Social network Graph Link Prediction - Facebook Challenge

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 [2]:

```
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=947318989803-6bn6
qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%
b&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2
www.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly
ttps%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

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

In [0]:

```
#reading
from pandas import read_hdf
df_final_train = read_hdf('storage_sample_stage6.h5', 'train_df',mode='r')
df_final_test = read_hdf('storage_sample_stage6.h5', 'test_df',mode='r')
```

In [5]:

```
df_final_train.columns
```

Out[5]:

```
Index(['source_node', 'destination_node', 'indicator_link',
```

```
       'jaccard_followers', 'jaccard_followees', 'cosine_followers',
       'cosine_followees', 'num_followers_s', 'num_followers_d',
       'num_followees_s', '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', 'Svd_dot',
       'Pref_attach_followers', 'Pref_attach_followees'],
      dtype='object')
```

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)
```

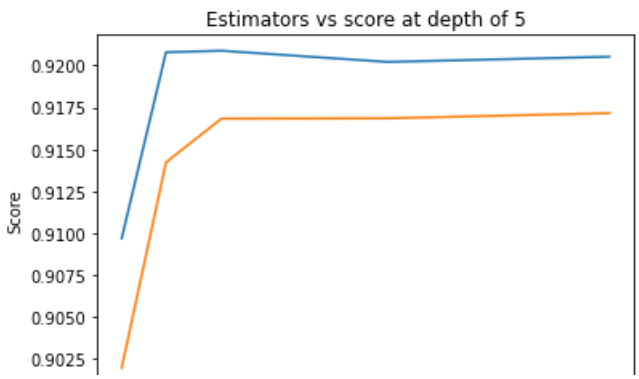
In [8]:

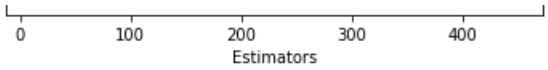
```
estimators = [10,50,100,250,450]
train_scores = []
test_scores = []
for i in estimators:
    clf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=5, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            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 0.9096838519380166 test Score 0.9019690163315001
Estimators =  50 Train Score 0.9207880477338929 test Score 0.9142242651710032
Estimators =  100 Train Score 0.9208781586189642 test Score 0.9168293963254592
Estimators =  250 Train Score 0.9202095464696429 test Score 0.916850532686125
Estimators =  450 Train Score 0.9205213017936705 test Score 0.9171551633574955
```

Out[8]:

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

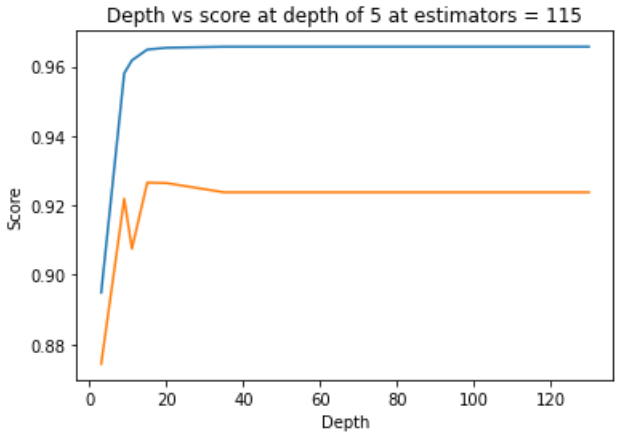




In [9]:

```
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,war
m_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 0.8949137770030505 test Score 0.8743882544861338
depth =  9 Train Score 0.958084563198633 test Score 0.9219588456500614
depth =  11 Train Score 0.9617955193152199 test Score 0.9075382047001413
depth =  15 Train Score 0.9649058747160012 test Score 0.9265896441221118
depth =  20 Train Score 0.965418685261387 test Score 0.9264467165595024
depth =  35 Train Score 0.9657170644729912 test Score 0.9237920179163762
depth =  50 Train Score 0.9657170644729912 test Score 0.9237920179163762
depth =  70 Train Score 0.9657170644729912 test Score 0.9237920179163762
depth =  130 Train Score 0.9657170644729912 test Score 0.9237920179163762
```



In [11]:

```
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,verbose=2)
```

```
rf_random.fit(df_final_train,y_train)
```

```
Fitting 10 folds for each of 5 candidates, totalling 50 fits
[CV] max_depth=14, min_samples_leaf=51, min_samples_split=125, n_estimators=117

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV]  max_depth=14, min_samples_leaf=51, min_samples_split=125, n_estimators=117, total=  19.8s
[CV] max_depth=14, min_samples_leaf=51, min_samples_split=125, n_estimators=117

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:   19.8s remaining:    0.0s

[CV]  max_depth=14, min_samples_leaf=51, min_samples_split=125, n_estimators=117, total=  18.7s
[CV] max_depth=14, min_samples_leaf=51, min_samples_split=125, n_estimators=117
[CV]  max_depth=14, min_samples_leaf=51, min_samples_split=125, n_estimators=117, total=  18.9s
[CV] max_depth=14, min_samples_leaf=51, min_samples_split=125, n_estimators=117
[CV]  max_depth=14, min_samples_leaf=51, min_samples_split=125, n_estimators=117, total=  18.8s
[CV] max_depth=14, min_samples_leaf=51, min_samples_split=125, n_estimators=117
[CV]  max_depth=14, min_samples_leaf=51, min_samples_split=125, n_estimators=117, total=  18.7s
[CV] max_depth=14, min_samples_leaf=51, min_samples_split=125, n_estimators=117
[CV]  max_depth=14, min_samples_leaf=51, min_samples_split=125, n_estimators=117, total=  19.3s
[CV] max_depth=14, min_samples_leaf=51, min_samples_split=125, n_estimators=117
[CV]  max_depth=14, min_samples_leaf=51, min_samples_split=125, n_estimators=117, total=  18.7s
[CV] max_depth=14, min_samples_leaf=51, min_samples_split=125, n_estimators=117
[CV]  max_depth=14, min_samples_leaf=51, min_samples_split=125, n_estimators=117, total=  18.4s
[CV] max_depth=14, min_samples_leaf=51, min_samples_split=125, n_estimators=117
[CV]  max_depth=14, min_samples_leaf=51, min_samples_split=125, n_estimators=117, total=  18.8s
[CV] max_depth=14, min_samples_leaf=51, min_samples_split=125, n_estimators=117
[CV]  max_depth=14, min_samples_leaf=51, min_samples_split=125, n_estimators=117, total=  19.1s
[CV] max_depth=12, min_samples_leaf=33, min_samples_split=138, n_estimators=109
[CV]  max_depth=12, min_samples_leaf=33, min_samples_split=138, n_estimators=109, total=  17.7s
[CV] max_depth=12, min_samples_leaf=33, min_samples_split=138, n_estimators=109
[CV]  max_depth=12, min_samples_leaf=33, min_samples_split=138, n_estimators=109, total=  17.6s
[CV] max_depth=12, min_samples_leaf=33, min_samples_split=138, n_estimators=109
[CV]  max_depth=12, min_samples_leaf=33, min_samples_split=138, n_estimators=109, total=  17.4s
[CV] max_depth=12, min_samples_leaf=33, min_samples_split=138, n_estimators=109
[CV]  max_depth=12, min_samples_leaf=33, min_samples_split=138, n_estimators=109, total=  17.5s
[CV] max_depth=12, min_samples_leaf=33, min_samples_split=138, n_estimators=109
[CV]  max_depth=12, min_samples_leaf=33, min_samples_split=138, n_estimators=109, total=  17.4s
[CV] max_depth=12, min_samples_leaf=33, min_samples_split=138, n_estimators=109
[CV]  max_depth=12, min_samples_leaf=33, min_samples_split=138, n_estimators=109, total=  17.1s
[CV] max_depth=12, min_samples_leaf=33, min_samples_split=138, n_estimators=109
[CV]  max_depth=12, min_samples_leaf=33, min_samples_split=138, n_estimators=109, total=  17.7s
[CV] max_depth=12, min_samples_leaf=33, min_samples_split=138, n_estimators=109
[CV]  max_depth=12, min_samples_leaf=33, min_samples_split=138, n_estimators=109, total=  17.6s
[CV] max_depth=12, min_samples_leaf=33, min_samples_split=138, n_estimators=109
[CV]  max_depth=12, min_samples_leaf=33, min_samples_split=138, n_estimators=109, total=  17.4s
[CV] max_depth=12, min_samples_leaf=33, min_samples_split=138, n_estimators=109
[CV]  max_depth=12, min_samples_leaf=33, min_samples_split=138, n_estimators=109, total=  17.7s
[CV] max_depth=11, min_samples_leaf=56, min_samples_split=179, n_estimators=106
[CV]  max_depth=11, min_samples_leaf=56, min_samples_split=179, n_estimators=106, total=  16.6s
[CV] max_depth=11, min_samples_leaf=56, min_samples_split=179, n_estimators=106
[CV]  max_depth=11, min_samples_leaf=56, min_samples_split=179, n_estimators=106, total=  16.3s
[CV] max_depth=11, min_samples_leaf=56, min_samples_split=179, n_estimators=106
[CV]  max_depth=11, min_samples_leaf=56, min_samples_split=179, n_estimators=106, total=  16.5s
[CV] max_depth=11, min_samples_leaf=56, min_samples_split=179, n_estimators=106
[CV]  max_depth=11, min_samples_leaf=56, min_samples_split=179, n_estimators=106, total=  16.2s
[CV] max_depth=11, min_samples_leaf=56, min_samples_split=179, n_estimators=106
[CV]  max_depth=11, min_samples_leaf=56, min_samples_split=179, n_estimators=106, total=  16.4s
[CV] max_depth=11, min_samples_leaf=56, min_samples_split=179, n_estimators=106
[CV]  max_depth=11, min_samples_leaf=56, min_samples_split=179, n_estimators=106, total=  16.2s
[CV] max_depth=11, min_samples_leaf=56, min_samples_split=179, n_estimators=106
[CV]  max_depth=11, min_samples_leaf=56, min_samples_split=179, n_estimators=106, total=  16.3s
[CV] max_depth=11, min_samples_leaf=56, min_samples_split=179, n_estimators=106
[CV]  max_depth=11, min_samples_leaf=56, min_samples_split=179, n_estimators=106, total=  16.1s
[CV] max_depth=11, min_samples_leaf=56, min_samples_split=179, n_estimators=106
[CV]  max_depth=11, min_samples_leaf=56, min_samples_split=179, n_estimators=106, total=  16.3s
[CV] max_depth=11, min_samples_leaf=56, min_samples_split=179, n_estimators=106
[CV]  max_depth=11, min_samples_leaf=56, min_samples_split=179, n_estimators=106, total=  16.4s
[CV] max_depth=13, min_samples_leaf=49, min_samples_split=165, n_estimators=108
[CV]  max_depth=13, min_samples_leaf=49, min_samples_split=165, n_estimators=108, total=  17.4s
[CV] max_depth=13, min_samples_leaf=49, min_samples_split=165, n_estimators=108
[CV]  max_depth=13, min_samples_leaf=49, min_samples_split=165, n_estimators=108, total=  17.1s
```

```
[CV] max_depth=13, min_samples_leaf=49, min_samples_split=165, n_estimators=108
[CV]  max_depth=13, min_samples_leaf=49, min_samples_split=165, n_estimators=108, total=  17.3s
[CV] max_depth=13, min_samples_leaf=49, min_samples_split=165, n_estimators=108
[CV]  max_depth=13, min_samples_leaf=49, min_samples_split=165, n_estimators=108, total=  17.4s
[CV] max_depth=13, min_samples_leaf=49, min_samples_split=165, n_estimators=108
[CV]  max_depth=13, min_samples_leaf=49, min_samples_split=165, n_estimators=108, total=  17.4s
[CV] max_depth=13, min_samples_leaf=49, min_samples_split=165, n_estimators=108
[CV]  max_depth=13, min_samples_leaf=49, min_samples_split=165, n_estimators=108, total=  17.5s
[CV] max_depth=13, min_samples_leaf=49, min_samples_split=165, n_estimators=108
[CV]  max_depth=13, min_samples_leaf=49, min_samples_split=165, n_estimators=108, total=  17.4s
[CV] max_depth=13, min_samples_leaf=49, min_samples_split=165, n_estimators=108
[CV]  max_depth=13, min_samples_leaf=49, min_samples_split=165, n_estimators=108, total=  17.0s
[CV] max_depth=13, min_samples_leaf=49, min_samples_split=165, n_estimators=108
[CV]  max_depth=13, min_samples_leaf=49, min_samples_split=165, n_estimators=108, total=  17.3s
[CV] max_depth=13, min_samples_leaf=49, min_samples_split=165, n_estimators=108
[CV]  max_depth=13, min_samples_leaf=49, min_samples_split=165, n_estimators=108, total=  17.1s
[CV] max_depth=14, min_samples_leaf=28, min_samples_split=111, n_estimators=121
[CV]  max_depth=14, min_samples_leaf=28, min_samples_split=111, n_estimators=121, total=  20.2s
[CV] max_depth=14, min_samples_leaf=28, min_samples_split=111, n_estimators=121
[CV]  max_depth=14, min_samples_leaf=28, min_samples_split=111, n_estimators=121, total=  19.9s
[CV] max_depth=14, min_samples_leaf=28, min_samples_split=111, n_estimators=121
[CV]  max_depth=14, min_samples_leaf=28, min_samples_split=111, n_estimators=121, total=  20.4s
[CV] max_depth=14, min_samples_leaf=28, min_samples_split=111, n_estimators=121
[CV]  max_depth=14, min_samples_leaf=28, min_samples_split=111, n_estimators=121, total=  20.3s
[CV] max_depth=14, min_samples_leaf=28, min_samples_split=111, n_estimators=121
[CV]  max_depth=14, min_samples_leaf=28, min_samples_split=111, n_estimators=121, total=  20.3s
[CV] max_depth=14, min_samples_leaf=28, min_samples_split=111, n_estimators=121
[CV]  max_depth=14, min_samples_leaf=28, min_samples_split=111, n_estimators=121, total=  20.1s
[CV] max_depth=14, min_samples_leaf=28, min_samples_split=111, n_estimators=121
[CV]  max_depth=14, min_samples_leaf=28, min_samples_split=111, n_estimators=121, total=  20.1s
[CV] max_depth=14, min_samples_leaf=28, min_samples_split=111, n_estimators=121
[CV]  max_depth=14, min_samples_leaf=28, min_samples_split=111, n_estimators=121, total=  20.2s
[CV] max_depth=14, min_samples_leaf=28, min_samples_split=111, n_estimators=121
[CV]  max_depth=14, min_samples_leaf=28, min_samples_split=111, n_estimators=121, total=  20.4s
[CV] max_depth=14, min_samples_leaf=28, min_samples_split=111, n_estimators=121
[CV]  max_depth=14, min_samples_leaf=28, min_samples_split=111, n_estimators=121, total=  20.3s

[Parallel(n_jobs=1)]: Done  50 out of  50 | elapsed: 15.0min finished
```

Out[11]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=RandomForestClassifier(bootstrap=True,
                                                    ccp_alpha=0.0,
                                                    class_weight=None,
                                                    criterion='gini',
                                                    max_depth=None,
                                                    max_features='auto',
                                                    max_leaf_nodes=None,
                                                    max_samples=None,
                                                    min_impurity_decrease=0.0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
                                                    min_weight_fraction_leaf=0.0,
                                                    n_estimators=100, n_job...
                                        'min_samples_leaf':
<scipy.stats._distn_infrastructure.rv_frozen object at 0x7f5adaebfcf8>,
                                        'min_samples_split':
<scipy.stats._distn_infrastructure.rv_frozen object at 0x7f5adaebf6a0>,
                                        'n_estimators': <scipy.stats._distn_infrastructure.rv_froze
object at 0x7f5adc83e128>},
                   pre_dispatch='2*n_jobs', random_state=25, refit=True,
                   return_train_score=False, scoring='f1', verbose=2)
```

In [14]:

```
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 [0.96339318 0.96253764 0.96059567 0.96262681 0.96466444]
```

In [15]:

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

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=14, max_features='auto',
                       max_leaf_nodes=None, max_samples=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 [0]:

```
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 [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 [18]:

```
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 0.9667410261610221
Test f1 score 0.927516041877744
```

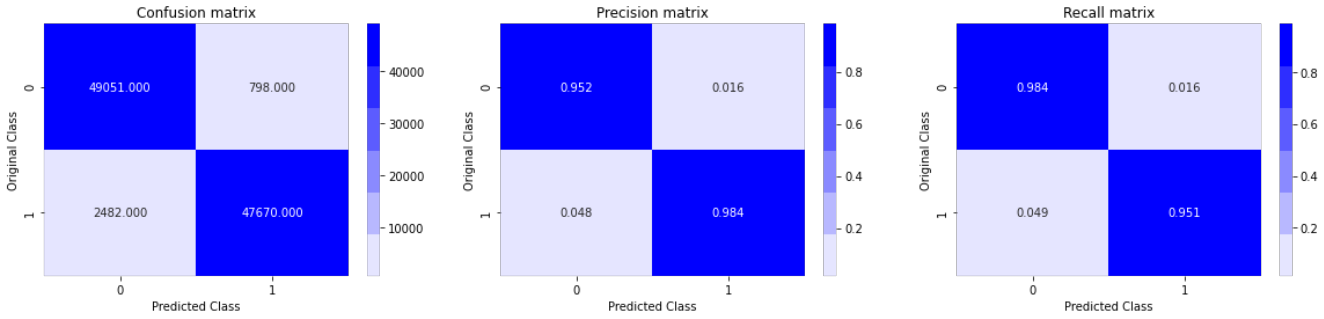
In [0]:

```
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()
```

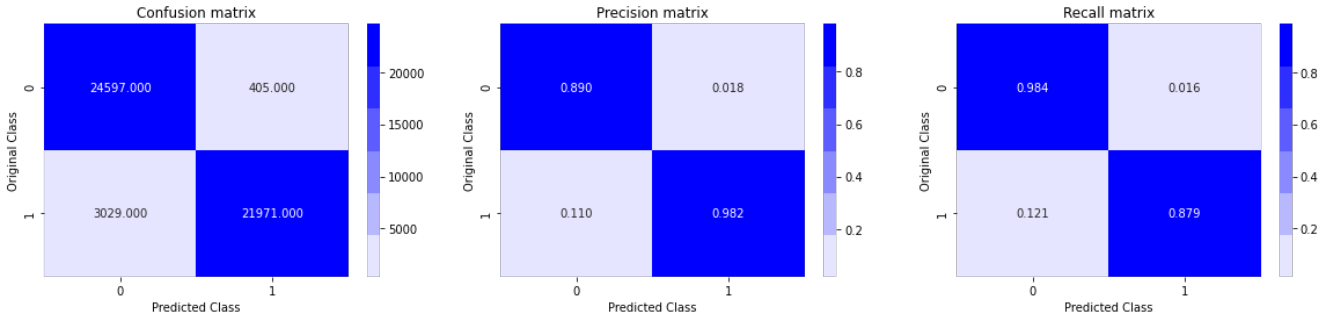
In [20]:

```
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
```

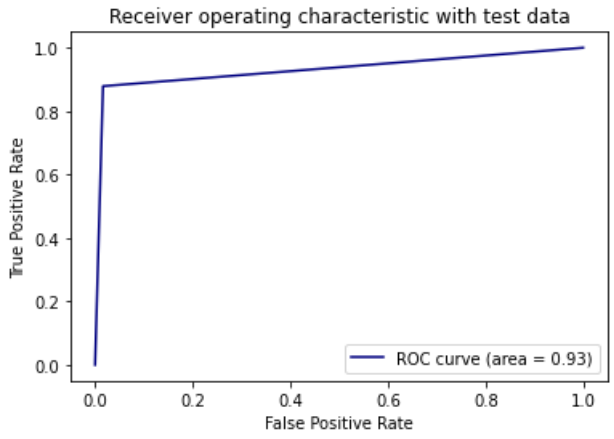



```
Test confusion_matrix
```



In [21]:

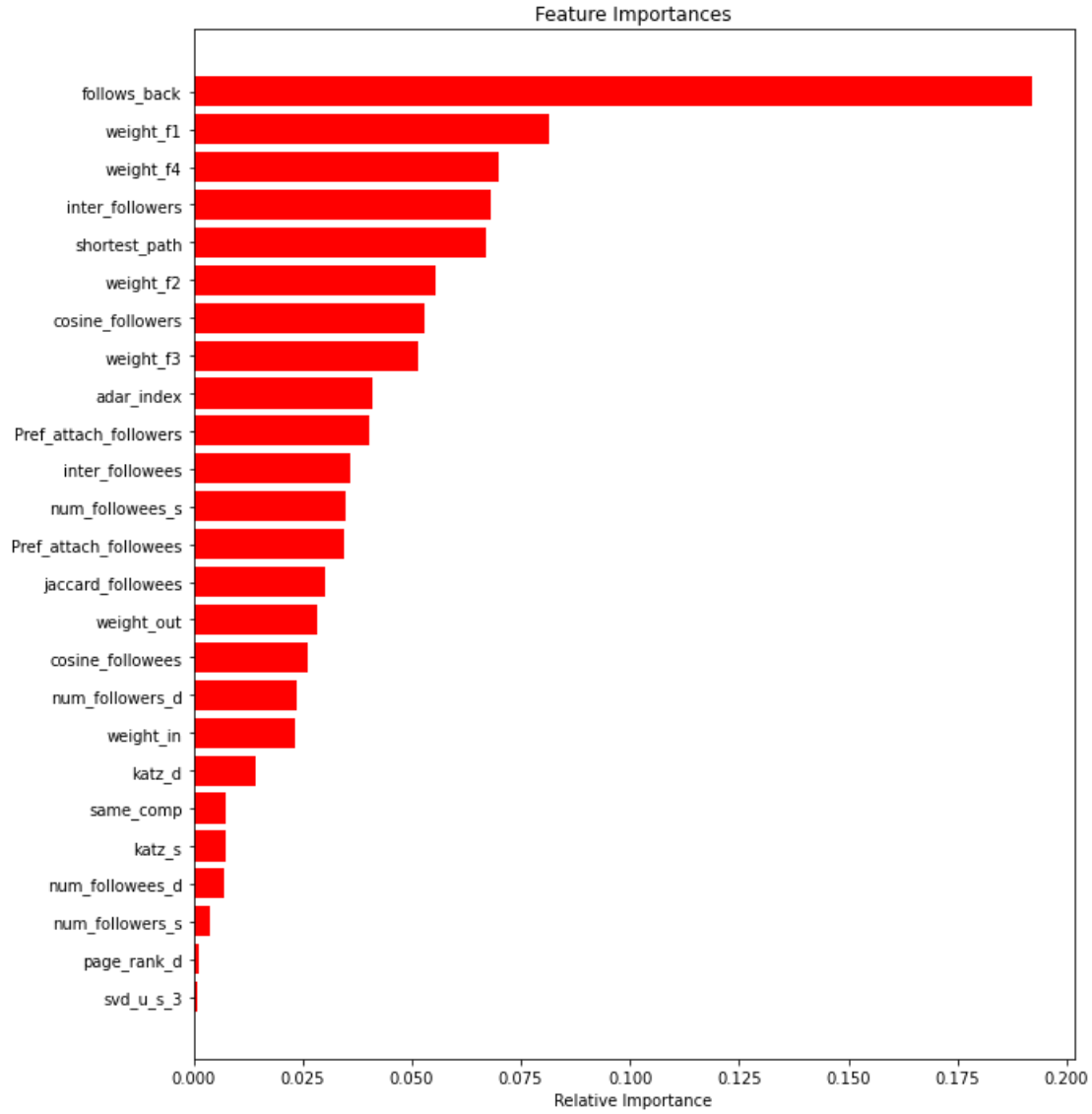
```
from sklearn.metrics import roc_curve, auc
fpr,tpr,ths = roc_curve(y_test,y_test_pred)
auc_sc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='navy',label='ROC curve (area = %0.2f)' % auc_sc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic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



In [22]:

```
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 [24]:

```
import xgboost as xgb
from sklearn.model_selection import GridSearchCV

param={
'learning_rate':[0.001,0.05,0.5,1],
'n_estimators':[25,50,100,200,500]
}
x_model = xgb.XGBClassifier(n_jobs=-1)
clf = GridSearchCV(x_model,param, cv=2,verbose=2,return_train_score=True)
clf.fit(df_final_train,y_train)
```

```
Fitting 2 folds for each of 20 candidates, totalling 40 fits
[CV] learning_rate=0.001, n_estimators=25 .............................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV] ............. learning_rate=0.001, n_estimators=25, total=   3.1s
[CV] learning_rate=0.001, n_estimators=25 .............................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    3.1s remaining:    0.0s

[CV] ............. learning_rate=0.001, n_estimators=25, total=   3.0s
[CV] learning_rate=0.001, n_estimators=50 .............................
[CV] ............. learning_rate=0.001, n_estimators=50, total=   5.5s
[CV] learning_rate=0.001, n_estimators=50 .............................
[CV] ............. learning_rate=0.001, n_estimators=50, total=   5.6s
[CV] learning_rate=0.001, n_estimators=100 ............................
[CV] ............. learning_rate=0.001, n_estimators=100, total=  10.8s
[CV] learning_rate=0.001, n_estimators=100 ............................
[CV] ............. learning_rate=0.001, n_estimators=100, total=  10.9s
[CV] learning_rate=0.001, n_estimators=200 ............................
[CV] ............. learning_rate=0.001, n_estimators=200, total=  21.7s
[CV] learning_rate=0.001, n_estimators=200 ............................
[CV] ............. learning_rate=0.001, n_estimators=200, total=  21.3s
[CV] learning_rate=0.001, n_estimators=500 ............................
[CV] ............. learning_rate=0.001, n_estimators=500, total=  53.3s
[CV] learning_rate=0.001, n_estimators=500 ............................
[CV] ............. learning_rate=0.001, n_estimators=500, total=  53.0s
[CV] learning_rate=0.05, n_estimators=25 ..............................
[CV] ............... learning_rate=0.05, n_estimators=25, total=   3.0s
[CV] learning_rate=0.05, n_estimators=25 ..............................
[CV] ............... learning_rate=0.05, n_estimators=25, total=   3.0s
[CV] learning_rate=0.05, n_estimators=50 ..............................
[CV] ............... learning_rate=0.05, n_estimators=50, total=   5.8s
[CV] learning_rate=0.05, n_estimators=50 ..............................
[CV] ............... learning_rate=0.05, n_estimators=50, total=   5.8s
[CV] learning_rate=0.05, n_estimators=100 .............................
[CV] ............. learning_rate=0.05, n_estimators=100, total=  11.5s
[CV] learning_rate=0.05, n_estimators=100 .............................
[CV] ............. learning_rate=0.05, n_estimators=100, total=  11.2s
[CV] learning_rate=0.05, n_estimators=200 .............................
[CV] ............. learning_rate=0.05, n_estimators=200, total=  22.2s
[CV] learning_rate=0.05, n_estimators=200 .............................
[CV] ............. learning_rate=0.05, n_estimators=200, total=  22.4s
[CV] learning_rate=0.05, n_estimators=500 .............................
[CV] ............. learning_rate=0.05, n_estimators=500, total=  54.2s
[CV] learning_rate=0.05, n_estimators=500 .............................
[CV] ............. learning_rate=0.05, n_estimators=500, total=  54.4s
[CV] learning_rate=0.5, n_estimators=25 ...............................
[CV] ............... learning_rate=0.5, n_estimators=25, total=   3.0s
[CV] learning_rate=0.5, n_estimators=25 ...............................
[CV] ............... learning_rate=0.5, n_estimators=25, total=   3.0s
[CV] learning_rate=0.5, n_estimators=50 ...............................
[CV] ............... learning_rate=0.5, n_estimators=50, total=   5.7s
[CV] learning_rate=0.5, n_estimators=50 ...............................
[CV] ............... learning_rate=0.5, n_estimators=50, total=   5.7s
[CV] learning_rate=0.5, n_estimators=100 ..............................
[CV] ............. learning_rate=0.5, n_estimators=100, total=  11.2s
[CV] learning_rate=0.5, n_estimators=100 ..............................
[CV] ............. learning_rate=0.5, n_estimators=100, total=  11.2s
[CV] learning_rate=0.5, n_estimators=200 ..............................
[CV] ............. learning_rate=0.5, n_estimators=200, total=  21.8s
[CV] learning_rate=0.5, n_estimators=200 ..............................
[CV] ............. learning_rate=0.5, n_estimators=200, total=  21.8s
[CV] learning_rate=0.5, n_estimators=500 ..............................
[CV] ............. learning_rate=0.5, n_estimators=500, total=  54.7s
[CV] learning_rate=0.5, n_estimators=500 ..............................
[CV] ............. learning_rate=0.5, n_estimators=500, total=  53.9s
[CV] learning_rate=1, n_estimators=25 .................................
[CV] ................. learning_rate=1, n_estimators=25, total=   3.0s
[CV] learning_rate=1, n_estimators=25 .................................
[CV] ................. learning_rate=1, n_estimators=25, total=   3.0s
[CV] learning_rate=1, n_estimators=50 .................................
[CV] ................. learning_rate=1, n_estimators=50, total=   5.7s
[CV] learning_rate=1, n_estimators=50 .................................
[CV] ................. learning_rate=1, n_estimators=50, total=   5.9s
```

```
[CV] learning_rate=1, n_estimators=100 ..............................
[CV] ................ learning_rate=1, n_estimators=100, total=  11.1s
[CV] learning_rate=1, n_estimators=100 ..............................
[CV] ................ learning_rate=1, n_estimators=100, total=  11.1s
[CV] learning_rate=1, n_estimators=200 ..............................
[CV] ................ learning_rate=1, n_estimators=200, total=  22.1s
[CV] learning_rate=1, n_estimators=200 ..............................
[CV] ................ learning_rate=1, n_estimators=200, total=  22.0s
[CV] learning_rate=1, n_estimators=500 ..............................
[CV] ................ learning_rate=1, n_estimators=500, total=  53.2s
[CV] learning_rate=1, n_estimators=500 ..............................
[CV] ................ learning_rate=1, n_estimators=500, total=  53.8s

[Parallel(n_jobs=1)]: Done  40 out of  40 | elapsed: 12.9min finished
```

Out[24]:

```
GridSearchCV(cv=2, error_score=nan,
             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=3, min_child_weight=1,
                                     missing=None, n_estimators=100, n_jobs=-1,
                                     nthread=None, objective='binary:logistic',
                                     random_state=0, reg_alpha=0, reg_lambda=1,
                                     scale_pos_weight=1, seed=None, silent=None,
                                     subsample=1, verbosity=1),
             iid='deprecated', n_jobs=None,
             param_grid={'learning_rate': [0.001, 0.05, 0.5, 1],
                         'n_estimators': [25, 50, 100, 200, 500]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=2)
```

In [26]:

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

```
mean test scores [0.89988099 0.90073099 0.90073099 0.90086099 0.9100108  0.9280007
 0.93841065 0.96538036 0.97393026 0.97692023 0.97466025 0.97676023
 0.97891021 0.98168019 0.98266017 0.97544024 0.97709023 0.9792602
 0.98050019 0.98068019]
mean train scores [0.900191   0.90085099 0.90085099 0.90126099 0.91057099 0.92851074
 0.93867058 0.96551034 0.97462026 0.9793602  0.97538025 0.97944021
 0.98705013 0.99644004 1.         0.97928021 0.98605014 0.99480006
 0.99997    1.        ]
```

In [27]:

```
print(clf.best_estimator_)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.5, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=500, n_jobs=-1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

In [0]:

```
x_model=xgb.XGBClassifier(n_estimators=500,learning_rate=0.5).fit(df_final_train,y_train)
```

In [0]:

```
y_train_pred = clf.predict(df_final_train)
y_test_pred = clf.predict(df_final_test)
```

In [30]:

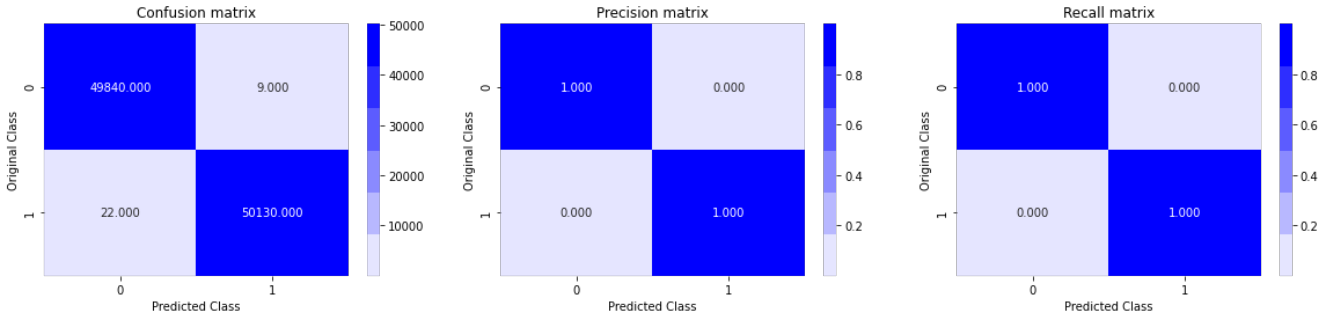
```
from sklearn.metrics import f1_score
print('Train f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))
```

```
Train f1 score 0.9996908994825059
Test f1 score 0.8712367869408144
```

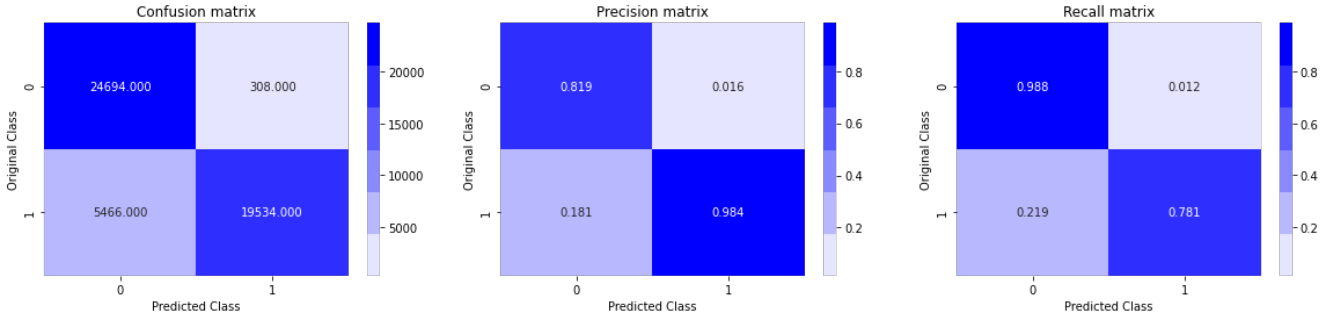
In [31]:

```
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
```

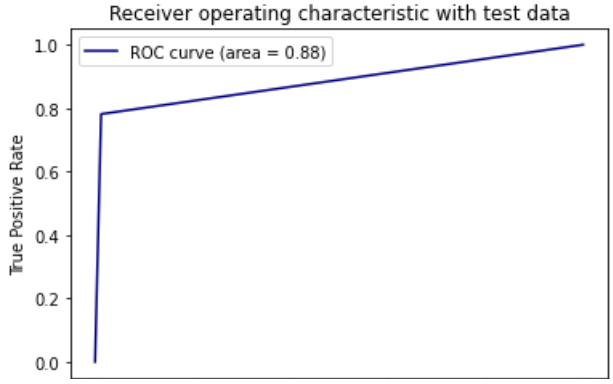


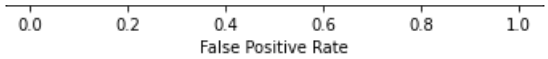

```
Test confusion_matrix
```



In [32]:

```
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 [34]:

```
features = df_final_train.columns
importances = x_model.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()
```