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
from xgboost import XGBClassifier
```

In [2]:

```
# from google.colab import drive
# drive.mount('/content/drive')
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

In [3]:

```
# cd drive/My Drive/DONOR_CHOOSE_KNN/facebook
```

/content/drive/My Drive/DONOR_CHOOSE_KNN/facebook

In [0]:

```
#reading
from pandas import read_hdf
df_final_train = read_hdf('data/fea_sample/storage_sample_stage4.h5', 'train_df',mode='r')
df_final_test = read_hdf('data/fea_sample/storage_sample_stage4.h5', 'test_df',mode='r')
```

In [23]:

```
df_final_train.shape,df_final_test.shape
```

Out[23]:

((100002, 54), (50002, 54))

((1505602, 54), (1505602, 54))

In [24]:

```
df_final_train.head(5)
```

Out[24]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	num_followe
0	273084	1505602	1	0	0.000000	0.000000	0.000000	
1	832016	1543415	1	0	0.187135	0.028382	0.343828	
2	1325247	760242	1	0	0.369565	0.156957	0.566038	
3	1368400	1006992	1	0	0.000000	0.000000	0.000000	
4	140165	1708748	1	0	0.000000	0.000000	0.000000	

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)
```

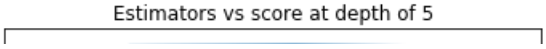
In [10]:

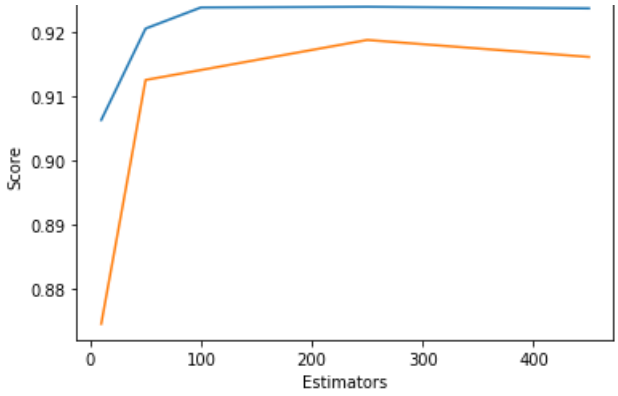
```
estimators = [10,50,100,250,450]
train_scores = []
test_scores = []
for i in estimators:
    clf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=5, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            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 0.9063252121775113 test Score 0.8745605278006858
Estimators =  50 Train Score 0.9205725512208812 test Score 0.9125653355634538
Estimators =  100 Train Score 0.9238690848446947 test Score 0.9141199714153599
Estimators =  250 Train Score 0.9239789348046863 test Score 0.9188007232664732
Estimators =  450 Train Score 0.9237190618658074 test Score 0.9161507685828595
```

Out[10]:

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

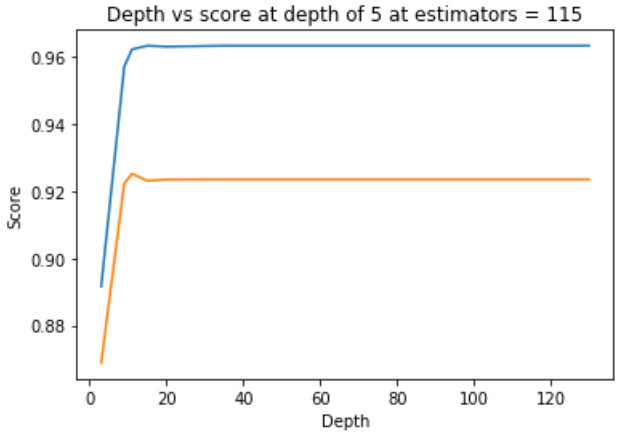




In [11]:

```
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 0.8916120853581238 test Score 0.8687934859875491
depth =  9 Train Score 0.9572226298198419 test Score 0.9222953031452904
depth =  11 Train Score 0.9623451340902863 test Score 0.9252318758281279
depth =  15 Train Score 0.9634267621927706 test Score 0.9231288356496615
depth =  20 Train Score 0.9631629153051491 test Score 0.9235051024711141
depth =  35 Train Score 0.9634333127085721 test Score 0.9235601652753184
depth =  50 Train Score 0.9634333127085721 test Score 0.9235601652753184
depth =  70 Train Score 0.9634333127085721 test Score 0.9235601652753184
depth =  130 Train Score 0.9634333127085721 test Score 0.9235601652753184
```



In [12]:

```
from sklearn.metrics import f1_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint as sp_randint
```

```
from scipy.stats import uniform

param_dist = {"n_estimators":sp_randint(105,125),
              "max_depth": sp_randint(10,15),
              "min_samples_split": sp_randint(110,190),
              "min_samples_leaf": sp_randint(25,65)}

clf = RandomForestClassifier(random_state=25,n_jobs=-1)

rf_random = RandomizedSearchCV(clf, param_distributions=param_dist,
                               n_iter=5,cv=10,scoring='f1',return_train_score=True,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 [0.96225043 0.96215493 0.96057081 0.96194015 0.96330005]
mean train scores [0.96294922 0.96266735 0.96115674 0.96263457 0.96430539]
```

In [13]:

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

```
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 = 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 [16]:

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

```
Train f1 score 0.9652533106548414
Test f1 score 0.9241678239279553
```

In [0]:

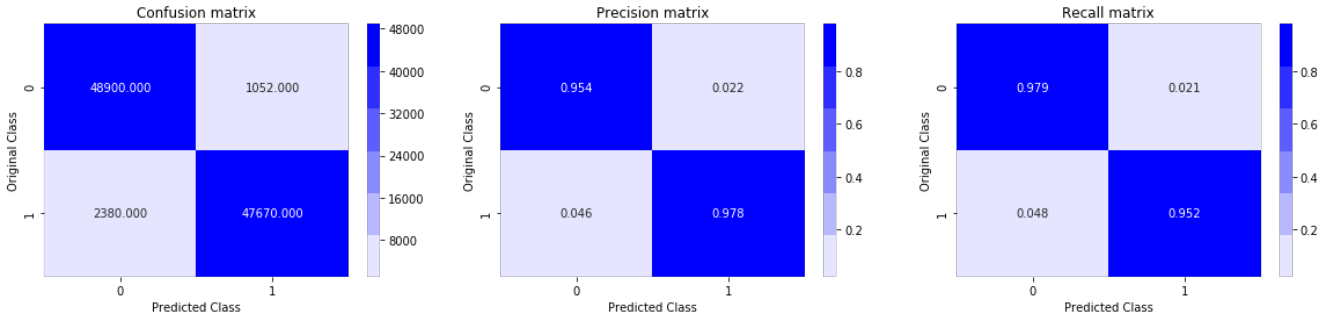
```
from sklearn.metrics import confusion_matrix
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)

    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()
```

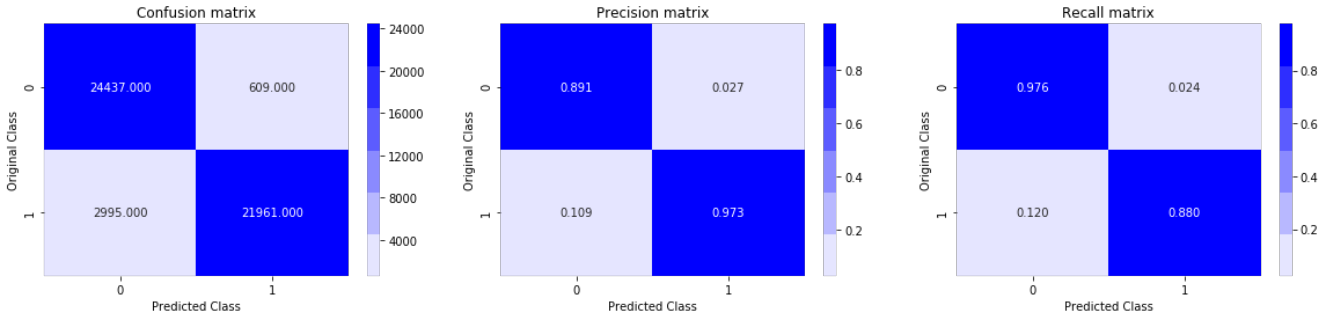
In [18]:

```
print('Train confusion_matrix')
plot_confusion_matrix(y_train,y_train_pred)
print('Test confusion_matrix')
plot_confusion_matrix(y_test,y_test_pred)
```

Train confusion_matrix

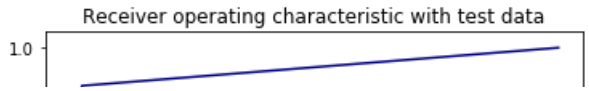


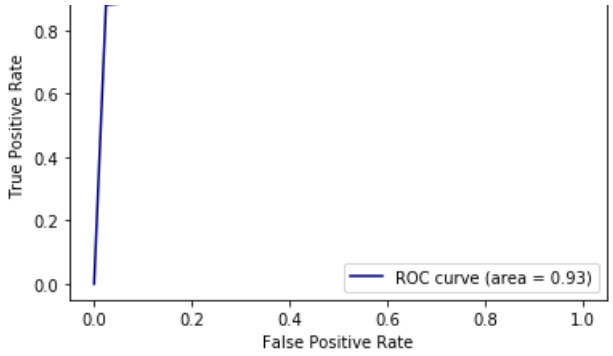
Test confusion_matrix



In [19]:

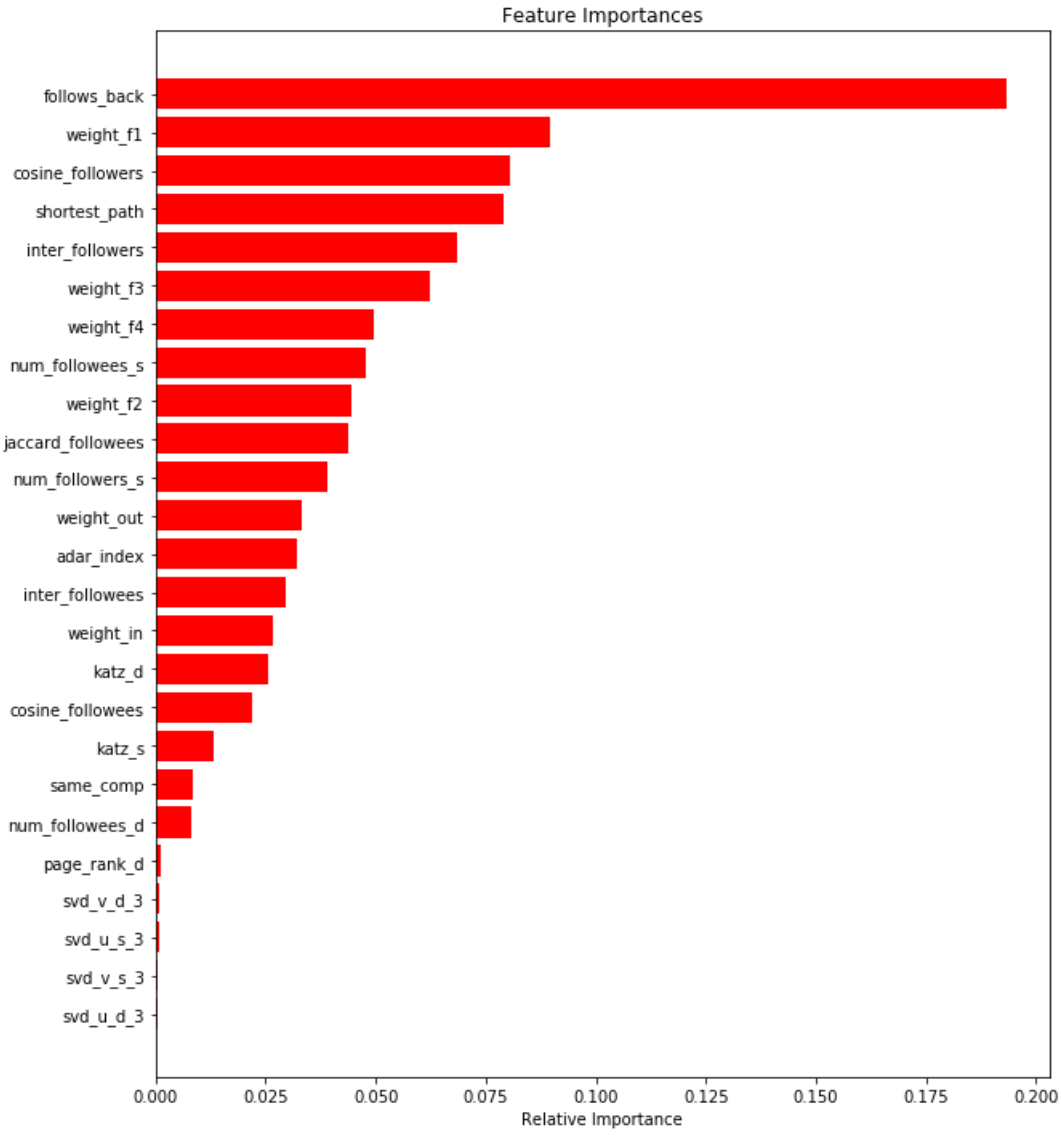
```
from sklearn.metrics import roc_curve, auc
fpr,tpr,ths = roc_curve(y_test,y_test_pred)
auc_sc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='navy',label='ROC curve (area = %0.2f)' % auc_sc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic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





In [20]:

```
features = df_final_train.columns
importances = clf.feature_importances_
indices = (np.argsort(importances))[-25:]
plt.figure(figsize=(10,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='r', 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/

Attachment in below link http://snap.stanford.edu/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.

Task-1 Add another feature called Preferential Attachment with followers and followees data of vertex.

In [0]:

```
if os.path.isfile('../input/facebook-data/data/after_eda/train_pos_after_eda.csv'):
    train_graph=nx.read_edgelist('../input/facebook-data/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
```

In [0]:

```
def prefrential_attachment_followee(a,b):
  try:
      if len(set(train_graph.successors(a))) == 0  | len(set(train_graph.successors(b))) == 0:
          return 0
      preattachfollowee = len(set(train_graph.successors(a)))* len(set(train_graph.successors(b)))
  except:
#      print("entred")
      return 0
  return preattachfollowee
```

In [0]:

```
def prefrential_attachment_follower(a,b):
  try:
      if len(set(train_graph.predecessors(a))) == 0  | len(set(train_graph.predecessors(b))) == 0:
          return 0
      preattachfollower  = len(set(train_graph.predecessors(a)))* len(set(train_graph.predecessors(b)))
      return preattachfollower
  except:
#      print("entred")
      return 0
```

In [0]:

```
#mapping  Preferential Attachment followers to train and test data
df_final_train['prefrential_attachment_follower'] = df_final_train.apply(lambda row:
                                    prefrential_attachment_follower(row['source_node'],row['destination_node']),axis=1)
df_final_test['prefrential_attachment_follower'] = df_final_test.apply(lambda row:
                                    prefrential_attachment_follower(row['source_node'],row['destination_node']),axis=1)

#mapping  Preferential Attachment followees to train and test data
df_final_train['prefrential_attachment_followee'] = df_final_train.apply(lambda row:
                                    prefrential_attachment_followee(row['source_node'],row['destination_node']),axis=1)
df_final_test['prefrential_attachment_followee'] = df_final_test.apply(lambda row:
                                    prefrential_attachment_followee(row['source_node'],row['destination_node']),axis=1)


# #mapping jaccrd followers to train and test data
# df_final_train['cosine_followers'] = df_final_train.apply(lambda row:
#                                     cosine_for_followers(row['source_node'],row['destination_node']),axis=1)
# df_final_test['cosine_followers'] = df_final_test.apply(lambda row:
#                                     cosine_for_followers(row['source_node'],row['destination_node']),axis=1)

# #mapping jaccrd followees to train and test data
```

```
# df_final_train['cosine_followees'] = df_final_train.apply(lambda row:
#                         cosine_for_followees(row['source_node'],row['destination_node']),axis=1)
# df_final_test['cosine_followees'] = df_final_test.apply(lambda row:
#                         cosine_for_followees(row['source_node'],row['destination_node']),axis=1)
```

In [0]:

```
df_final_train.head(10)
```

Out[0]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	num_followe
0	273084	1505602	1	0	0.000000	0.000000	0.000000	
1	832016	1543415	1	0	0.187135	0.028382	0.343828	
2	1325247	760242	1	0	0.369565	0.156957	0.566038	
3	1368400	1006992	1	0	0.000000	0.000000	0.000000	
4	140165	1708748	1	0	0.000000	0.000000	0.000000	
5	1377733	375423	1	0	0.125000	0.148148	0.223607	
6	1691962	1039906	1	0	0.000000	0.000000	0.000000	
7	628080	812266	1	0	0.000000	0.117851	0.000000	
8	1725153	1822102	1	0	0.000000	0.000000	0.000000	
9	654494	1487831	1	0	0.111111	0.188982	0.218218	

10 rows × 56 columns

Task-2 Add feature called svd_dot. you can calculate svd_dot as Dot product between sourse node svd and destination node svd features.

In [0]:

```
df_final_train['svd_dot_u'] = 'null'
# df_final_train.shape[0]
from tqdm import tqdm
for i in tqdm(range(0,df_final_train.shape[0])):
    source_svd = df_final_train[['svd_u_s_1', 'svd_u_s_2','svd_u_s_3', 'svd_u_s_4', 'svd_u_s_5', 'svd_u_s_6']].iloc[i:i+1]
    destination_svd = df_final_train[['svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3', 'svd_u_d_4', 'svd_u_d_5','svd_u_d_6']].iloc[i:i+1]
    svd_dot_train = np.dot(source_svd,destination_svd.T)
    df_final_train['svd_dot_u'].iloc[i:i+1] = svd_dot_train
```

```
100%|██████████| 100002/100002 [11:37<00:00, 143.44it/s]
```

In [0]:

```
df_final_train['svd_dot_v'] = 'null'
# df_final_train.shape[0]
from tqdm import tqdm
for i in tqdm(range(0,df_final_train.shape[0])):
    source_svd = df_final_train[['svd_v_s_1','svd_v_s_2', 'svd_v_s_3', 'svd_v_s_4', 'svd_v_s_5', 'svd_v_s_6']].iloc[i:i+1]
    destination_svd = df_final_train[['svd_v_d_1', 'svd_v_d_2', 'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5','svd_v_d_6']].iloc[i:i+1]
    svd_dot_train_v = np.dot(source_svd,destination_svd.T)
    df_final_train['svd_dot_v'].iloc[i:i+1] = svd_dot_train_v
```

```
100%|██████████| 100002/100002 [11:37<00:00, 143.43it/s]
```

In [0]:

```
df_final_test['svd_dot_u'] = 'null'
# df_final_train.shape[0]
from tqdm import tqdm
for i in tqdm(range(0,df_final_test.shape[0])):
    source_svd = df_final_test[['svd_u_s_1', 'svd_u_s_2','svd_u_s_3', 'svd_u_s_4', 'svd_u_s_5', 'svd_u_s_6']].iloc[i:i+1]
    destination_svd = df_final_test[['svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3', 'svd_u_d_4', 'svd_u_d_5','svd_u_d_6']].iloc[i:i+1]
    svd_dot_test = np.dot(source_svd,destination_svd.T)
    df_final_test['svd_dot_u'].iloc[i:i+1] = svd_dot_test
```

100%|██████████| 50002/50002 [03:57<00:00, 210.24it/s]

In [0]:

```
df_final_test['svd_dot_v'] = 'null'
# df_final_train.shape[0]
from tqdm import tqdm
for i in tqdm(range(0,df_final_test.shape[0])):
    source_svd = df_final_test[['svd_v_s_1','svd_v_s_2', 'svd_v_s_3', 'svd_v_s_4', 'svd_v_s_5', 'svd_v_s_6']].iloc[i:i+1]
    destination_svd = df_final_test[['svd_v_d_1', 'svd_v_d_2', 'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5','svd_v_d_6']].iloc[i:i+1]
    svd_dot_test_v = np.dot(source_svd,destination_svd.T)
    df_final_test['svd_dot_v'].iloc[i:i+1] = svd_dot_test_v
```

100%|██████████| 50002/50002 [03:58<00:00, 209.56it/s]

In [0]:

```
df_final_train.head(10)
```

Out[0]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	num_followe
0	273084	1505602	1	0	0.000000	0.000000	0.000000	
1	832016	1543415	1	0	0.187135	0.028382	0.343828	
2	1325247	760242	1	0	0.369565	0.156957	0.566038	
3	1368400	1006992	1	0	0.000000	0.000000	0.000000	
4	140165	1708748	1	0	0.000000	0.000000	0.000000	
5	1377733	375423	1	0	0.125000	0.148148	0.223607	
6	1691962	1039906	1	0	0.000000	0.000000	0.000000	
7	628080	812266	1	0	0.000000	0.117851	0.000000	
8	1725153	1822102	1	0	0.000000	0.000000	0.000000	
9	654494	1487831	1	0	0.111111	0.188982	0.218218	

10 rows × 58 columns

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

Splitting the data

In [0]:

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

```
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 [0]:

```
df_final_train.head(10)
```

Out[0]:

	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	num_followers_s	num_followees_s	num_followees_d	inter
0	0	0.000000	0.000000	0.000000	6	15	8	
1	0	0.187135	0.028382	0.343828	94	61	142	
2	0	0.369565	0.156957	0.566038	28	41	22	
3	0	0.000000	0.000000	0.000000	11	5	7	
4	0	0.000000	0.000000	0.000000	1	11	3	
5	0	0.125000	0.148148	0.223607	9	10	8	
6	0	0.000000	0.000000	0.000000	5	1	4	
7	0	0.000000	0.117851	0.000000	3	7	7	
8	0	0.000000	0.000000	0.000000	13	9	9	
9	0	0.111111	0.188982	0.218218	2	7	3	

10 rows × 55 columns

In [0]:

```
df_final_train = df_final_train.astype(float)
df_final_test = df_final_test.astype(float)
```

In [0]:

```
# df_final_train
```

Hyperarameter tunning

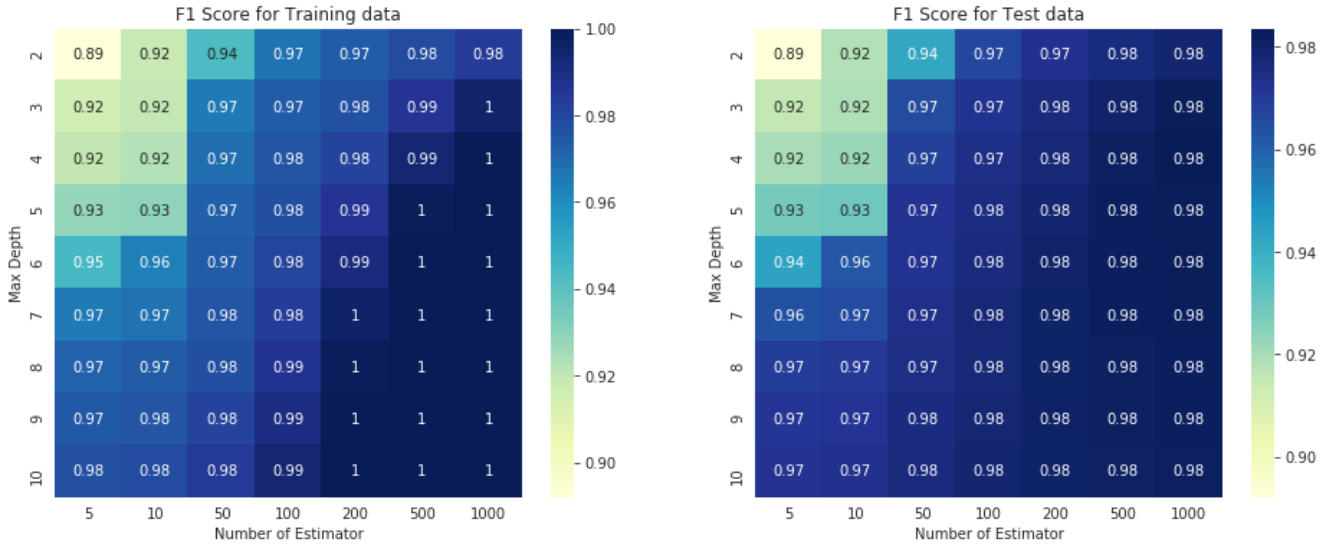
In [0]:

```
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV

xg_clf = XGBClassifier()
parameters = {'n_estimators':[5, 10, 50, 100, 200, 500, 1000],'max_depth':[2,3, 4, 5, 6, 7, 8, 9, 10]}
xgclf = GridSearchCV(xg_clf, parameters, n_jobs= -1, verbose=10, cv=2, scoring='f1',return_train_score=True)
xgclf.fit(df_final_train, y_train)

results = pd.DataFrame.from_dict(xgclf.cv_results_)
# results4 = results4.sort_values(['param_alpha'])
max_depth_list = list(xgclf.cv_results_['param_max_depth'].data)
n_estimator_list = list(xgclf.cv_results_['param_n_estimators'].data)


sns.set_style("whitegrid")
plt.figure(figsize=(16,6))
plt.subplot(1,2,1)

data = pd.DataFrame(data={'Number of Estimator':n_estimator_list, 'Max Depth':max_depth_list, 'F1 Score':xgclf.cv_results_['mean_train_score']})
```

```
data = data.reset_index().pivot_table(index='Max Depth', columns='Number of Estimator', values='F1 Score')
sns.heatmap(data, annot=True, cmap="YlGnBu").set_title('F1 Score for Training data')
plt.subplot(1,2,2)

# Testing Heatmap
data = pd.DataFrame(data={'Number of Estimator':n_estimator_list, 'Max Depth':max_depth_list, 'F1 Score':xgclf.cv_results_['mean_test_score']})
data = data.reset_index().pivot_table(index='Max Depth', columns='Number of Estimator', values='F1 Score')
sns.heatmap(data, annot=True, cmap="YlGnBu").set_title('F1 Score for Test data')
plt.show()

results.head()
```

Fitting 2 folds for each of 63 candidates, totalling 126 fits

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:    4.3s
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:    6.5s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:   58.5s
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:  4.5min
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed:  5.1min
[Parallel(n_jobs=-1)]: Done  28 tasks      | elapsed: 10.9min
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed: 12.5min
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed: 19.4min
[Parallel(n_jobs=-1)]: Done  57 tasks      | elapsed: 29.7min
[Parallel(n_jobs=-1)]: Done  68 tasks      | elapsed: 35.4min
[Parallel(n_jobs=-1)]: Done  81 tasks      | elapsed: 48.6min
[Parallel(n_jobs=-1)]: Done  94 tasks      | elapsed: 59.1min
[Parallel(n_jobs=-1)]: Done 109 tasks      | elapsed: 79.4min
[Parallel(n_jobs=-1)]: Done 126 out of 126 | elapsed: 105.4min finished
```



Out[0]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_n_estimators	params	split0_test_sco
0	1.661970	0.012343	0.120193	0.016703	2	5	{'max_depth': 2, 'n_estimators': 5}	0.91970
1	1.874729	0.010904	0.112733	0.006491	2	10	{'max_depth': 2, 'n_estimators': 10}	0.91977
2	7.536040	0.040642	0.145592	0.001685	2	50	{'max_depth': 2, 'n_estimators': 50}	0.94427
3	14.585046	0.157704	0.216631	0.000395	2	100	{'max_depth': 2, 'n_estimators': 100}	0.96834
4	28.407301	0.195649	0.338354	0.010053	2	200	{'max_depth': 2, 'n_estimators':	0.97394

mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_n_estimators	param_n_estimators : 200	split0_test_sco

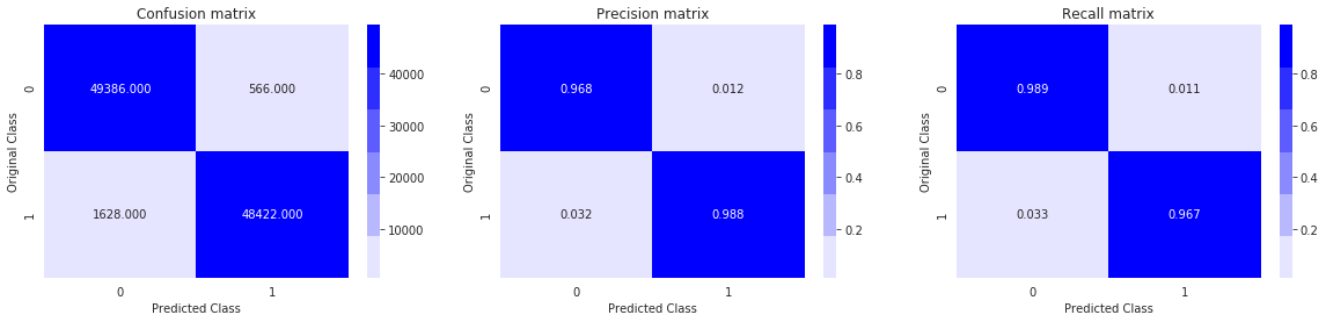
In [0]:

```
# from sklearn.metrics import f1_score
n_estimators = 500
max_depth = 2


xg_clf = XGBClassifier(max_depth = max_depth, n_estimators = n_estimators , n_jobs=1)
xg_clf.fit(df_final_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

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

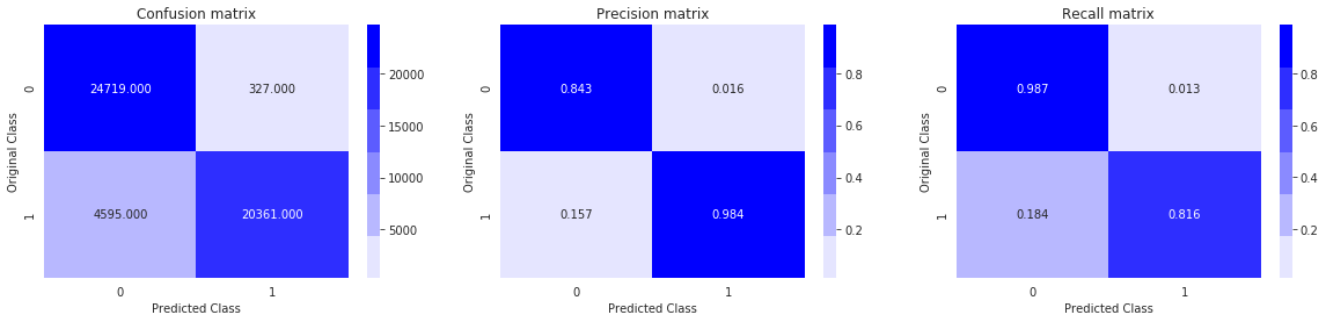
In [0]:

```
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



Conclusion

Step by Step step by step procedure I followed to solve this case study.

1. First I identify which type of machine learning problem this case study is.
2. Then I found the business constraints
3. Then I did the exploratory data analysis such as : Number of followees and follower, Number of people each person is following
4. Then I split the data into train and split
5. Then I created the other class 0
6. After that I created some features such as Jaccard Distance, Cosine Distance, page rank and many more
7. Then I did the hyperparameter tunning and trained Random Forest and XGboost

Comparision tables

In [28]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model","F1 Score(test)","Precission class 1(test)", "Precission class 0(test)", "Recall class 1(test)", "Recall class 0(test)"]
x.add_row(["Random Forest",0.92,97.3,89.1,97.6,88.0])
x.add_row(["XGBoost",0.98,98.4,84.3,81.6,98.7])
print(x)
```

```
+---------------+---------------+-------------------------+-------------------------+---------------------+---------------------+
|     Model     | F1 Score(test) | Precission class 1(test) | Precission class 0(test) | Recall class 1(test) | Recall class 0(test) |
+---------------+---------------+-------------------------+-------------------------+---------------------+---------------------+
| Random Forest |      0.92     |           97.3          |           89.1          |         97.6        |         88.0        |
|    XGBoost    |      0.98     |           98.4          |           84.3          |         81.6        |         98.7        |
+---------------+---------------+-------------------------+-------------------------+---------------------+---------------------+
```

In [0]: