metric used here is same f1 score and confusion matrix
6. Bellow table shows the result comparision of both Random Forest and XGBoost. Which f1 score of XGBoost is increased with 100

n_estimators

In [2]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model", "n_estimators", "max_depth", "Train f1-Score","Test f1-Score"]
x.add_row(['Random Forest','121','14','0.9652','0.9241'])
x.add_row(['XGBOOST','100','15','0.9905','0.9353'])
print(x)
```

```
+---------------+--------------+-----------+----------------+---------------+
|     Model     | n_estimators | max_depth | Train f1-Score | Test f1-Score |
+---------------+--------------+-----------+----------------+---------------+
| Random Forest |     121      |     14    |     0.9652     |     0.9241    |
|    XGBOOST    |     100      |     15    |     0.9905     |     0.9353    |
+---------------+--------------+-----------+----------------+---------------+
```

In [ ]: