Social network Graph Link Prediction - Facebook Challenge

In [1]:

```
#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 [34]:

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

In [35]:

```
df_final_train.columns
```

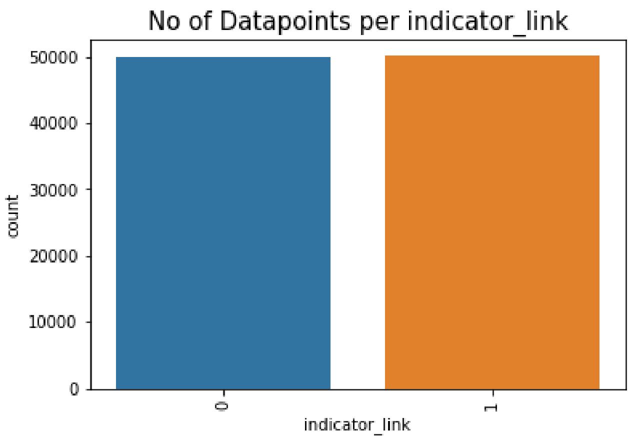
Out[35]:

```
Index(['source_node', 'destination_node', 'indicator_link',
       'jaccard_followers', 'jaccard_followees', 'cosine_followers',
       'cosine_followees', 'Preferential_attachmentfollowers',
       'Preferential_attachmentfollowees', 'num_followers_s',
       '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', 'svddot_U',
       'svddot_V'],
      dtype='object')
```

EDA

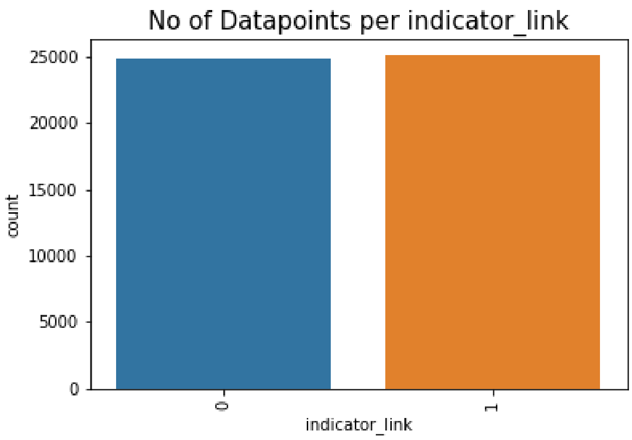
In [36]:

```
plt.title('No of Datapoints per indicator_link', fontsize=15)
sns.countplot(df_final_train.indicator_link)
plt.xticks(rotation=90)
plt.show()
```



In [37]:

```
plt.title('No of Datapoints per indicator_link', fontsize=15)
sns.countplot(df_final_test.indicator_link)
plt.xticks(rotation=90)
plt.show()
```

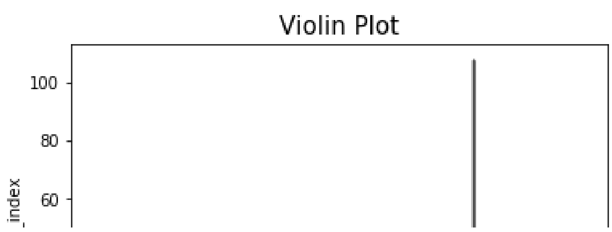


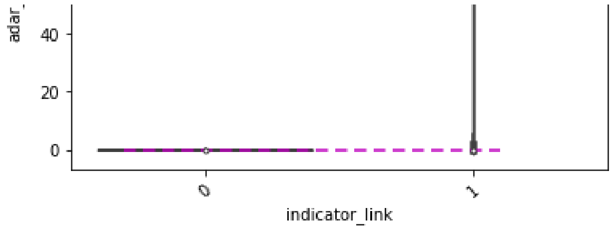
It is a Known in this case study, but we are again checking if Train data and Test data is balanced or not

Done some EDA's on new data .. Most of EDA's are not more usefull

In [38]:

```
sns.violinplot(x='indicator_link', y='adar_index', data = df_final_train, showfliers=False)
plt.title('Violin Plot', fontsize=15)
plt.xticks(rotation = 40)
plt.axhline(y=-0.22, xmin=0.1, xmax=0.8, dashes=(5,3), c='m')
plt.show()
```

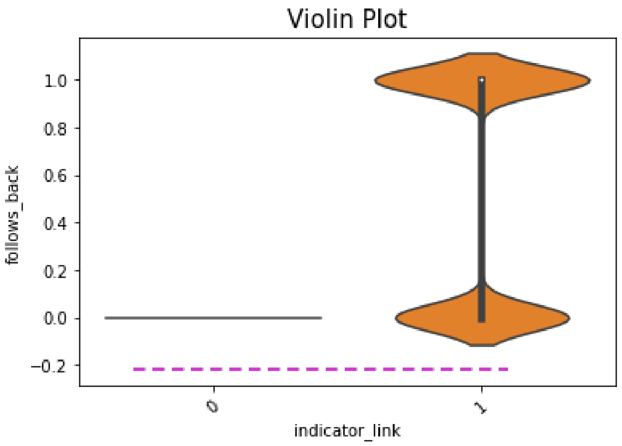




It is not that clear , but we can conclude that adar_index is not a single feature which can classify data nicely because, We can see value of indicator ink is totally overlapping at 0.

In [39]:

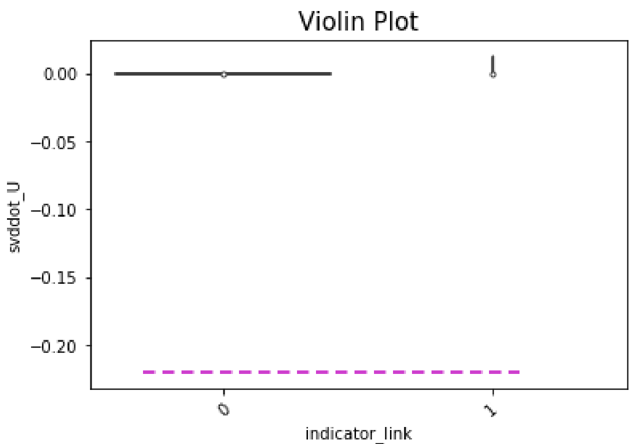
```
sns.violinplot(x='indicator_link', y='follows_back', data = df_final_train, showfliers=False)
plt.title('Violin Plot', fontsize=15)
plt.xticks(rotation = 40)
plt.axhline(y=-0.22, xmin=0.1, xmax=0.8, dashes=(5,3), c='m')
plt.show()
```



It is not that clear , but we can conclude that follows_back is not a single feature which can classify data nicely because, We can see value of indicator ink is totally overlapping at 0.

In [40]:

```
sns.violinplot(x='indicator_link', y='svddot_U', data = df_final_train, showfliers=False)
plt.title('Violin Plot', fontsize=15)
plt.xticks(rotation = 40)
plt.axhline(y=-0.22, xmin=0.1, xmax=0.8, dashes=(5,3), c='m')
plt.show()
```

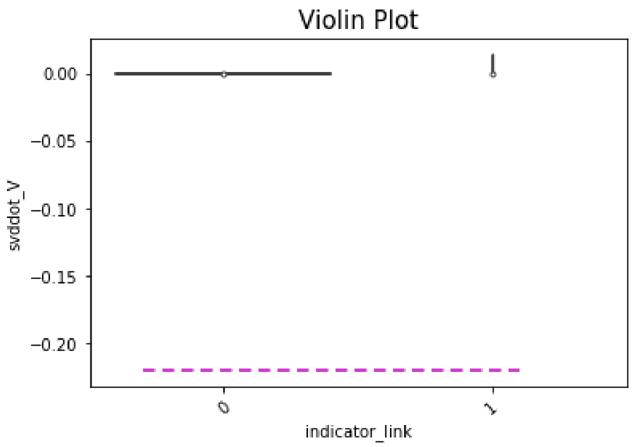


It is not that clear , but we can conclude that svddot_U is not a single feature which can classify data nicely because, We can see value of indicator ink is totally overlapping at 0.

In [41]:

```
sns.violinplot(x='indicator_link', y='svddot_V', data = df_final_train, showfliers=False)
plt.title('Violin Plot', fontsize=15)
plt.xticks(rotation = 40)
plt.axhline(y=-0.22, xmin=0.1, xmax=0.8, dashes=(5,3), c='m')
plt.show()
```

```
plt.show()
```



It is not that clear , but we can conclude that svddot_V is not a single feature which can classify data nicely because, We can see value of indicator ink is totally overlapping at 0.

In [4]:

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

In [5]:

```
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)
```

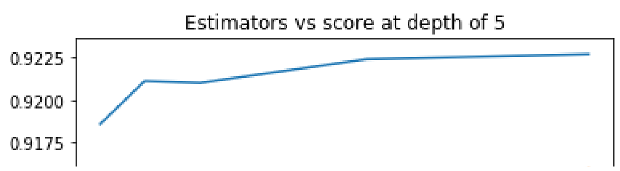
In [6]:

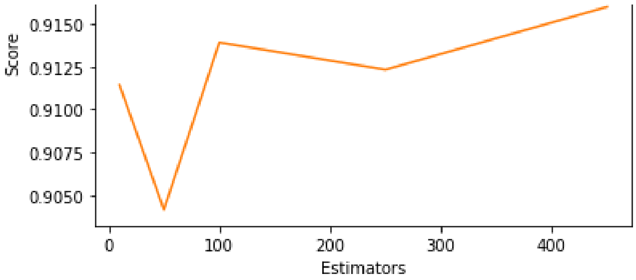
```
estimators = [10,50,100,250,450]
train_scores = []
test_scores = []
for i in estimators:
    clf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=5, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            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 0.9185874658029621 test Score 0.9114280905495246
Estimators =  50 Train Score 0.921102373716804 test Score 0.9041345035423496
Estimators =  100 Train Score 0.9209942670628112 test Score 0.9139125687649602
Estimators =  250 Train Score 0.9223810120763863 test Score 0.9123212783851975
Estimators =  450 Train Score 0.9226754624024049 test Score 0.9159927933967403
```

Out[6]:

```
Text(0.5,1,'Estimators vs score at depth of 5')
```



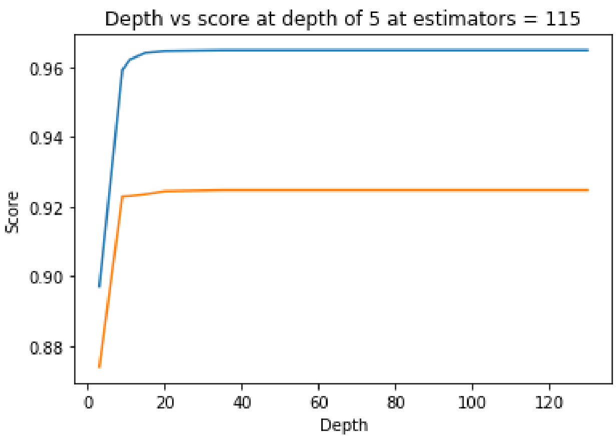


Here , we can observe that train score is not increasing much after estimators = 250, but , test score is being increasing even after estimators = 250

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,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 0.8971375039034039 test Score 0.8740709370037058
depth =  9 Train Score 0.9591407939292562 test Score 0.9228824805679018
depth =  11 Train Score 0.9621391723730935 test Score 0.9230088681987656
depth =  15 Train Score 0.964080065807513 test Score 0.9235239619304303
depth =  20 Train Score 0.9645918823386016 test Score 0.9243952249615764
depth =  35 Train Score 0.9648078953781172 test Score 0.9247280149828497
depth =  50 Train Score 0.9648078953781172 test Score 0.9247280149828497
depth =  70 Train Score 0.9648078953781172 test Score 0.9247280149828497
depth =  130 Train Score 0.9648078953781172 test Score 0.9247280149828497
```



The Train score and test score are not increaseng after the depth is more than 15

In [8]:

```
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 [0.96346914 0.96371786 0.96157795 0.96333565 0.96455314]
mean train scores [0.96436767 0.96434448 0.96222644 0.96389065 0.96553072]
```

In [9]:

```
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 [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 0.9657857910641491
Test f1 score 0.9220579239352839
```

In [14]:

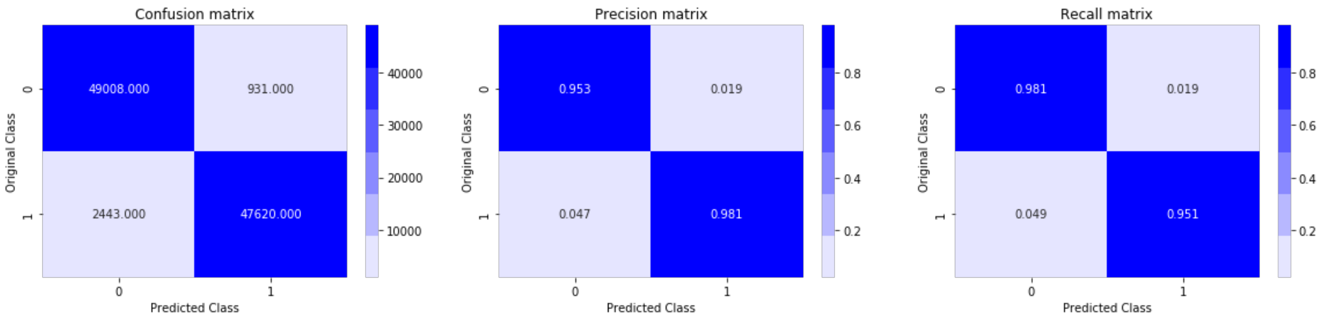
```
from sklearn.metrics import confusion_matrix
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)

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

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

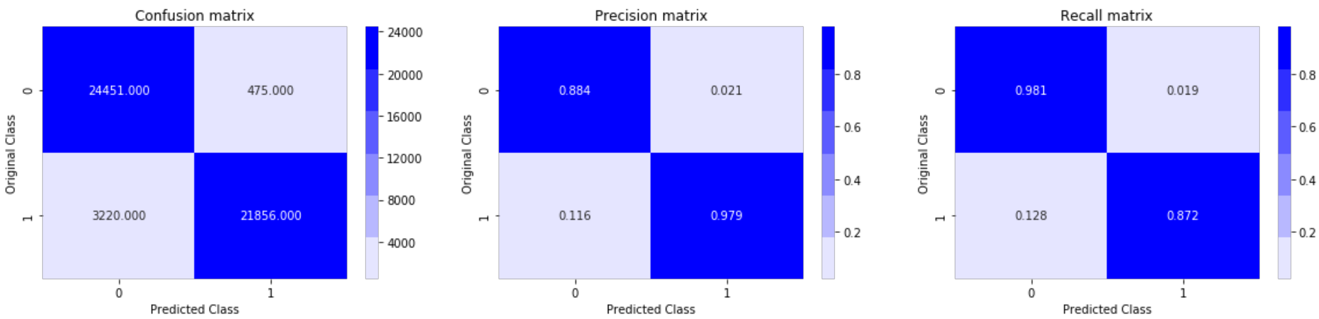
In [15]:

```
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
```



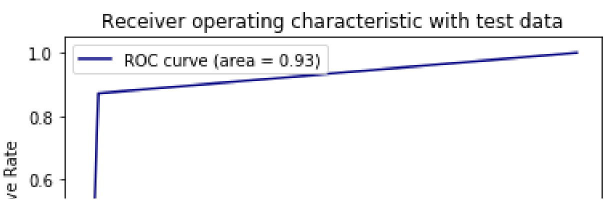

```
Test confusion_matrix
```

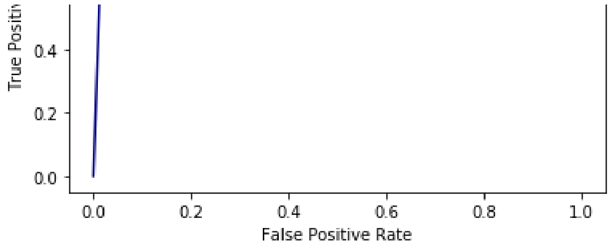


We can see here it is classifying both train and test data with good accuracy

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()
```

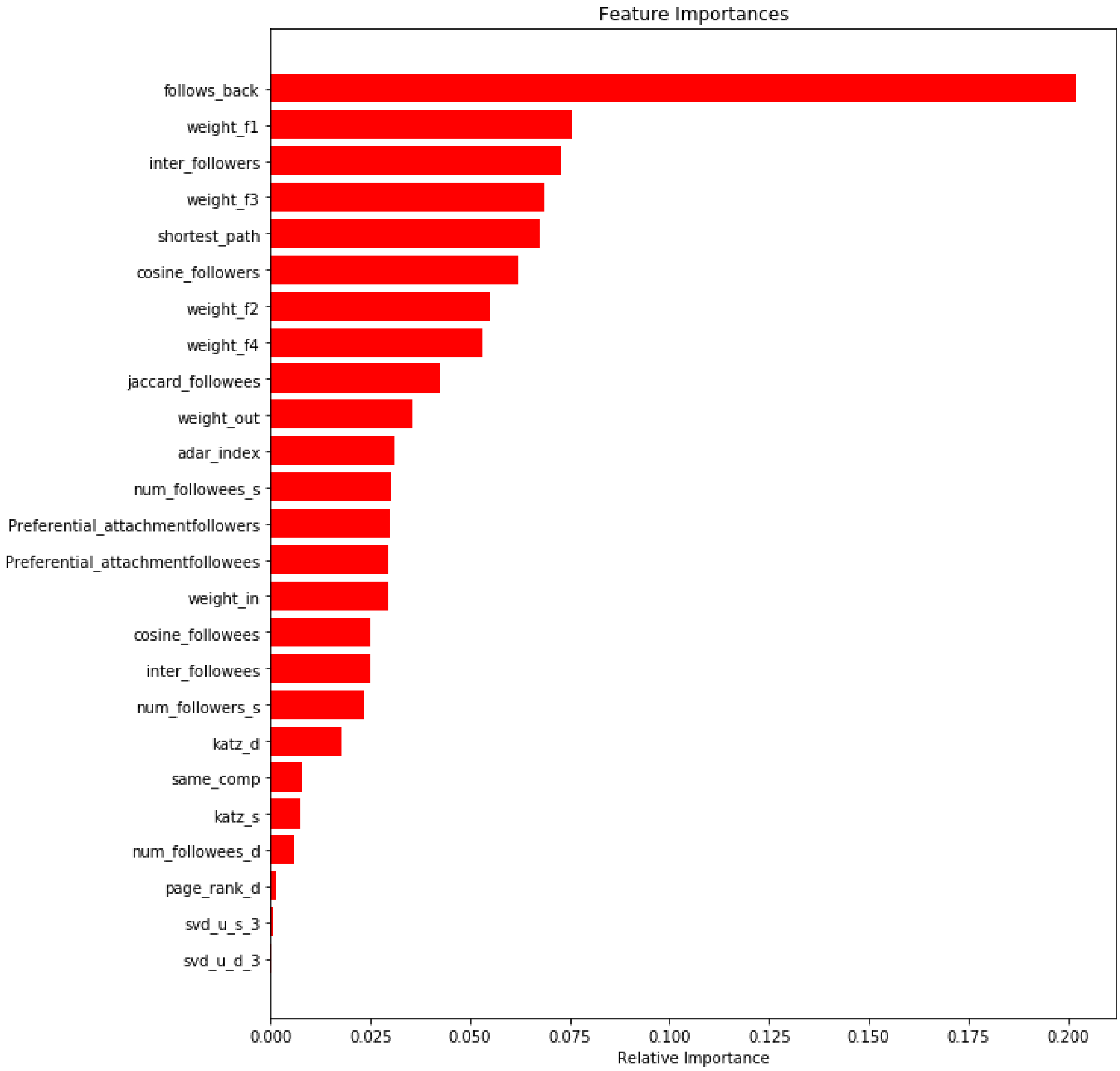




Even here with AUC curve we have good accuracy , that is area is 0.93

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 important features are followback and weight_f1 and svd features are not that important

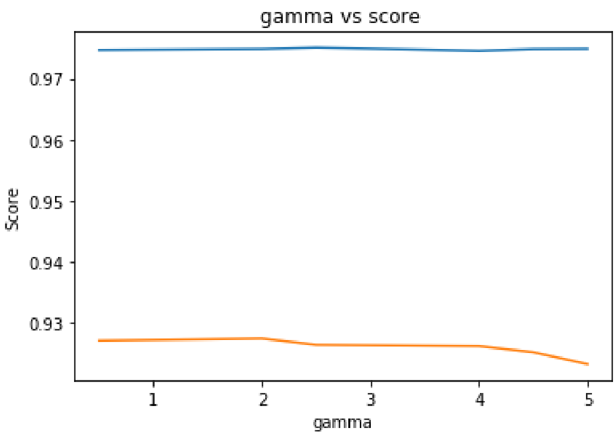
XGBOOST

In [20]:

```
import xgboost as xgb
gamma= [0.5, 2 , 2.5 , 4,4.5, 5]
train_scores = []
```

```
test_scores = []
for i in gamma:
    clf = xgb.XGBClassifier(gamma=i)
    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('gamma = ',i,'Train Score',train_sc,'test Score',test_sc)
plt.plot(gamma,train_scores,label='Train Score')
plt.plot(gamma,test_scores,label='Test Score')
plt.xlabel('gamma')
plt.ylabel('Score')
plt.title('gamma vs score ')
plt.show()
```

```
gamma =  0.5 Train Score 0.974666491091235 test Score 0.927059720780459
gamma =  2 Train Score 0.974832639585169 test Score 0.9274171343852589
gamma =  2.5 Train Score 0.9750610058626381 test Score 0.9263733468972533
gamma =  4 Train Score 0.9745373647487107 test Score 0.9261719411067862
gamma =  4.5 Train Score 0.9748113637514559 test Score 0.9251596749209583
gamma =  5 Train Score 0.9748567726785027 test Score 0.9232239566741001
```

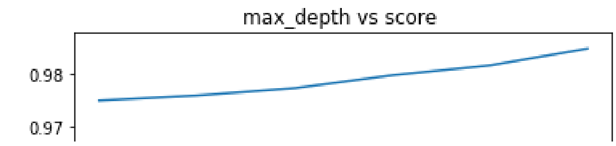


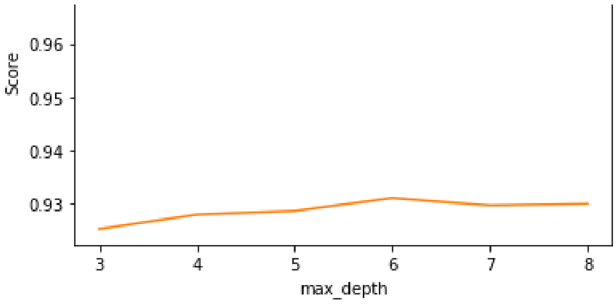
Here, we can observe that test score is decreasing after gamma is more than 4... So, there might be overfitting

In [32]:

```
import xgboost as xgb
max_depth=[3, 4, 5,6,7,8]
train_scores = []
test_scores = []
for i in max_depth:
    clf = xgb.XGBClassifier(max_depth=i)
    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('max_depth = ',i,'Train Score',train_sc,'test Score',test_sc)
plt.plot(max_depth,train_scores,label='Train Score')
plt.plot(max_depth,test_scores,label='Test Score')
plt.xlabel('max_depth')
plt.ylabel('Score')
plt.title('max_depth vs score ')
plt.show()
```

```
max_depth =  3 Train Score 0.9750506277845281 test Score 0.9253060098856573
max_depth =  4 Train Score 0.975979439857536 test Score 0.928010831164985
max_depth =  5 Train Score 0.9773552465233881 test Score 0.92868932244276l
max_depth =  6 Train Score 0.9797913489602796 test Score 0.9311101730688054
max_depth =  7 Train Score 0.9816506361784929 test Score 0.9297226505566001
max_depth =  8 Train Score 0.9847903769691478 test Score 0.9300244787709969
```





There is no drastic change , but there may be overfitting on further increasing of max_depth

In [23]:

```
from sklearn.metrics import f1_score
import xgboost as xgb
from sklearn.metrics import f1_score
from sklearn.model_selection import RandomizedSearchCV


xgb_model = xgb.XGBClassifier()

parameters = {
              'objective':['binary:logistic'],
              'gamma': [0.5, 2 , 2.5 , 4, 5],
              'max_depth': [3, 4, 5,6],
              'n_estimators': [5,6,7],
              }



clf = RandomizedSearchCV(xgb_model, parameters, cv=5,scoring="f1")
clf.fit(df_final_train,y_train)
print('mean test scores',clf.cv_results_['mean_test_score'])
print('mean train scores',clf.cv_results_['mean_train_score'])
```

```
mean test scores [0.93067189 0.93067189 0.92161439 0.92324099 0.91783364 0.91783364
 0.92506282 0.96131867 0.94249156 0.96131867]
mean train scores [0.93083209 0.93083209 0.92192876 0.92361113 0.91794432 0.91794432
 0.92519252 0.96177186 0.94248495 0.96177186]
```

In [24]:

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

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0.5, learning_rate=0.1, max_delta_step=0,
       max_depth=6, min_child_weight=1, missing=None, n_estimators=7,
       n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1)
```

In [26]:

```
clf = xgb.XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0.5, learning_rate=0.1, max_delta_step=0,
       max_depth=6, min_child_weight=1, missing=None, n_estimators=7,
       n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1)
```

In [27]:

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

In [28]:

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

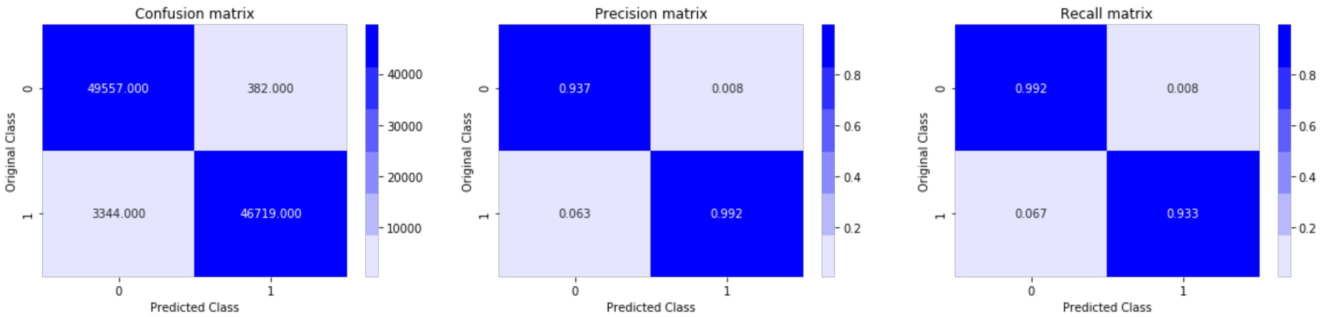
```
print('Train f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))
```

```
Train f1 score 0.9616524638755094
Test f1 score 0.9239972749723239
```

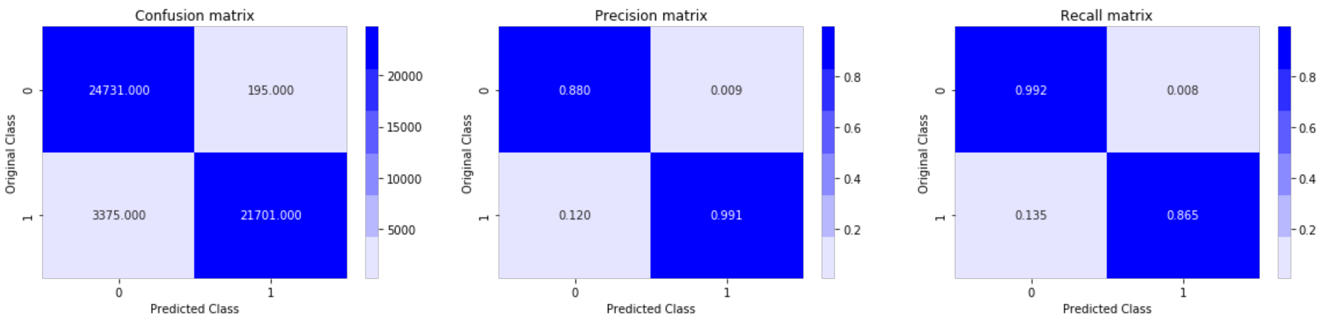
In [29]:

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

```
Train confusion_matrix
```



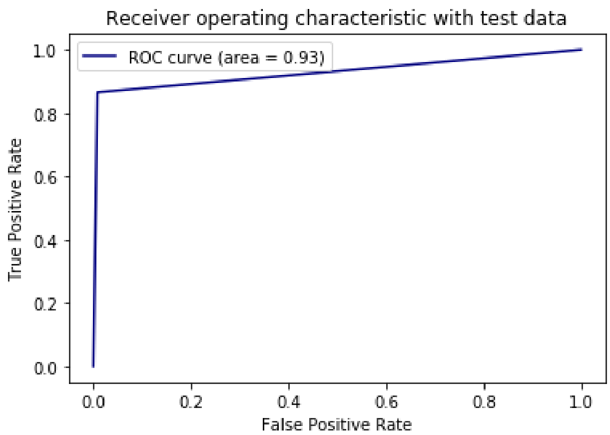

```
Test confusion_matrix
```



We are getting good accuracy than that of previous for test data

In [30]:

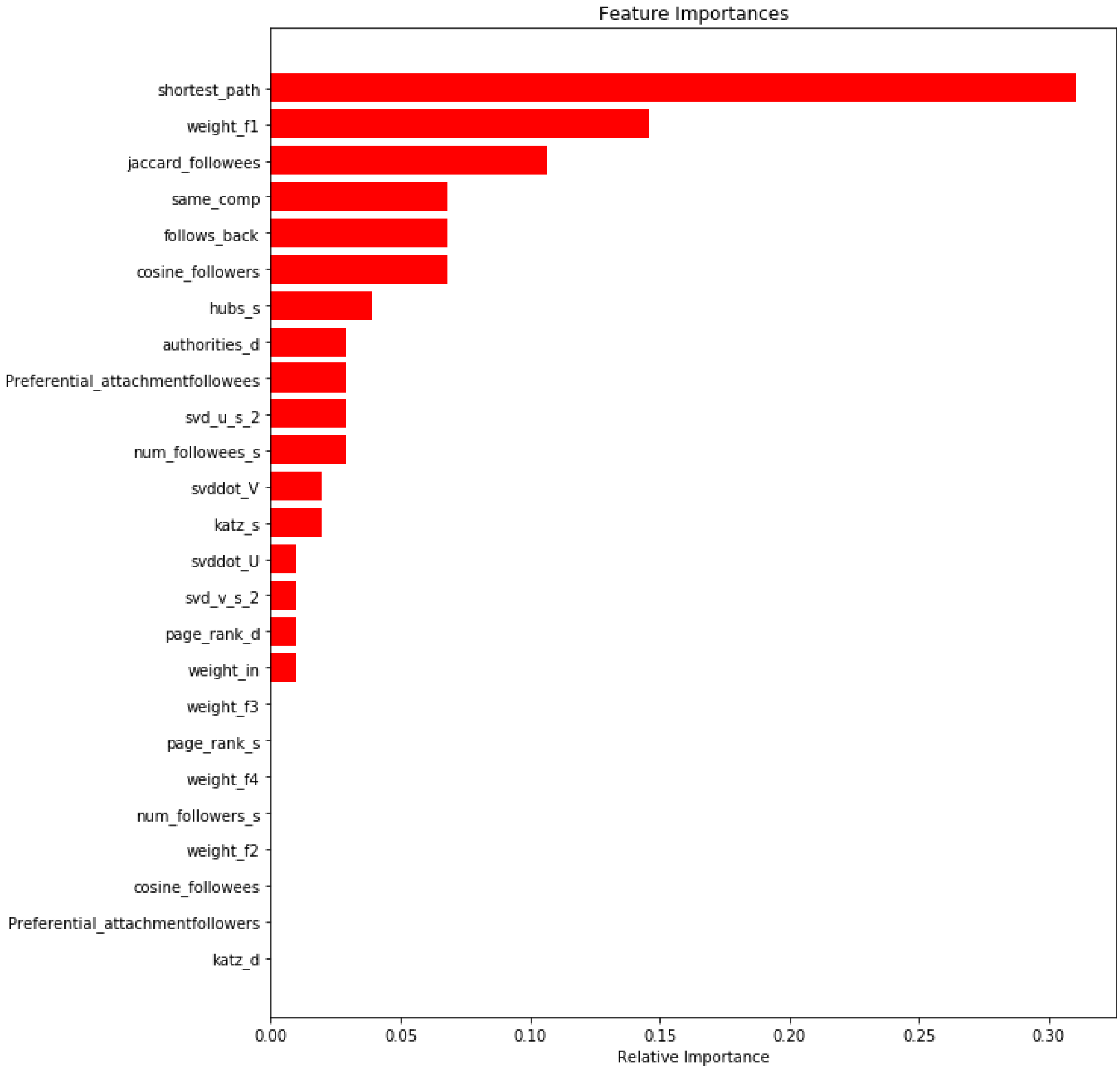
```
from sklearn.metrics import roc_curve, auc
fpr,tpr,ths = roc_curve(y_test,y_test_pred)
auc_sc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='navy',label='ROC curve (area = %0.2f)' % auc_sc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic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



AUC of both models have same area 0.93

In [31]:

```
features = df_final_train.columns
importances = clf.feature_importances_
indices = (np.argsort(importances))[-25:]
plt.figure(figsize=(10,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='r', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



Important features vary with that of the previous , Here the important features are shortest path, weight_f1 etc

In [43]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Model","Train  score","Test score"]

x.add_row(["Random Forest Classifier","0.9657857910641491","0.9220579239352839"])
x.add_row(["XGBOOST","0.9616524638755094","0.9239972749723239"])


print(x)
```

```
+--------------------------+--------------------+--------------------+
|          Model           |    Train  score    |     Test score     |
+--------------------------+--------------------+--------------------+
| Random Forest Classifier | 0.9657857910641491 | 0.9220579239352839 |
|         XGBOOST          | 0.9616524638755094 | 0.9239972749723239 |
+--------------------------+--------------------+--------------------+
```

+--------------------------+--------------------+--------------------+

Xg-boost have good test accuracy than that of Random Forest Classifier

Conclusion

EDA

1)importing all the packages 2)Using networknx get the graph data 3)Draw a graph for the sample data 4)Draw a line plot and box plots for indegree and number of followers 5)Draw a line plot and box plots for outdegree and number of followers 6)Draw a line plot and box plots for degree and number of followers 7)Compare the data with percentile values 8)Adding y value to the data and alsoadding some more values to have balanced y values 9)Splitting the data in to test and train

Featurisation

1)importing all the packages which are required 2)then calculating and adding all the features jaccard_followers jaccard_followees cosine_followers cosine_followees Preferential_attachmentfollowers Preferential_attachmentfollowees num_followers_s num_followees_s num_followers_d num_followees_d inter_followers inter_followees adar index is following back belongs to same weakly connect components shortest path between source and destination Weight Features Page Ranking of source Page Ranking of dest katz of source katz of dest hubs of source hubs of dest authorities_s of source authorities_s of dest SVD Features SVD Dot Features

Models

1)Get the data and try to do some EDA'S 2)Do hyper-parameter tuning and fit Random forest classifier 3)Do Hyper parameter tuning and fit Xgboost 4)compare both the models and their f1 scores