Social network Graph Link Prediction - Facebook Challenge

In [20]:

```
#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 [21]:

```
#reading
from pandas import read_hdf
df_final_train = read_hdf('data/fea_sample/storage_sample_stage5.h5', 'train_df',mode='r')
df_final_test = read_hdf('data/fea_sample/storage_sample_stage5.h5', 'test_df',mode='r')
```

In [22]:

```
df_final_train.columns
```

Out[22]:

```
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', 'pref_attach_followers', 'pref_attach_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', 'svd_dot'],
      dtype='object')
```

In [23]:

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

In [24]:

```
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)
```

Here Preferential Attachmen and SVD_Dot features added to the dataframe

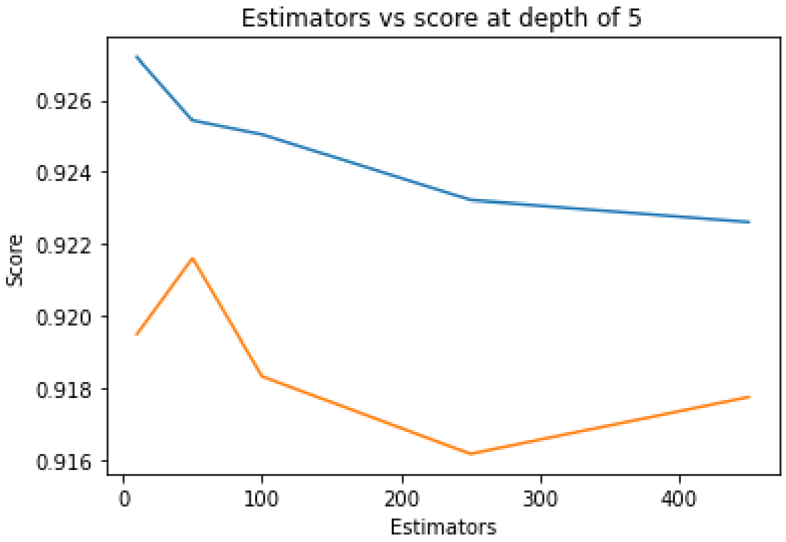
In [6]:

```
estimators = [10,50,100,250,450]
train_scores = []
```

```
test_scores = []
for i in estimators:
    clf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=5, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=52, min_samples_split=120,
            min_weight_fraction_leaf=0.0, n_estimators=i, n_jobs=-1,random_state=25,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 0.9271832513137042 test Score 0.9195044486044213
Estimators =  50 Train Score 0.925433132687778 test Score 0.9216086106497929
Estimators =  100 Train Score 0.9250437131579223 test Score 0.9183436940259193
Estimators =  250 Train Score 0.9232249294154554 test Score 0.9161914781693845
Estimators =  450 Train Score 0.9226215379466153 test Score 0.9177690983159045
```

Out[6]:
```
Text(0.5, 1.0, 'Estimators vs score at depth of 5')
```

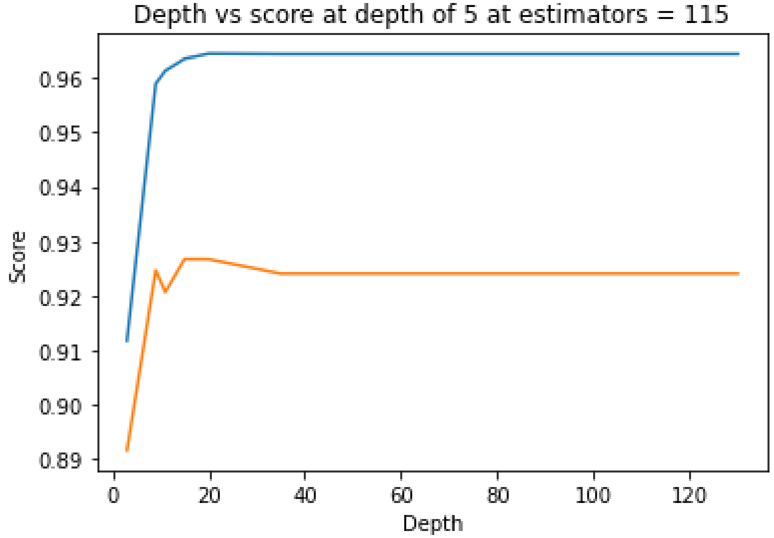


In [7]:

```
depths = [3,9,11,15,20,35,50,70,130]
train_scores = []
test_scores = []
for i in depths:
    clf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=i, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=52, min_samples_split=120,
            min_weight_fraction_leaf=0.0, n_estimators=115, n_jobs=-1,random_state=25,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 0.9118338557993729 test Score 0.891641126831521
```

```
depth =  9 Train Score 0.9589437437600602 test Score 0.9246813441483198
depth =  11 Train Score 0.9613180079737859 test Score 0.9206563462268528
depth =  15 Train Score 0.9634779780237652 test Score 0.9267377650381486
depth =  20 Train Score 0.9644030668127053 test Score 0.9267102689743645
depth =  35 Train Score 0.9643255521528383 test Score 0.9240690396408927
depth =  50 Train Score 0.9643255521528383 test Score 0.9240690396408927
depth =  70 Train Score 0.9643255521528383 test Score 0.9240690396408927
depth =  130 Train Score 0.9643255521528383 test Score 0.9240690396408927
```



In [9]:

```
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 [0.96197671 0.96197703 0.96093433 0.96185818 0.96297578]
```

In [10]:

```
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 [11]:

```
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 [12]:

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

In [13]:

```
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 0.9643993917891536
Test f1 score 0.9236279363741704
```

In [14]:

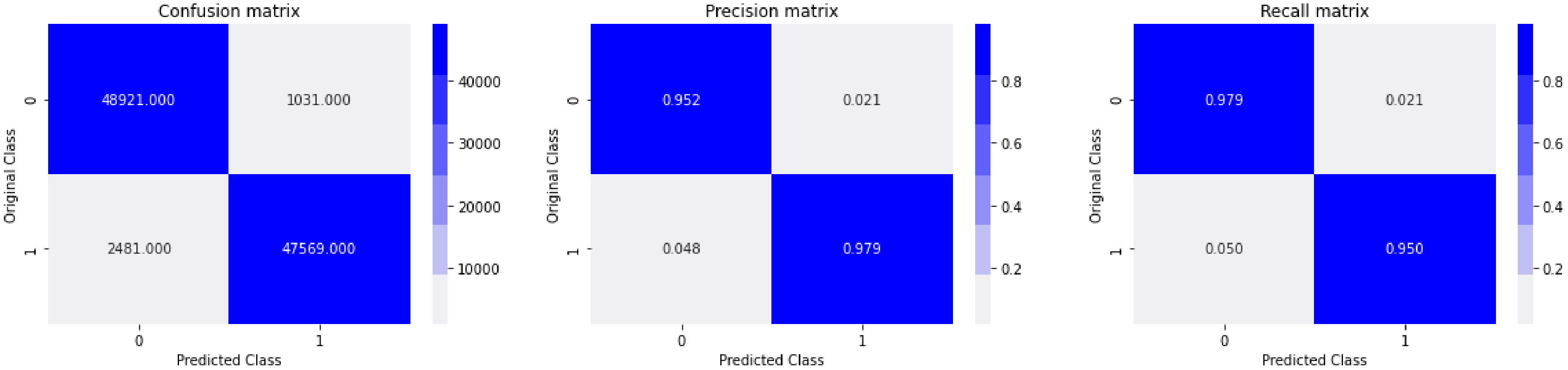
```
from sklearn.metrics import confusion_matrix
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)

    A =(((C.T)/(C.sum(axis=1))).T)

    B =(C/C.sum(axis=0))
```

```python
    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()
```

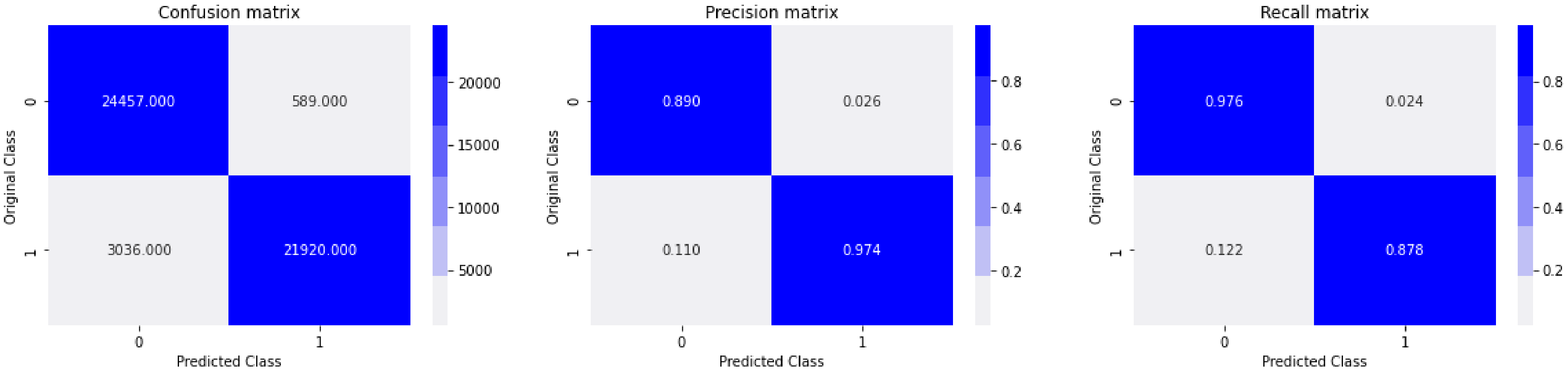
In [15]:

```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
```

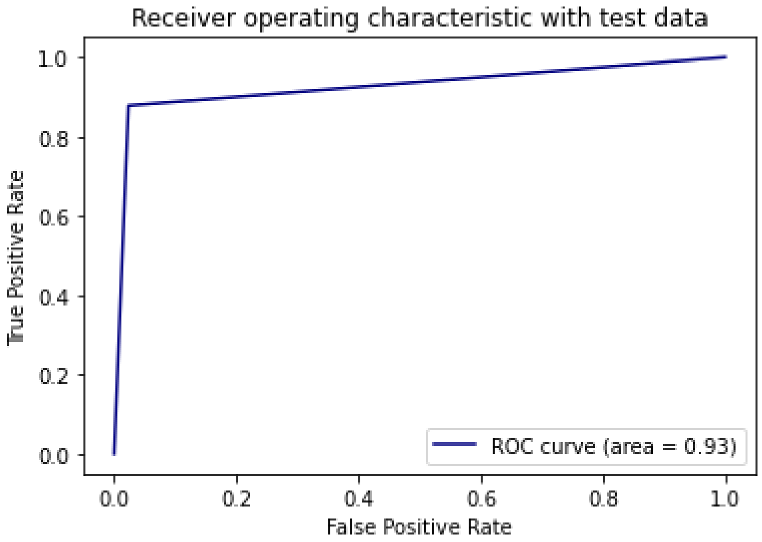


Test confusion_matrix



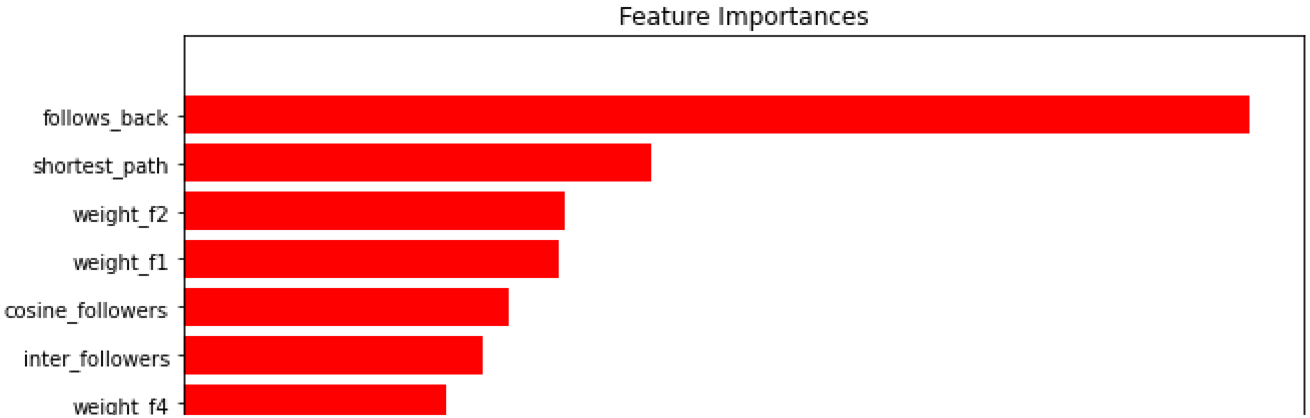
In [16]:

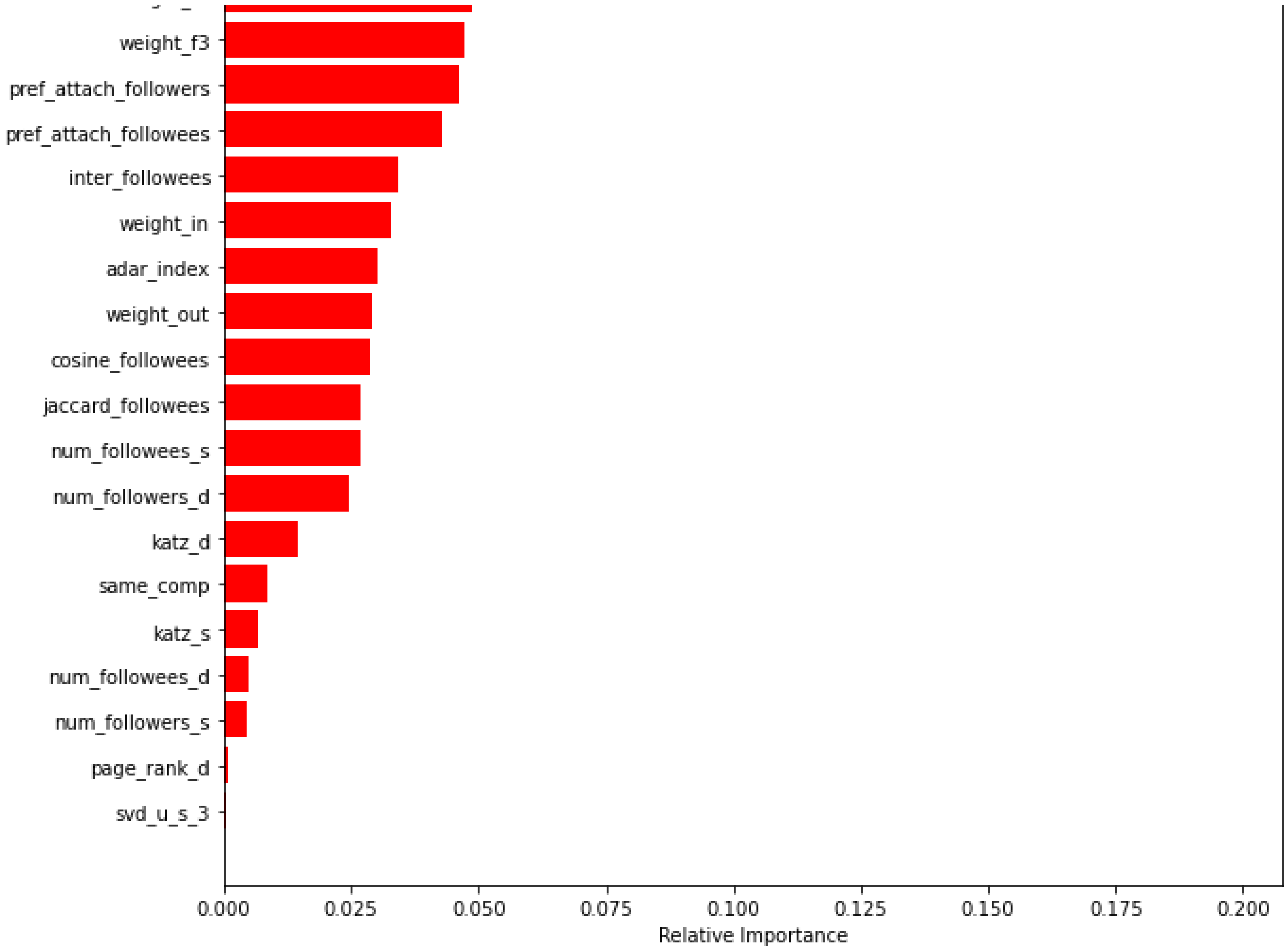
```
from sklearn.metrics import roc_curve, auc
fpr,tpr,ths = roc_curve(y_test,y_test_pred)
auc_sc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='navy',label='ROC curve (area = %0.2f)' % auc_sc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic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



In [17]:

```
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()
```





Here Preferenrial attachment features are there in the top 10 features

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.

XG boost

In [25]:
```
from sklearn.model_selection import RandomizedSearchCV
from xgboost import XGBClassifier
clf=XGBClassifier(nthread=-1,eval_metric='logloss')

params={
    'learning_rate':[0.01,0.05,0.1,0.15,1],
     'n_estimators':[100,200,500,1000,2000],
     'max_depth':[1,3,5,10,20]
}
search=RandomizedSearchCV(clf,param_distributions=params,cv=4,verbose=10,n_jobs=-1)
search.fit(df_final_train, y_train)
```

```
Fitting 4 folds for each of 10 candidates, totalling 40 fits
```

Out[25]:
```
RandomizedSearchCV(cv=4,
                   estimator=XGBClassifier(base_score=None, booster=None,
                                           colsample_bylevel=None,
                                           colsample_bynode=None,
                                           colsample_bytree=None,
                                           eval_metric='logloss', gamma=None,
                                           gpu_id=None, importance_type='gain',
                                           interaction_constraints=None,
                                           learning_rate=None,
                                           max_delta_step=None, max_depth=None,
                                           min_child_weight=None, missing=nan,
                                           monotone_constraints=...
                                           n_estimators=100, n_jobs=None,
                                           nthread=-1, num_parallel_tree=None,
                                           random_state=None, reg_alpha=None,
                                           reg_lambda=None,
                                           scale_pos_weight=None,
                                           subsample=None, tree_method=None,
                                           validate_parameters=None,
                                           verbosity=None),
                   n_jobs=-1,
```

```
                   param_distributions={'learning_rate': [0.01, 0.05, 0.1, 0.15,
                                                          1],
                                        'max_depth': [1, 3, 5, 10, 20],
                                        'n_estimators': [100, 200, 500, 1000,
                                                         2000]},
                   verbose=10)
```

In [26]:

```
print(search.best_estimator_)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, eval_metric='logloss',
              gamma=0, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.15, max_delta_step=0,
              max_depth=3, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=2000, n_jobs=8,
              nthread=-1, num_parallel_tree=1, random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [29]:

```
search.cv_results_
```

Out[29]:

```
{'mean_fit_time': array([ 857.01489031,  112.38152009,  307.77579129, 1293.68449301,
          23.10638642,  145.88762873,  711.17722774,  585.00323015,
        1356.16203922,   55.15535563]),
 'std_fit_time': array([4.32173387, 0.69246424, 0.22509718, 6.93610677, 0.74514928,
        1.07084191, 2.1249781 , 2.99934243, 6.02446555, 1.8554061 ]),
 'mean_score_time': array([0.27176958, 0.07500529, 0.13775975, 0.30202186, 0.0555042 ,
        0.09350675, 0.252518  , 0.2115168 , 0.68491399, 0.05150592]),
 'std_score_time': array([0.03648755, 0.01629451, 0.01399236, 0.01093073, 0.01373972,
        0.00585165, 0.01640757, 0.01614763, 0.20780426, 0.0132772 ]),
 'param_n_estimators': masked_array(data=[500, 1000, 1000, 500, 200, 200, 2000, 2000, 1000, 200],
              mask=[False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'param_max_depth': masked_array(data=[20, 1, 3, 20, 1, 20, 3, 3, 20, 3],
              mask=[False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'param_learning_rate': masked_array(data=[0.1, 0.15, 1, 0.05, 0.01, 1, 0.1, 0.15, 0.01, 0.05],
              mask=[False, False, False, False, False, False, False, False,
                    False, False],
```

```
        fill_value='?',
             dtype=object),
 'params': [{'n_estimators': 500, 'max_depth': 20, 'learning_rate': 0.1},
  {'n_estimators': 1000, 'max_depth': 1, 'learning_rate': 0.15},
  {'n_estimators': 1000, 'max_depth': 3, 'learning_rate': 1},
  {'n_estimators': 500, 'max_depth': 20, 'learning_rate': 0.05},
  {'n_estimators': 200, 'max_depth': 1, 'learning_rate': 0.01},
  {'n_estimators': 200, 'max_depth': 20, 'learning_rate': 1},
  {'n_estimators': 2000, 'max_depth': 3, 'learning_rate': 0.1},
  {'n_estimators': 2000, 'max_depth': 3, 'learning_rate': 0.15},
  {'n_estimators': 1000, 'max_depth': 20, 'learning_rate': 0.01},
  {'n_estimators': 200, 'max_depth': 3, 'learning_rate': 0.05}],
 'split0_test_score': array([0.98188072, 0.97448102, 0.98480061, 0.98116075, 0.88776449,
        0.9801208 , 0.98540058, 0.98556058, 0.97952082, 0.97420103]),
 'split1_test_score': array([0.98128075, 0.97452102, 0.9824007 , 0.98072077, 0.8949642 ,
        0.9799208 , 0.98440062, 0.98472061, 0.97924083, 0.97420103]),
 'split2_test_score': array([0.98088, 0.97328, 0.98272, 0.9802 , 0.88564, 0.978  , 0.98452,
        0.98516, 0.9788 , 0.97304]),
 'split3_test_score': array([0.981  , 0.97248, 0.98268, 0.98048, 0.88896, 0.97932, 0.9838 ,
        0.9846 , 0.97876, 0.97316]),
 'mean_test_score': array([0.98126037, 0.97369051, 0.98315033, 0.98064038, 0.88933217,
        0.9793404 , 0.9845303 , 0.9850103 , 0.97908041, 0.97365052]),
 'std_test_score': array([0.00038656, 0.00085856, 0.0009607 , 0.00035246, 0.00346223,
        0.0008281 , 0.00057176, 0.00037996, 0.00031662, 0.00055215]),
 'rank_test_score': array([ 4,  8,  3,  5, 10,  6,  2,  1,  7,  9])}
```

In [30]:

```
print('mean test scores',search.cv_results_['mean_test_score'])
```

```
mean test scores [0.98126037 0.97369051 0.98315033 0.98064038 0.88933217 0.9793404
 0.9845303  0.9850103  0.97908041 0.97365052]
```

In [32]:

```
clf=XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, eval_metric='logloss',
              gamma=0, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.15, max_delta_step=0,
              max_depth=3, min_child_weight=1,
              monotone_constraints='()', n_estimators=2000, n_jobs=8,
              nthread=-1, num_parallel_tree=1, random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
clf.fit(df_final_train,y_train)
y_train_pred = clf.predict(df_final_train)
y_test_pred = clf.predict(df_final_test)
```

In [33]:

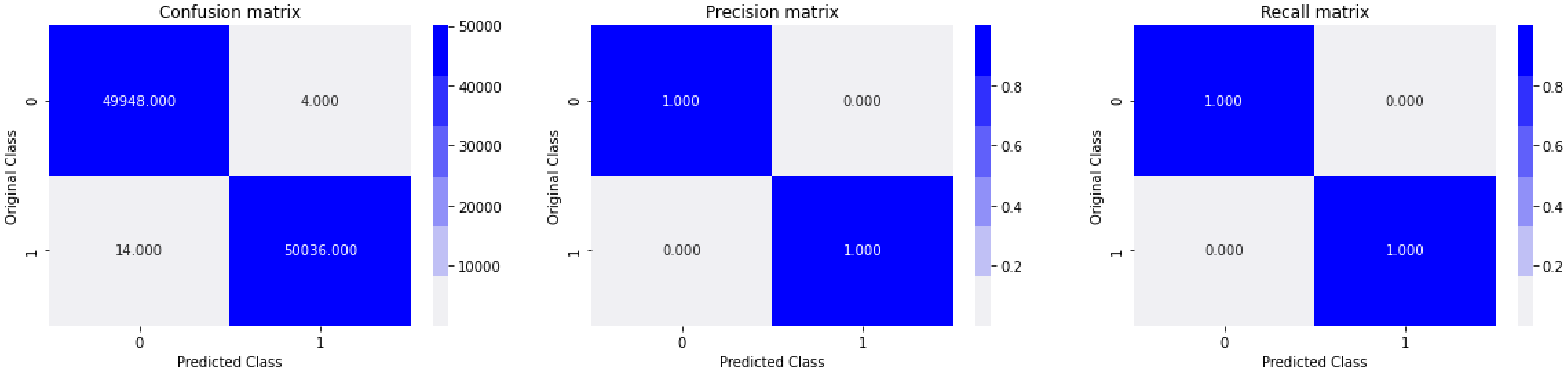
```
from sklearn.metrics import f1_score
print('Train f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))
```

```
Train f1 score 0.9998201618543312
Test f1 score 0.8677519171263286
```

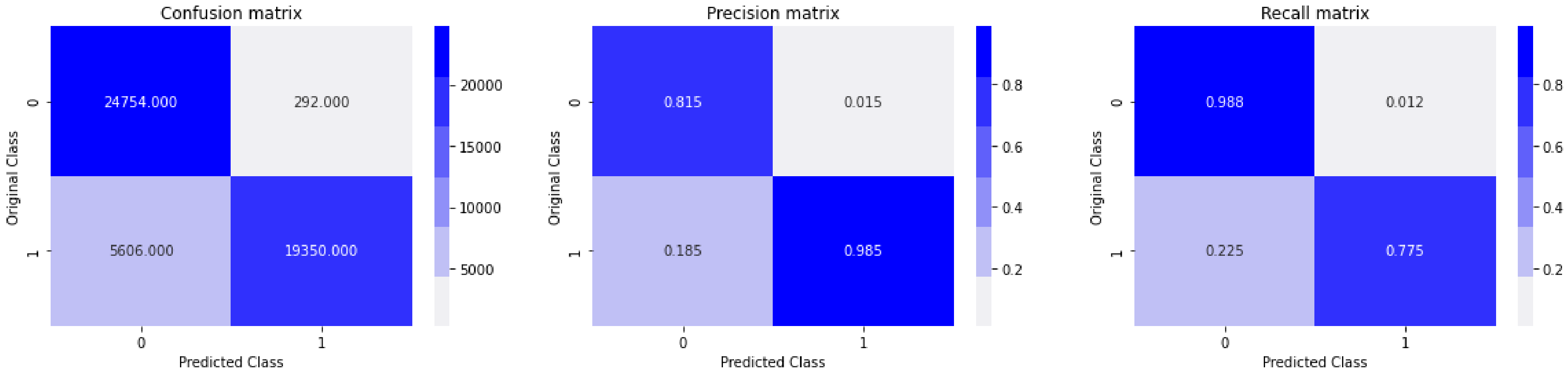
In [34]:

```
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
```

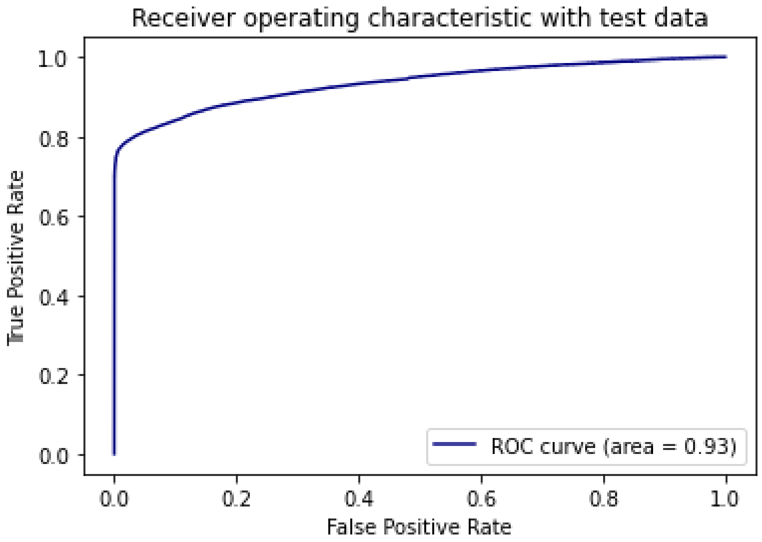



```
Test confusion_matrix
```



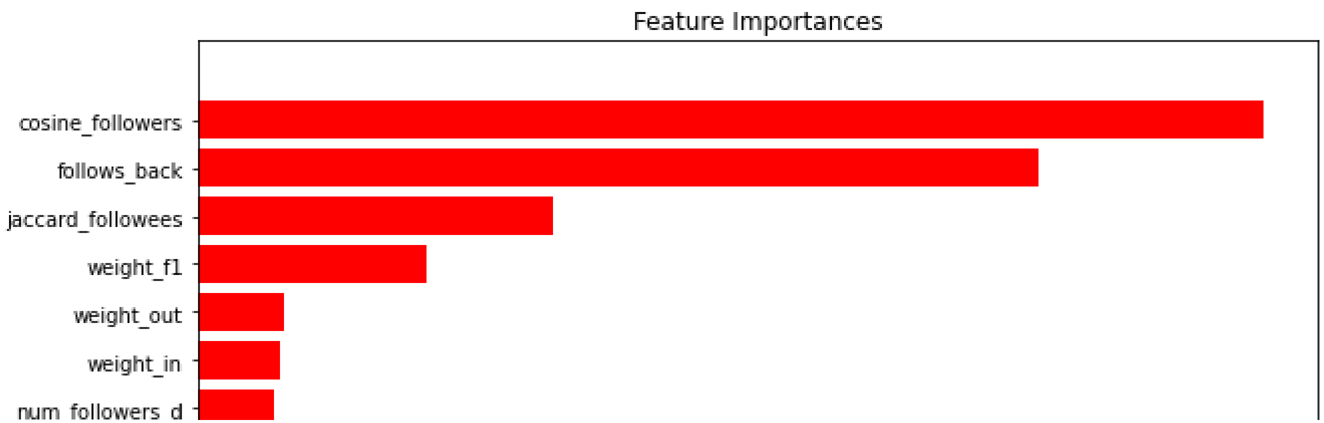
In [38]:

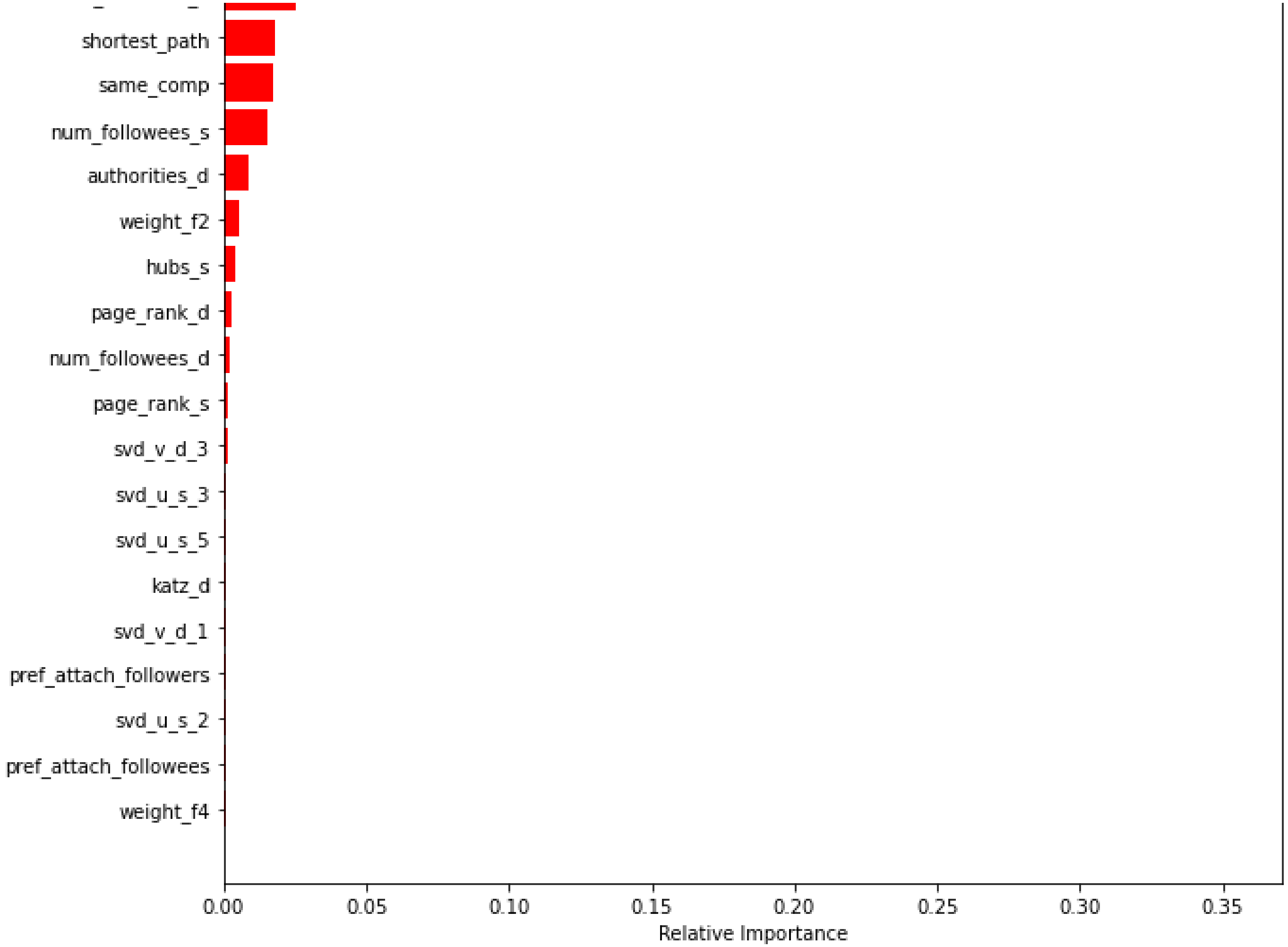
```
from sklearn.metrics import roc_curve, auc
fpr,tpr,ths = roc_curve(y_test,clf.predict_proba(df_final_test)[:,1])
auc_sc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='navy',label='ROC curve (area = %0.2f)' % auc_sc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic with test data')
plt.legend()
plt.show()
```



In [39]:

```
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()
```





Here Preferential Attachment features contributing more to Random Forest compared to XG boost and SVD_Dot feature is not contributing much to either of the models.

```
In [ ]:
```