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

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?
client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleuser
content.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=emai
l%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2
Fwww.googleapis.com%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2
Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fwww.googleapis.com%2Faut
h%2Fpeopleapi.readonly&response_type=code

Enter your authorization code:
..........
Mounted at /content/drive
```

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 [3]:
```
if os.path.isfile('drive/My Drive/Facebook/data/after_eda/train_pos_after_eda.csv'):
    train_graph=nx.read_edgelist('drive/My Drive/Facebook/data/after_eda/train_pos_after_eda.csv',delimiter=',',create_using=nx.DiGraph(),nodetype=int)
    print(nx.info(train_graph))
else:
    print("please run the FB_EDA.ipynb or download the files from drive")
```

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

loading all prevoius featurized data

In [0]:
```
df_final_train = read_hdf('drive/My Drive/Facebook/data/fea_sample/storage_sample_stage4.h5', 'train_df',mode='r')
df_final_test = read_hdf('drive/My Drive/Facebook/data/fea_sample/storage_sample_stage4.h5', 'test_df',mode='r')
```

In [5]:
```
print(len(df_final_train.columns))
print(len(df_final_test.columns))
```

```
54
54
```

Adding feature preferential attachement

In [0]:
```
#http://be.amazd.com/link-prediction/
def preferential_attachment(a,b):
  try:
    pred=np.dot(len(set(train_graph.neighbors(a))),len(set(train_graph.
neighbors(b))))
    return pred
  except:
    return 0
```

In [0]:
```
df_final_train['pref_attach'] =df_final_train.apply(lambda row: prefere
ntial_attachment(row['source_node'],row['destination_node']),axis=1)
df_final_test['pref_attach'] =df_final_test.apply(lambda row: preferent
ial_attachment(row['source_node'],row['destination_node']),axis=1)
```

In [8]:
```
print("printing few preferential_attachement values...")
print(df_final_train.pref_attach.head(5))
```

```
printing few preferential_attachement values...
0     120
1    8662
2     902
3      35
4      33
Name: pref_attach, dtype: int64
```

adding new feature svd_dot

In [0]:
```
def svd_dot(s_1,s_2,s_3,s_4,s_5,s_6,d_1,d_2,d_3,d_4,d_5,d_6):
    try:
      a=[s_1,s_2,s_3,s_4,s_5,s_6]
      b=[d_1,d_2,d_3,d_4,d_5,d_6]
      ans=np.dot(a,b)
      return ans
    except:
        return 0
```

In [0]:
```
#added new feature svd_u_dot and svd_v_dot to train data
df_final_train['svd_u_dot'] =df_final_train.apply(lambda row: svd_dot(r
ow['svd_u_s_1'],row['svd_u_s_2'],row['svd_u_s_3'],row['svd_u_s_4'],row[
'svd_u_s_5'],row['svd_u_s_6'],row['svd_u_d_1'],row['svd_u_d_2'],row['sv
d_u_d_3'],row['svd_u_d_4'],row['svd_u_d_5'],row['svd_u_d_6']),axis=1)
df_final_train['svd_v_dot'] =df_final_train.apply(lambda row: svd_dot(r
ow['svd_v_s_1'],row['svd_v_s_2'],row['svd_v_s_3'],row['svd_v_s_4'],row[
'svd_v_s_5'],row['svd_v_s_6'],row['svd_v_d_1'],row['svd_v_d_2'],row['sv
d_v_d_3'],row['svd_v_d_4'],row['svd_v_d_5'],row['svd_v_d_6']),axis=1)
```

In [0]:
```
#added new feature svd_u_dot and svd_v_dot to test data
df_final_test['svd_u_dot'] =df_final_test.apply(lambda row: svd_dot(row
['svd_u_s_1'],row['svd_u_s_2'],row['svd_u_s_3'],row['svd_u_s_4'],row['s
vd_u_s_5'],row['svd_u_s_6'],row['svd_u_d_1'],row['svd_u_d_2'],row['svd_
u_d_3'],row['svd_u_d_4'],row['svd_u_d_5'],row['svd_u_d_6']),axis=1)
df_final_test['svd_v_dot'] =df_final_test.apply(lambda row: svd_dot(row
['svd_v_s_1'],row['svd_v_s_2'],row['svd_v_s_3'],row['svd_v_s_4'],row['s
vd_v_s_5'],row['svd_v_s_6'],row['svd_v_d_1'],row['svd_v_d_2'],row['svd_
v_d_3'],row['svd_v_d_4'],row['svd_v_d_5'],row['svd_v_d_6']),axis=1)
```

In [21]:
```
print(len(df_final_train.columns))
print(len(df_final_test.columns))
```

```
57
57
```

In [0]:
```
#we added three features here that is :
#1.pref_attach
```

```
#2.svd_u_dot
#3.svd_v_dot
```

Machine learning model

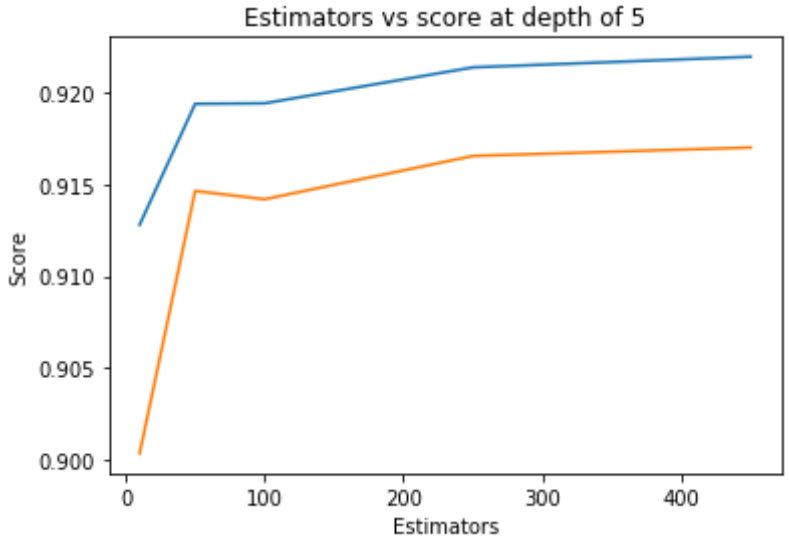
In [0]:
```
y_train = df_final_train.indicator_link
y_test = df_final_test.indicator_link
```

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 [25]:
```
estimators = [10,50,100,250,450]
train_scores = []
test_scores = []
for i in estimators:
    clf = RandomForestClassifier(bootstrap=True, class_weight=None, cri
terion='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,ran
dom_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 0.9128093143379071 test Score 0.9003502069
404649
Estimators =  50 Train Score 0.919392168567449 test Score 0.91464952092
7887
Estimators =  100 Train Score 0.9194221575990877 test Score 0.914196960
7925446
Estimators =  250 Train Score 0.921380953376105 test Score 0.9165547910
723276
Estimators =  450 Train Score 0.9219604521546362 test Score 0.917014800
041986
```

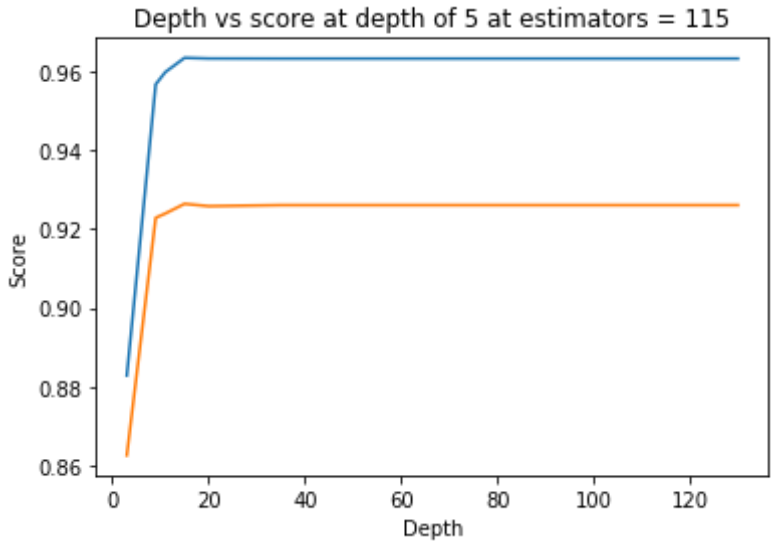
Out[25]: Text(0.5, 1.0, 'Estimators vs score at depth of 5')



In [26]:
```
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, cri
terion='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,r
andom_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 0.882871311104326 test Score 0.862624876731123
6
depth =  9 Train Score 0.9567402987514558 test Score 0.92288525697875
76
depth =  11 Train Score 0.9598402575439597 test Score 0.9239729062480
219
depth =  15 Train Score 0.9634670268842869 test Score 0.9264284510365
751
depth =  20 Train Score 0.9632966096536613 test Score 0.9258386295719
516
depth =  35 Train Score 0.9632726792981318 test Score 0.9260990399191
51
depth =  50 Train Score 0.9632726792981318 test Score 0.9260990399191
51
depth =  70 Train Score 0.9632726792981318 test Score 0.9260990399191
51
depth =  130 Train Score 0.9632726792981318 test Score 0.926099039919
151
```



In [27]:
```
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_s
tate=25)
rf_random.fit(df_final_train,y_train)
```

Out[27]:
```
RandomizedSearchCV(cv=10, error_score='raise-deprecating',
                   estimator=RandomForestClassifier(bootstrap=True,
                                                    class_weight=None,
                                                    criterion='gini',
                                                    max_depth=None,
                                                    max_features='aut
o',
                                                    max_leaf_nodes=Non
e,
                                                    min_impurity_decrea
se=0.0,
                                                    min_impurity_split=
None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=
2,
                                                    min_weight_fraction
_leaf=0.0,
                                                    n_estimators='war
n',
                                                    n_jobs=-1, oob_sc
o...
                                           'min_samples_leaf': <scipy.stat
s._distn_infrastructure.rv_frozen object at 0x7fee290bf208>,
                                           'min_samples_split': <scipy.sta
ts._distn_infrastructure.rv_frozen object at 0x7fee290bf908>,
                                           'n_estimators': <scipy.stats._d
istn_infrastructure.rv_frozen object at 0x7fee290bf6a0>},
                   pre_dispatch='2*n_jobs', random_state=25, refit=Tru
e,
                   return_train_score=False, scoring='f1', verbose=0)
```

In [28]:
```
print('mean test scores',rf_random.cv_results_.keys())
```

```
mean test scores dict_keys(['mean_fit_time', 'std_fit_time', 'mean_scor
e_time', 'std_score_time', 'param_max_depth', 'param_min_samples_leaf',
'param_min_samples_split', 'param_n_estimators', 'params', 'split0_test
_score', 'split1_test_score', 'split2_test_score', 'split3_test_score',
'split4_test_score', 'split5_test_score', 'split6_test_score', 'split7_
test_score', 'split8_test_score', 'split9_test_score', 'mean_test_scor
e', 'std_test_score', 'rank_test_score'])
```

In [29]:
```
print('mean test scores',rf_random.cv_results_['mean_test_score'])
#print('mean train scores',rf_random.cv_results_['mean_train_score'])#m
ean_train_score
```

```
mean test scores [0.96162089 0.9618011  0.9597477  0.96143249 0.9628469
4]
```

In [30]:
```
print(rf_random.best_estimator_)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gi
ni',
                       max_depth=14, max_features='auto', max_leaf_node
s=None,
                       min_impurity_decrease=0.0, min_impurity_split=No
ne,
                       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, ver
bose=0,
                       warm_start=False)
```

In [0]:
```
clf = RandomForestClassifier(bootstrap=True, class_weight=None, criteri
on='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=Fal
se)
```

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 [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))
```

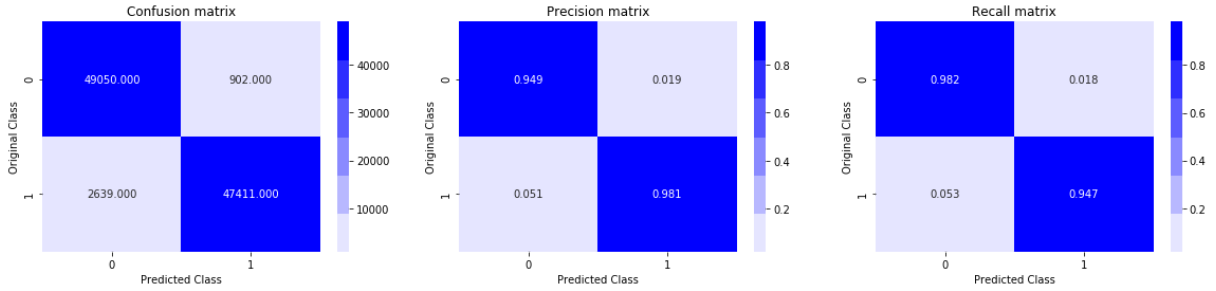
```
Train f1 score 0.964000691316857
Test f1 score 0.9264547697888764
```

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
```

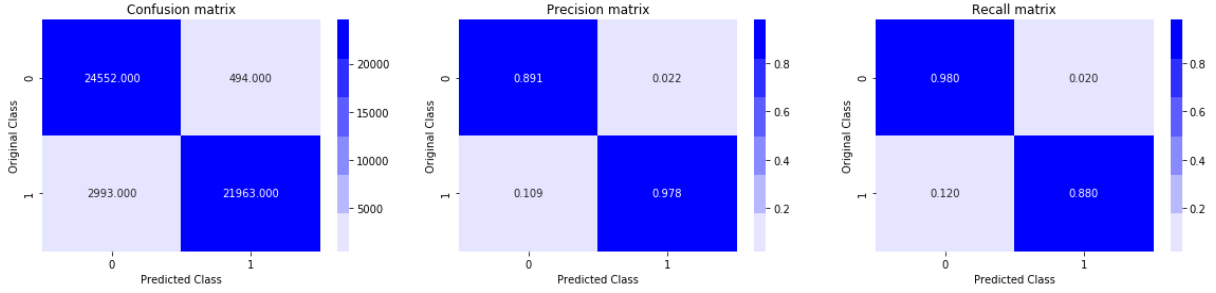
```
    sns.heatmap(A, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels
, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()
```

In [35]:
```
print('Train confusion_matrix')
plot_confusion_matrix(y_train,y_train_pred)
print('Test confusion_matrix')
plot_confusion_matrix(y_test,y_test_pred)
```

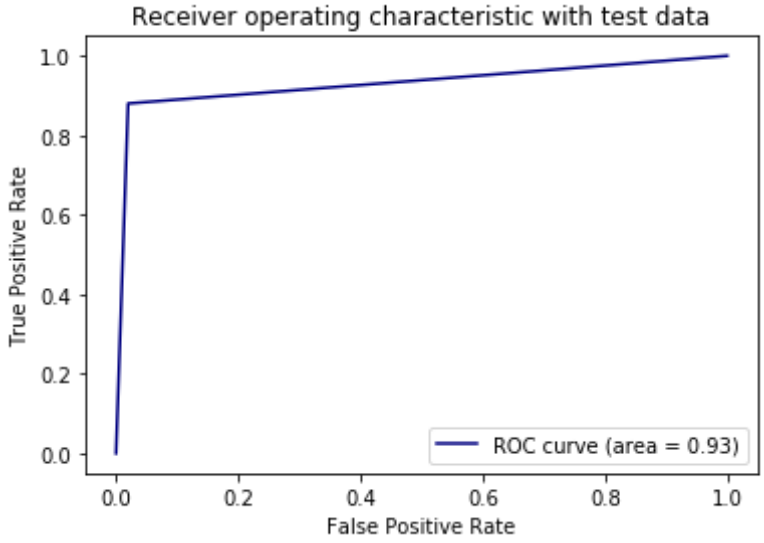
Train confusion_matrix



Test confusion_matrix

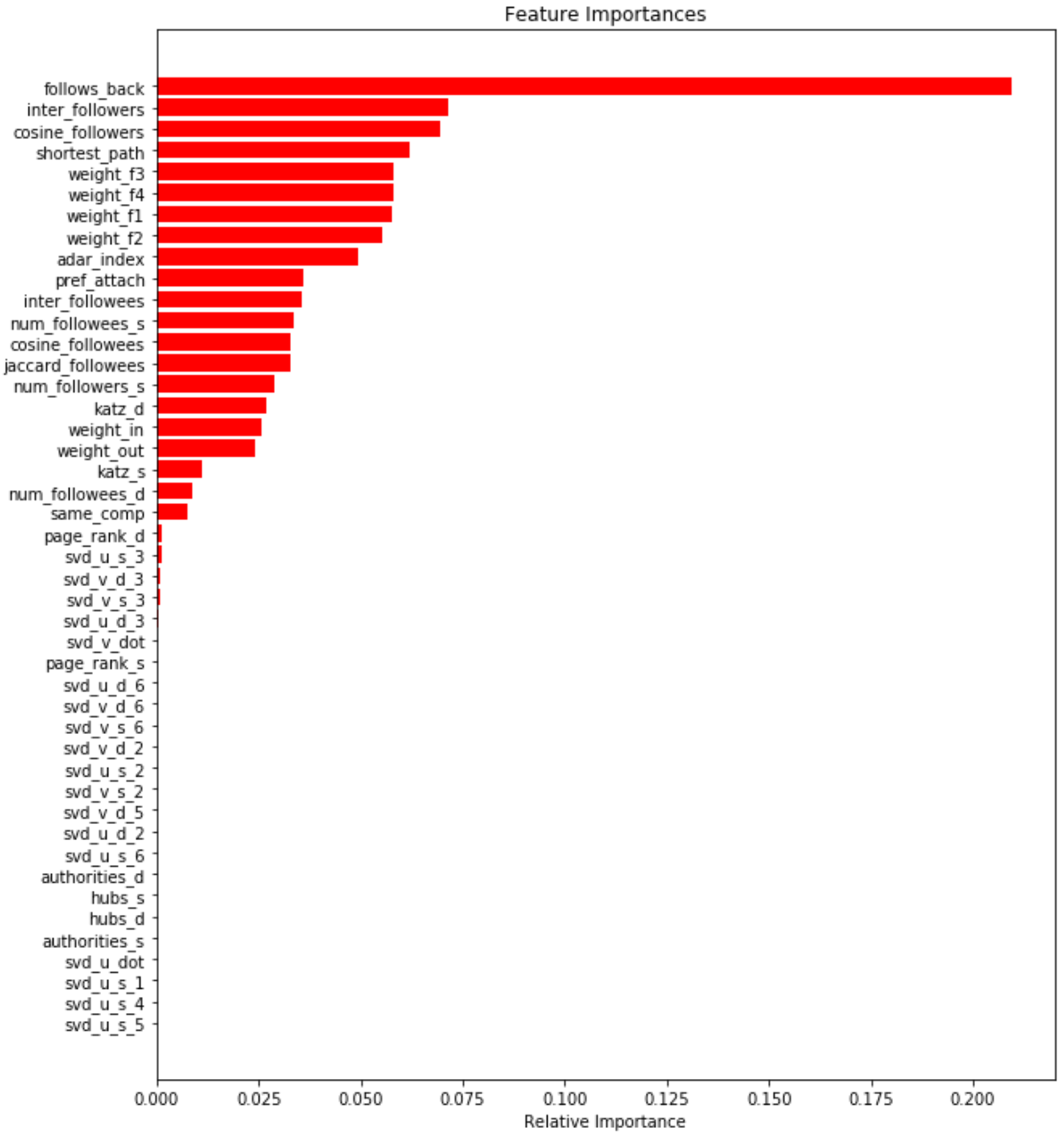


In [36]:
```
from sklearn.metrics import roc_curve, auc
fpr,tpr,ths = roc_curve(y_test,y_test_pred)
auc_sc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='navy',label='ROC curve (area = %0.2f)' % auc_
sc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic with test data')
plt.legend()
plt.show()
```



In [38]:
```
features = df_final_train.columns
importances = clf.feature_importances_
indices = (np.argsort(importances))[-45:]
plt.figure(figsize=(10,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='r', align='c
enter')
```

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



In [39]:
```
print("observation:")
print("by above feature importance we can say that preferential attacha
ment has good importance where as svd_u_dot and svd_v_dot has less impo
rtance")
```

```
observation:
by above feature importance we can say that preferential attachament ha
s good importance where as svd_u_dot and svd_v_dot has less importance
```