Social network Graph Link Prediction - Facebook Challenge

In [1]:

```
import warnings
warnings.filterwarnings("ignore")

import csv
import pandas as pd
import datetime
import time
import numpy as np
import matplotlib
import matplotlib.pylab as plt
import seaborn as sns
from matplotlib import rcParams
from sklearn.cluster import MiniBatchKMeans, KMeans
import math
import pickle
import os
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
```

2. Posing a problem as classification problem

2.1 Generating some edges which are not present in graph for supervised learning

Generated Bad links from graph which are not in graph and whose shortest path is greater than 2.

In [13]:

```
%%time
###generating bad edges from given graph
import random
if not os.path.isfile('after_eda/missing_edges_final.p'):
    #getting all set of edges
    r = csv.reader(open('after_eda/train_woheader.csv','r'))
    edges = dict()
    for edge in r:
        edges[(edge[0], edge[1])] = 1


    missing_edges = set([])
    while (len(missing_edges)<9437519):
        a=random.randint(1, 1862220)
        b=random.randint(1, 1862220)
        tmp = edges.get((a,b),-1)
        if tmp == -1 and a!=b:
            try:
                if nx.shortest_path_length(g,source=a,target=b) > 2:

                    missing_edges.add((a,b))
                else:
                    continue
            except:
                    missing_edges.add((a,b))
        else:
            continue
    pickle.dump(missing_edges,open('after_eda/missing_edges_final.p','wb'))
else:
    missing_edges = pickle.load(open('after_eda/missing_edges_final.p','rb'))
```

```
Wall time: 2.32 s
```

In [14]:

```
len(missing_edges)
```

Out[14]:

```
9437519
```

2.2 Training and Test data split:

Removed edges from Graph and used as test data and after removing used that graph for creating features for Train and test data

In [16]:

```
from sklearn.model_selection import train_test_split
if (not os.path.isfile('after_eda/train_pos_after_eda.csv')) and (not os.path
    #reading total data df
    df_pos = pd.read_csv('train.csv')
    df_neg = pd.DataFrame(list(missing_edges), columns=['source_node', 'desti

    print("Number of nodes in the graph with edges", df_pos.shape[0])
    print("Number of nodes in the graph without edges", df_neg.shape[0])

    #Trian test split
    #Spiltted data into 80-20
    #positive links and negative links seperatly because we need positive tra
    #and for feature generation
    X_train_pos, X_test_pos, y_train_pos, y_test_pos  = train_test_split(df_p
    X_train_neg, X_test_neg, y_train_neg, y_test_neg  = train_test_split(df_n

    print('='*60)
    print("Number of nodes in the train data graph with edges", X_train_pos.s
    print("Number of nodes in the train data graph without edges", X_train_ne
    print('='*60)
    print("Number of nodes in the test data graph with edges", X_test_pos.sha
    print("Number of nodes in the test data graph without edges", X_test_neg.

    #removing header and saving
    X_train_pos.to_csv('after_eda/train_pos_after_eda.csv',header=False, inde
    X_test_pos.to_csv('after_eda/test_pos_after_eda.csv',header=False, index=
    X_train_neg.to_csv('after_eda/train_neg_after_eda.csv',header=False, inde
    X_test_neg.to_csv('after_eda/test_neg_after_eda.csv',header=False, index=
else:
    #Graph from Traing data only
    del missing_edges
```

In [17]:

```
if (os.path.isfile('after_eda/train_pos_after_eda.csv')) and (os.path.isfile(
    train_graph=nx.read_edgelist('after_eda/train_pos_after_eda.csv',delimite
    test_graph=nx.read_edgelist('after_eda/test_pos_after_eda.csv',delimiter=
    print(nx.info(train_graph))
    print(nx.info(test_graph))

    # finding the unique nodes in the both train and test graphs
    train_nodes_pos = set(train_graph.nodes())
    test_nodes_pos = set(test_graph.nodes())

    trY_teY = len(train_nodes_pos.intersection(test_nodes_pos))
    trY_teN = len(train_nodes_pos - test_nodes_pos)
    teY_trN = len(test_nodes_pos - train_nodes_pos)

    print('no of people common in train and test -- ',trY_teY)
    print('no of people present in train but not present in test -- ',trY_teN

    print('no of people present in test but not present in train -- ',teY_trN
    print(' % of people not there in Train but exist in Test in total Test da
```

```
Name: 
Type: DiGraph
Number of nodes: 1780722
Number of edges: 7550015
Average in degree:   4.2399
Average out degree:   4.2399
Name: 
Type: DiGraph
Number of nodes: 1144623
Number of edges: 1887504
Average in degree:   1.6490
Average out degree:   1.6490
no of people common in train and test --  1063125
no of people present in train but not present in test --  7175
97
no of people present in test but not present in train --  8149
8
 % of people not there in Train but exist in Test in total Tes
t data are 7.120735962845405 %
```

we have a cold start problem here

In [18]:

```
#final train and test data sets
if (not os.path.isfile('after_eda/train_after_eda.csv')) and \
(not os.path.isfile('after_eda/test_after_eda.csv')) and \
(not os.path.isfile('train_y.csv')) and \
(not os.path.isfile('test_y.csv')) and \
(os.path.isfile('after_eda/train_pos_after_eda.csv')) and \
(os.path.isfile('after_eda/test_pos_after_eda.csv')) and \
(os.path.isfile('after_eda/train_neg_after_eda.csv')) and \
(os.path.isfile('after_eda/test_neg_after_eda.csv')):

    X_train_pos = pd.read_csv('after_eda/train_pos_after_eda.csv', names=['sc
    X_test_pos = pd.read_csv('after_eda/test_pos_after_eda.csv', names=['sour
    X_train_neg = pd.read_csv('after_eda/train_neg_after_eda.csv', names=['sc
    X_test_neg = pd.read_csv('after_eda/test_neg_after_eda.csv', names=['sour

    print('='*60)
    print("Number of nodes in the train data graph with edges", X_train_pos.s
    print("Number of nodes in the train data graph without edges", X_train_ne
    print('='*60)
    print("Number of nodes in the test data graph with edges", X_test_pos.sha
    print("Number of nodes in the test data graph without edges", X_test_neg.

    X_train = X_train_pos.append(X_train_neg,ignore_index=True)
    y_train = np.concatenate((y_train_pos,y_train_neg))
    X_test = X_test_pos.append(X_test_neg,ignore_index=True)
    y_test = np.concatenate((y_test_pos,y_test_neg))

    X_train.to_csv('after_eda/train_after_eda.csv',header=False,index=False)
    X_test.to_csv('after_eda/test_after_eda.csv',header=False,index=False)
    pd.DataFrame(y_train.astype(int)).to_csv('train_y.csv',header=False,index
    pd.DataFrame(y_test.astype(int)).to_csv('test_y.csv',header=False,index=F
```

In [19]:

```
print("Data points in train data",X_train.shape)
print("Data points in test data",X_test.shape)
print("Shape of traget variable in train",y_train.shape)
print("Shape of traget variable in test", y_test.shape)
```

```
--------------------------------------------------------------
-------------
NameError                                 Traceback (most rece
nt call last)
<ipython-input-19-a5d9170065c0> in <module>
----> 1 print("Data points in train data",X_train.shape)
      2 print("Data points in test data",X_test.shape)
      3 print("Shape of traget variable in train",y_train.shap
e)
      4 print("Shape of traget variable in test", y_test.shape
)

NameError: name 'X_train' is not defined
```

1. Reading Data

In [20]:

```
if os.path.isfile('after_eda/train_pos_after_eda.csv'):
    train_graph=nx.read_edgelist('after_eda/train_pos_after_eda.csv',delimite
    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
```

2. Similarity measures

2.1 Jaccard Distance:

[http://www.statisticshowto.com/jaccard-index/ (http://www.statisticshowto.com/jaccard-index/)](http://www.statisticshowto.com/jaccard-index/)

$$j = \frac{|X \cap Y|}{|X \cup Y|}$$

In [21]:

```
#for followees
def jaccard_for_followees(a,b):
    try:
        if len(set(train_graph.successors(a))) == 0  | len(set(train_graph.su
            return 0
        sim = (len(set(train_graph.successors(a)).intersection(set(train_grap
                                    (len(set(train_graph.successors(a)).union
    except:
        return 0
    return sim
```

In [22]:

```
#one test case
print(jaccard_for_followees(273084,1505602))
```

```
0.0
```

In [23]:

```
#node 1635354 not in graph
print(jaccard_for_followees(273084,1505602))
```

```
0.0
```

In [24]:

```
#for followers
def jaccard_for_followers(a,b):
    try:
        if len(set(train_graph.predecessors(a))) == 0  | len(set(g.predecesso
            return 0
        sim = (len(set(train_graph.predecessors(a)).intersection(set(train_gr
                                 (len(set(train_graph.predecessors(a)).union(
        return sim
    except:
        return 0
```

In [25]:

```
print(jaccard_for_followers(273084,470294))
```

```
0
```

In [26]:

```
#node 1635354 not in graph
print(jaccard_for_followees(669354,1635354))
```

```
0
```

2.2 Cosine distance

$$CosineDistance = \frac{|X \cap Y|}{sqrt(|X| \cdot |Y|)}$$

In [27]:

```
#for followees
def cosine_for_followees(a,b):
    try:
        if len(set(train_graph.successors(a))) == 0  | len(set(train_graph.su
            return 0
        sim = (len(set(train_graph.successors(a)).intersection(set(train_grap
                                    (math.sqrt(len(set(train_graph.successors
        return sim
    except:
        return 0
```

In [28]:

```
print(cosine_for_followees(273084,1505602))
```

```
0.0
```

In [29]:

```
print(cosine_for_followees(273084,1635354))
```

```
0
```

In [30]:

```
def cosine_for_followers(a,b):
    try:
        
        if len(set(train_graph.predecessors(a))) == 0  | len(set(train_graph.
            return 0
        sim = (len(set(train_graph.predecessors(a)).intersection(set(train_gr
                                     (math.sqrt(len(set(train_graph.predecess
        return sim
    except:
        return 0
```

In [31]:

```
print(cosine_for_followers(2,470294))
```

0.02886751345948129

In [32]:

```
print(cosine_for_followers(669354,1635354))
```

0

3. Ranking Measures

[https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.algorithms.link_analysis.pagerank_alg.pagerank.html (https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.algorithms.link_analysis.pagerank_alg.pagerank.html)](https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.algorithms.link_analysis.pagerank_alg.pagerank.html)

PageRank computes a ranking of the nodes in the graph G based on the structure of the incoming links.

Mathematical PageRanks for a simple network, expressed as percentages. (Google uses a logarithmic scale.) Page C has a higher PageRank than Page E, even though there are fewer links to C; the one link to C comes from an important page and hence is of high value. If web surfers who start on a random page have an 85% likelihood of choosing a random link from the page they are currently visiting, and a 15% likelihood of jumping to a page chosen at random from the entire web, they will reach Page E 8.1% of the time. **(The 15% likelihood of jumping to an arbitrary page corresponds to a damping factor of 85%.) Without damping, all web surfers would eventually end up on Pages A, B, or C, and all other pages would have PageRank zero. In the presence of damping, Page A effectively links to all pages in the web, even though it has no outgoing links of its own.**

to all pages in the web, even though it has no outgoing links of its own.

3.1 Page Ranking

[https://en.wikipedia.org/wiki/PageRank (https://en.wikipedia.org/wiki/PageRank)](https://en.wikipedia.org/wiki/PageRank)

In [34]:

```
if not os.path.isfile('fea_sample/page_rank.p'):
    pr = nx.pagerank(train_graph, alpha=0.85)
    pickle.dump(pr,open('fea_sample/page_rank.p','wb'))
else:
    pr = pickle.load(open('fea_sample/page_rank.p','rb'))
```

In [35]:

```
print('min',pr[min(pr, key=pr.get)])
print('max',pr[max(pr, key=pr.get)])
print('mean',float(sum(pr.values())) / len(pr))
```

```
min 1.6556497245737814e-07
max 2.7098251341935827e-05
mean 5.615699699389075e-07
```

In [36]:

```
#for imputing to nodes which are not there in Train data
mean_pr = float(sum(pr.values())) / len(pr)
print(mean_pr)
```

```
5.615699699389075e-07
```

4. Other Graph Features

4.1 Shortest path:

Getting Shortest path between twoo nodes, if nodes have direct path i.e directly connected then we are removing that edge and calculating path.

In [37]:

```
#if has direct edge then deleting that edge and calculating shortest path
def compute_shortest_path_length(a,b):
    p=-1
    try:
        if train_graph.has_edge(a,b):
            train_graph.remove_edge(a,b)
            p= nx.shortest_path_length(train_graph,source=a,target=b)
            train_graph.add_edge(a,b)
        else:
            p= nx.shortest_path_length(train_graph,source=a,target=b)
        return p
    except:
        return -1
```

In [38]:

```
#testing
compute_shortest_path_length(77697, 826021)
```

Out[38]:

```
10
```

In [39]:

```
#testing
compute_shortest_path_length(669354,1635354)
```

Out[39]:

```
-1
```

4.2 Checking for same community

In [40]:

```
#getting weekly connected edges from graph
wcc=list(nx.weakly_connected_components(train_graph))
def belongs_to_same_wcc(a,b):
    index = []
    if train_graph.has_edge(b,a):
        return 1
    if train_graph.has_edge(a,b):
            for i in wcc:
                if a in i:
                    index= i
                    break
            if (b in index):
                train_graph.remove_edge(a,b)
                if compute_shortest_path_length(a,b)==-1:
                    train_graph.add_edge(a,b)
                    return 0
                else:
                    train_graph.add_edge(a,b)
                    return 1
            else:
                return 0
    else:
            for i in wcc:
                if a in i:
                    index= i
                    break
            if(b in index):
                return 1
            else:
                return 0
```

In [41]:

```
belongs_to_same_wcc(861, 1659750)
```

Out[41]:

```
0
```

In [42]:

```
belongs_to_same_wcc(669354,1635354)
```

Out[42]:

```
0
```

4.3 Adamic/Adar Index:

Adamic/Adar measures is defined as inverted sum of degrees of common neighbours for given two vertices.

$$A(x, y) = \sum_{u \in N(x) \cap N(y)} \frac{1}{log(|N(u)|)}$$

In [43]:

```
#adar index
def calc_adar_in(a,b):
    sum=0
    try:
        n=list(set(train_graph.successors(a)).intersection(set(train_graph.su
        if len(n)!=0:
            for i in n:
                sum=sum+(1/np.log10(len(list(train_graph.predecessors(i)))))
            return sum
        else:
            return 0
    except:
        return 0
```

In [44]:

```
calc_adar_in(1,189226)
```

Out[44]:

```
0
```

In [45]:

```
calc_adar_in(669354,1635354)
```

Out[45]:

```
0
```

4.4 Is persion was following back:

In [46]:

```
def follows_back(a,b):
    if train_graph.has_edge(b,a):
        return 1
    else:
        return 0
```

In [47]:

```
follows_back(1,189226)
```

Out[47]:

```
1
```

In [48]:

```
follows_back(669354,1635354)
```

Out[48]:

```
0
```

4.5 Katz Centrality:

https://en.wikipedia.org/wiki/Katz_centrality (https://en.wikipedia.org/wiki/Katz_centrality)

https://www.geeksforgeeks.org/katz-centrality-centrality-measure/ (https://www.geeksforgeeks.org/katz-centrality-centrality-measure/) Katz centrality computes the centrality for a node based on the centrality of its neighbors. It is a generalization of the eigenvector centrality. The Katz centrality for node `i` is

$$x_i = \alpha \sum_j A_{ij} x_j + \beta,$$

where `A` is the adjacency matrix of the graph G with eigenvalues

$$\lambda$$

.

The parameter

$$\beta$$

controls the initial centrality and

$$\alpha < \frac{1}{\lambda_{max}}.$$

In [50]:

```
if not os.path.isfile('fea_sample/katz.p'):
    katz = nx.katz.katz_centrality(train_graph,alpha=0.005,beta=1)
    pickle.dump(katz,open('fea_sample/katz.p','wb'))
else:
    katz = pickle.load(open('fea_sample/katz.p','rb'))
```

In [51]:

```
print('min',katz[min(katz, key=katz.get)])
print('max',katz[max(katz, key=katz.get)])
print('mean',float(sum(katz.values())) / len(katz))
```

```
min 0.0007313532484065916
max 0.003394554981699122
mean 0.0007483800935562018
```

In [52]:

```
mean_katz = float(sum(katz.values())) / len(katz)
print(mean_katz)
```

```
0.0007483800935562018
```

4.6 Hits Score

The HITS algorithm computes two numbers for a node. Authorities estimates the node value based on the incoming links. Hubs estimates the node value based on outgoing links.

https://en.wikipedia.org/wiki/HITS_algorithm (https://en.wikipedia.org/wiki/HITS_algorithm)

In [53]:

```
if not os.path.isfile('fea_sample/hits.p'):
    hits = nx.hits(train_graph, max_iter=100, tol=1e-08, nstart=None, normali
    pickle.dump(hits,open('fea_sample/hits.p','wb'))
else:
    hits = pickle.load(open('fea_sample/hits.p','rb'))
```

In [54]:

```
print('min',hits[0][min(hits[0], key=hits[0].get)])
print('max',hits[0][max(hits[0], key=hits[0].get)])
print('mean',float(sum(hits[0].values())) / len(hits[0]))
```

```
min 0.0
max 0.004868653378780953
mean 5.615699699344123e-07
```

Preferential Attachment

Preferential Attachment One well-known concept in social networks is that users with many friends tend to create more connections in the future. This is due to the fact that in some social networks, like in finance, the rich get richer. We estimate how ”rich” our two vertices are by calculating the multiplication between the number of friends (|Γ(x)|) or followers each vertex has. It may be noted that the similarity index does not require any node neighbor information; therefore, this similarity index has the lowest computational complexity

http://be.amazd.com/link-prediction/ (http://be.amazd.com/link-prediction/)

<PA>

    The link between A and C is more probable than the link between A an
    d B as C have many more neighbors than B

In [55]:

```
def preferential_attachment_followees(a,b):

    try:
        if len(set(train_graph.successors(a))) == 0  | len(set(train_graph.su
                return 0
        else:
            pref=len(set(train_graph.successors(a)))*len(set(train_graph.succ
            return pref
    except:
        return 0
```

In [56]:

```
def preferential_attachment_followers(a,b):

    try:
        if len(set(train_graph.predecessors(a))) == 0  | len(set(train_graph.
                return 0
        else:
            pref=len(set(train_graph.predecessors(a)))*len(set(train_graph.pr
            return pref
    except:
        return 0
```

5. Featurization

5. 1 Reading a sample of Data from both train and test

In [57]:

```
import random
if os.path.isfile('after_eda/train_after_eda.csv'):
    filename = "after_eda/train_after_eda.csv"
    # you uncomment this line, if you dont know the lentgh of the file name
    # here we have hardcoded the number of lines as 15100030
    # n_train = sum(1 for line in open(filename)) #number of records in file
    n_train =  15100028
    s = 100000 #desired sample size
    skip_train = sorted(random.sample(range(1,n_train+1),n_train-s))
    #https://stackoverflow.com/a/22259008/4084039
```

In [58]:

```
if os.path.isfile('after_eda/test_after_eda.csv'):
    filename = "after_eda/test_after_eda.csv"
    # you uncomment this line, if you dont know the lentgh of the file name
    # here we have hardcoded the number of lines as 3775008
    # n_test = sum(1 for line in open(filename)) #number of records in file (
    n_test = 3775006
    s = 50000 #desired sample size
    skip_test = sorted(random.sample(range(1,n_test+1),n_test-s))
    #https://stackoverflow.com/a/22259008/4084039
```

In [59]:

```
print("Number of rows in the train data file:", n_train)
print("Number of rows we are going to elimiate in train data are",len(skip_tr
print("Number of rows in the test data file:", n_test)
print("Number of rows we are going to elimiate in test data are",len(skip_tes
```

```
Number of rows in the train data file: 15100028
Number of rows we are going to elimiate in train data are 1500
0028
Number of rows in the test data file: 3775006
Number of rows we are going to elimiate in test data are 37250
06
```

In [60]:

```
df_final_train = pd.read_csv('after_eda/train_after_eda.csv', skiprows=skip_t
df_final_train['indicator_link'] = pd.read_csv('train_y.csv', skiprows=skip_t
print("Our train matrix size ",df_final_train.shape)
df_final_train.head(2)
```

```
Our train matrix size  (100002, 3)
```

Out[60]:

	source_node	destination_node	indicator_link
0	273084	1505602	1
1	196674	421096	1

In [61]:

```
df_final_test = pd.read_csv('after_eda/test_after_eda.csv', skiprows=skip_tes
df_final_test['indicator_link'] = pd.read_csv('test_y.csv', skiprows=skip_tes
print("Our test matrix size ",df_final_test.shape)
df_final_test.head(2)
```

```
Our test matrix size  (50002, 3)
```

Out[61]:

	source_node	destination_node	indicator_link
0	848424	784690	1
1	1071089	1083536	1

5.2 Adding a set of features

we will create these each of these features for both train and test data points

1. jaccard_followers
2. jaccard_followees
3. cosine_followers
4. cosine_followees
5. num_followers_s
6. num_followees_s
7. num_followers_d
8. num_followees_d
9. inter_followers
10. inter_followees

In [62]:

```
if not os.path.isfile('fea_sample/storage_sample_stage1.h5'):
    #mapping jaccrd followers to train and test data
    df_final_train['jaccard_followers'] = df_final_train.apply(lambda row:
                                            jaccard_for_followers(row['source
    df_final_test['jaccard_followers'] = df_final_test.apply(lambda row:
                                            jaccard_for_followers(row['source

    #mapping jaccrd followees to train and test data
    df_final_train['jaccard_followees'] = df_final_train.apply(lambda row:
                                            jaccard_for_followees(row['source
    df_final_test['jaccard_followees'] = df_final_test.apply(lambda row:
                                            jaccard_for_followees(row['source


        #mapping jaccrd followers to train and test data
    df_final_train['cosine_followers'] = df_final_train.apply(lambda row:
                                            cosine_for_followers(row['source_
    df_final_test['cosine_followers'] = df_final_test.apply(lambda row:
                                            cosine_for_followers(row['source_

    #mapping jaccrd followees to train and test data
    df_final_train['cosine_followees'] = df_final_train.apply(lambda row:
                                            cosine_for_followees(row['source_
    df_final_test['cosine_followees'] = df_final_test.apply(lambda row:
                                            cosine_for_followees(row['source_
```

In [63]:

```
def compute_features_stage1(df_final):
    #calculating no of followers followees for source and destination
    #calculating intersection of followers and followees for source and desti
    num_followers_s=[]
    num_followees_s=[]
    num_followers_d=[]
    num_followees_d=[]
    inter_followers=[]
    inter_followees=[]
    for i,row in df_final.iterrows():
        try:
            s1=set(train_graph.predecessors(row['source_node']))
            s2=set(train_graph.successors(row['source_node']))
        except:
            s1 = set()
            s2 = set()
        try:
            d1=set(train_graph.predecessors(row['destination_node']))
            d2=set(train_graph.successors(row['destination_node']))
        except:
            d1 = set()
            d2 = set()
        num_followers_s.append(len(s1))
        num_followees_s.append(len(s2))

        num_followers_d.append(len(d1))
        num_followees_d.append(len(d2))

        inter_followers.append(len(s1.intersection(d1)))
        inter_followees.append(len(s2.intersection(d2)))

    return num_followers_s, num_followers_d, num_followees_s, num_followees_c
```

In [65]:

```
if not os.path.isfile('fea_sample/storage_sample_stage1.h5'):
    df_final_train['num_followers_s'], df_final_train['num_followers_d'], \
    df_final_train['num_followees_s'], df_final_train['num_followees_d'], \
    df_final_train['inter_followers'], df_final_train['inter_followees']= cor

    df_final_test['num_followers_s'], df_final_test['num_followers_d'], \
    df_final_test['num_followees_s'], df_final_test['num_followees_d'], \
    df_final_test['inter_followers'], df_final_test['inter_followees']= compu

    hdf = HDFStore('data/fea_sample/storage_sample_stage1.h5')
    hdf.put('train_df',df_final_train, format='table', data_columns=True)
    hdf.put('test_df',df_final_test, format='table', data_columns=True)
    hdf.close()
else:
    df_final_train = read_hdf('fea_sample/storage_sample_stage1.h5', 'train_c
    df_final_test = read_hdf('fea_sample/storage_sample_stage1.h5', 'test_df
```

5.3 Adding new set of features

we will create these each of these features for both train and test data points

1. adar index
2. is following back
3. belongs to same weakly connect components
4. shortest path between source and destination

In [66]:

```
if not os.path.isfile('fea_sample/storage_sample_stage2.h5'):
    #mapping adar index on train
    df_final_train['adar_index'] = df_final_train.apply(lambda row: calc_adar
    #mapping adar index on test
    df_final_test['adar_index'] = df_final_test.apply(lambda row: calc_adar_i

    #--------------------------------------------------------------------------
    #mapping followback or not on train
    df_final_train['follows_back'] = df_final_train.apply(lambda row: follows

    #mapping followback or not on test
    df_final_test['follows_back'] = df_final_test.apply(lambda row: follows_b

    #--------------------------------------------------------------------------
    #mapping same component of wcc or not on train
    df_final_train['same_comp'] = df_final_train.apply(lambda row: belongs_to

    ##mapping same component of wcc or not on train
    df_final_test['same_comp'] = df_final_test.apply(lambda row: belongs_to_s

    #--------------------------------------------------------------------------
    #mapping shortest path on train
    df_final_train['shortest_path'] = df_final_train.apply(lambda row: comput
    #mapping shortest path on test
    df_final_test['shortest_path'] = df_final_test.apply(lambda row: compute_

    hdf = HDFStore('fea_sample/storage_sample_stage2.h5')
    hdf.put('train_df',df_final_train, format='table', data_columns=True)
    hdf.put('test_df',df_final_test, format='table', data_columns=True)
    hdf.close()
else:
    df_final_train = read_hdf('fea_sample/storage_sample_stage2.h5', 'train_d
    df_final_test = read_hdf('fea_sample/storage_sample_stage2.h5', 'test_df
```

5.4 Adding new set of features

we will create these each of these features for both train and test data points

1. Weight Features
    - weight of incoming edges
    - weight of outgoing edges
    - weight of incoming edges + weight of outgoing edges
    - weight of incoming edges * weight of outgoing edges
    - 2*weight of incoming edges + weight of outgoing edges
    - weight of incoming edges + 2*weight of outgoing edges
2. Page Ranking of source

3. Page Ranking of dest
4. katz of source
5. katz of dest
6. hubs of source
7. hubs of dest
8. authorities_s of source
9. authorities_s of dest

Weight Features

In order to determine the similarity of nodes, an edge weight value was calculated between nodes. Edge weight decreases as the neighbor count goes up. Intuitively, consider one million people following a celebrity on a social network then chances are most of them never met each other or the celebrity. On the other hand, if a user has 30 contacts in his/her social network, the chances are higher that many of them know each other. `credit` - Graph-based Features for Supervised Link Prediction William Cukierski, Benjamin Hamner, Bo Yang

$$W = \frac{1}{\sqrt{1 + |X|}}$$

it is directed graph so calculated Weighted in and Weighted out differently

In [67]:

```
#weight for source and destination of each link
Weight_in = {}
Weight_out = {}
for i in  tqdm(train_graph.nodes()):
    s1=set(train_graph.predecessors(i))
    w_in = 1.0/(np.sqrt(1+len(s1)))
    Weight_in[i]=w_in

    s2=set(train_graph.successors(i))
    w_out = 1.0/(np.sqrt(1+len(s2)))
    Weight_out[i]=w_out

#for imputing with mean
mean_weight_in = np.mean(list(Weight_in.values()))
mean_weight_out = np.mean(list(Weight_out.values()))
```

```
100%|██████████████████████████████████████████████████████████████
██████████| 1780722/1780722 [00:11<00:00, 149099.47it/s]
```

In [68]:

```
if not os.path.isfile('fea_sample/storage_sample_stage3.h5'):
    #mapping to pandas train
    df_final_train['weight_in'] = df_final_train.destination_node.apply(lambd
    df_final_train['weight_out'] = df_final_train.source_node.apply(lambda x:

    #mapping to pandas test
    df_final_test['weight_in'] = df_final_test.destination_node.apply(lambda
    df_final_test['weight_out'] = df_final_test.source_node.apply(lambda x: W


    #some features engineerings on the in and out weights
    df_final_train['weight_f1'] = df_final_train.weight_in + df_final_train.w
    df_final_train['weight_f2'] = df_final_train.weight_in * df_final_train.w
    df_final_train['weight_f3'] = (2*df_final_train.weight_in + 1*df_final_tr
    df_final_train['weight_f4'] = (1*df_final_train.weight_in + 2*df_final_tr

    #some features engineerings on the in and out weights
    df_final_test['weight_f1'] = df_final_test.weight_in + df_final_test.weig
    df_final_test['weight_f2'] = df_final_test.weight_in * df_final_test.weig
    df_final_test['weight_f3'] = (2*df_final_test.weight_in + 1*df_final_test
    df_final_test['weight_f4'] = (1*df_final_test.weight_in + 2*df_final_test
```

In [69]:

```
if not os.path.isfile('fea_sample/storage_sample_stage3.h5'):

    #page rank for source and destination in Train and Test
    #if anything not there in train graph then adding mean page rank
    df_final_train['page_rank_s'] = df_final_train.source_node.apply(lambda x
    df_final_train['page_rank_d'] = df_final_train.destination_node.apply(lam

    df_final_test['page_rank_s'] = df_final_test.source_node.apply(lambda x:p
    df_final_test['page_rank_d'] = df_final_test.destination_node.apply(lambd
    #================================================================================

    #Katz centrality score for source and destination in Train and test
    #if anything not there in train graph then adding mean katz score
    df_final_train['katz_s'] = df_final_train.source_node.apply(lambda x: kat
    df_final_train['katz_d'] = df_final_train.destination_node.apply(lambda x

    df_final_test['katz_s'] = df_final_test.source_node.apply(lambda x: katz.
    df_final_test['katz_d'] = df_final_test.destination_node.apply(lambda x:
    #================================================================================

    #Hits algorithm score for source and destination in Train and test
    #if anything not there in train graph then adding 0
    df_final_train['hubs_s'] = df_final_train.source_node.apply(lambda x: hit
    df_final_train['hubs_d'] = df_final_train.destination_node.apply(lambda x

    df_final_test['hubs_s'] = df_final_test.source_node.apply(lambda x: hits[
    df_final_test['hubs_d'] = df_final_test.destination_node.apply(lambda x:
    #================================================================================

    #Hits algorithm score for source and destination in Train and Test
    #if anything not there in train graph then adding 0
    df_final_train['authorities_s'] = df_final_train.source_node.apply(lambda
    df_final_train['authorities_d'] = df_final_train.destination_node.apply(l

    df_final_test['authorities_s'] = df_final_test.source_node.apply(lambda x
    df_final_test['authorities_d'] = df_final_test.destination_node.apply(lam
    #================================================================================

    hdf = HDFStore('fea_sample/storage_sample_stage3.h5')
    hdf.put('train_df',df_final_train, format='table', data_columns=True)
    hdf.put('test_df',df_final_test, format='table', data_columns=True)
    hdf.close()
else:
    df_final_train = read_hdf('fea_sample/storage_sample_stage3.h5', 'train_d
    df_final_test = read_hdf('fea_sample/storage_sample_stage3.h5', 'test_df
```

5.5 Adding new set of features

we will create these each of these features for both train and test data points

1. SVD features for both source and destination

In [70]:

```
def svd(x, S):
    try:
        z = sadj_dict[x]
        return S[z]
    except:
        return [0,0,0,0,0,0]
```

In [71]:

```
#for svd features to get feature vector creating a dict node val and inedx ir
sadj_col = sorted(train_graph.nodes())
sadj_dict = { val:idx for idx,val in enumerate(sadj_col)}
```

In [72]:

```
Adj = nx.adjacency_matrix(train_graph,nodelist=sorted(train_graph.nodes())).a
```

In [73]:

```
U, s, V = svds(Adj, k = 6)
print('Adjacency matrix Shape',Adj.shape)
print('U Shape',U.shape)
print('V Shape',V.shape)
print('s Shape',s.shape)
```

```
Adjacency matrix Shape (1780722, 1780722)
U Shape (1780722, 6)
V Shape (6, 1780722)
s Shape (6,)
```

In [74]:

```
if not os.path.isfile('fea_sample/storage_sample_stage4.h5'):
    #===================================================================================

    df_final_train[['svd_u_s_1', 'svd_u_s_2','svd_u_s_3', 'svd_u_s_4', 'svd_u
    df_final_train.source_node.apply(lambda x: svd(x, U)).apply(pd.Series)

    df_final_train[['svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3', 'svd_u_d_4', 'svd_
    df_final_train.destination_node.apply(lambda x: svd(x, U)).apply(pd.Serie
    #===================================================================================

    df_final_train[['svd_v_s_1','svd_v_s_2', 'svd_v_s_3', 'svd_v_s_4', 'svd_v
    df_final_train.source_node.apply(lambda x: svd(x, V.T)).apply(pd.Series)

    df_final_train[['svd_v_d_1', 'svd_v_d_2', 'svd_v_d_3', 'svd_v_d_4', 'svd_
    df_final_train.destination_node.apply(lambda x: svd(x, V.T)).apply(pd.Ser
    #===================================================================================

    df_final_test[['svd_u_s_1', 'svd_u_s_2','svd_u_s_3', 'svd_u_s_4', 'svd_u_
    df_final_test.source_node.apply(lambda x: svd(x, U)).apply(pd.Series)

    df_final_test[['svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3', 'svd_u_d_4', 'svd_u
    df_final_test.destination_node.apply(lambda x: svd(x, U)).apply(pd.Series

    #===================================================================================

    df_final_test[['svd_v_s_1','svd_v_s_2', 'svd_v_s_3', 'svd_v_s_4', 'svd_v_
    df_final_test.source_node.apply(lambda x: svd(x, V.T)).apply(pd.Series)

    df_final_test[['svd_v_d_1', 'svd_v_d_2', 'svd_v_d_3', 'svd_v_d_4', 'svd_v
    df_final_test.destination_node.apply(lambda x: svd(x, V.T)).apply(pd.Seri
    #===================================================================================
```

In [75]:

```
df_final_train['preferential_attachment'] = df_final_train.apply(lambda row:
                                            preferential_attachment_followers
df_final_test['preferential_attachment'] = df_final_test.apply(lambda row:
                                            preferential_attachment_followers

    #mapping jaccrd followees to train and test data
df_final_train['preferential_attachment'] = df_final_train.apply(lambda row:
                                            preferential_attachment_followees
df_final_test['preferential_attachment'] = df_final_test.apply(lambda row:
                                            preferential_attachment_followees
```

SVD_DOT_Train

In [76]:

```
### svd decomposition
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components=20, n_iter=7, random_state=42)
svd_mat = svd.fit_transform(Adj)
```

In [77]:

```
def svd_dot(df):
    svd_dot = []
    for idx,temp in tqdm(df.iterrows(),total=df.shape[0]):
        source_index=sadj_dict.get(temp.destination_node,None)
        dest_index=sadj_dict.get(temp['source_node'])
        if ( source_index is not None and dest_index is not None):
            svd_temp = np.dot(svd_mat[source_index,:],svd_mat[dest_index,:])
            svd_dot.append(svd_temp)
        else:
            svd_dot.append(0)
    return svd_dot
```

In [78]:

```
df_final_train.shape[0]
df_final_train.iterrows()
```

Out[78]:

```
<generator object DataFrame.iterrows at 0x00000146A6C63AC8>
```

In [79]:

```python
df_final_train['svd_dot']=svd_dot(df_final_train)
df_final_test['svd_dot']=svd_dot(df_final_test)

print(df_final_train.head(5))
print(df_final_test.head(5))
```

```
100%|██████████████████████████████████████████████████████████████████
██████████████| 100002/100002 [00:06<00:00, 14851.38it/s]
100%|██████████████████████████████████████████████████████████████████
████████████████| 50002/50002 [00:03<00:00, 14721.06it/s]

   source_node  destination_node  indicator_link  jaccard_foll
owers  \
0       273084           1505602               1
0
1       832016           1543415               1
0
2      1325247            760242               1
0
3      1368400           1006992               1
0
4       140165           1708748               1
0

   jaccard_followees  cosine_followers  cosine_followees  num_
followers_s  \
0           0.000000          0.000000          0.000000
6
1           0.187135          0.028382          0.343828
94
2           0.369565          0.156957          0.566038
28
3           0.000000          0.000000          0.000000
11
4           0.000000          0.000000          0.000000
1

   num_followees_s  num_followees_d  ...   page_rank_s   page_
rank_d  \
0               15                8  ...  2.045290e-06  3.4599
63e-07
1               61              142  ...  2.353458e-07  6.4276
60e-07
2               41               22  ...  6.211019e-07  5.1798
01e-07
3                5                7  ...  2.998153e-07  1.7042
45e-06
4               11                3  ...  4.349180e-07  2.0895
90e-07
```

```
     katz_s    katz_d         hubs_s         hubs_d  authoriti
es_s  \
0  0.000773  0.000756   1.943132e-13   1.941103e-13   9.226339
e-16
1  0.000845  0.001317   3.906648e-11   9.424102e-11   1.208074
e-11
2  0.000885  0.000855  7.730764e-114  4.067322e-114  2.681298e
-113
3  0.000739  0.000773   5.443738e-17   4.139999e-16   2.413250
e-14
4  0.000751  0.000735   3.887821e-16   4.721269e-16   7.552255
e-16

   authorities_d  preferential_attachment       svd_dot
0   2.231877e-15                      120  1.183292e-07
1   1.273080e-10                     8662  3.495259e+01
2  2.199205e-113                      902  2.001332e-07
3   6.688064e-15                       35  5.038634e-16
4   2.734009e-18                       33  8.257568e-10

[5 rows x 32 columns]
   source_node  destination_node  indicator_link  jaccard_foll
owers  \
0       848424            784690               1
0
1       483294           1255532               1
0
2       626190           1729265               1
0
3       947219            425228               1
0
4       991374            975044               1
0

   jaccard_followees  cosine_followers  cosine_followees  num_
followers_s  \
0                0.0          0.029161          0.000000
14
1                0.0          0.000000          0.000000
17
2                0.0          0.000000          0.000000
10
3                0.0          0.000000          0.000000
37
4                0.2          0.042767          0.347833
27

   num_followees_s  num_followees_d  ...   page_rank_s  page_r
ank_d    katz_s  \
0                6                9  ...  6.557971e-07     0.0
00002  0.000754
1                1               19  ...  2.172064e-07     0.0
00001  0.000739
```

```
2                16                 9  ...  1.853369e-06     0.0
00002  0.000789
3                10                34  ...  7.000791e-07     0.0
00002  0.000778
4                15                27  ...  7.103008e-07     0.0
00001  0.000779

     katz_d        hubs_s        hubs_d  authorities_s  author
ities_d  \
0  0.000786  3.243237e-16  1.745627e-16   2.969838e-15   9.269
213e-14
1  0.000801  1.702625e-19  2.706300e-15   2.522357e-16   5.277
458e-15
2  0.000770  9.426796e-14  4.116616e-16   2.253244e-15   2.079
387e-15
3  0.000884  9.876114e-14  1.039593e-13   1.511694e-14   3.478
438e-14
4  0.000840  1.557332e-15  1.096037e-14   5.180869e-15   1.296
135e-14

   preferential_attachment       svd_dot
0                       54  9.146824e-16
1                       19  1.435302e-13
2                      144  3.897289e-09
3                      340  1.734630e-06
4                      405  1.323532e-07

[5 rows x 32 columns]
```

In [80]:

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

Out[80]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_fc
0	273084	1505602	1	0	(
1	832016	1543415	1	0	(
2	1325247	760242	1	0	(
3	1368400	1006992	1	0	(
4	140165	1708748	1	0	(

5 rows × 32 columns

In [81]:

```
#save
hdf = HDFStore('fea_sample/storage_sample_stage4.h5')
hdf.put('train_df',df_final_train, format='table', data_columns=True)
hdf.put('test_df',df_final_test, format='table', data_columns=True)
hdf.close()
```

In [2]:

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

In [3]:

```
df_final_train.columns
```

Out[3]:

```
Index(['source_node', 'destination_node', 'indicator_link',
       'jaccard_followers', 'jaccard_followees', 'cosine_follo
wers',
       'cosine_followees', 'num_followers_s', 'num_followees_
s',
       'num_followees_d', 'inter_followers', 'inter_followee
s', 'adar_index',
       'follows_back', 'same_comp', 'shortest_path', 'weight_i
n', 'weight_out',
       'weight_f1', 'weight_f2', 'weight_f3', 'weight_f4', 'pa
ge_rank_s',
       'page_rank_d', 'katz_s', 'katz_d', 'hubs_s', 'hubs_d',
'authorities_s',
       'authorities_d', 'preferential_attachment', 'svd_dot'],
      dtype='object')
```

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
df_final_test.drop(['source_node', 'destination_node','indicator_link'],axis=
```

In [6]:

```
df_final_train.head(2)
```

Out[6]:

	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	nı
0	0	0.000000	0.000000	0.000000	
1	0	0.187135	0.028382	0.343828	

2 rows × 29 columns

In [7]:

```
df_final_test.head(2)
```

Out[7]:

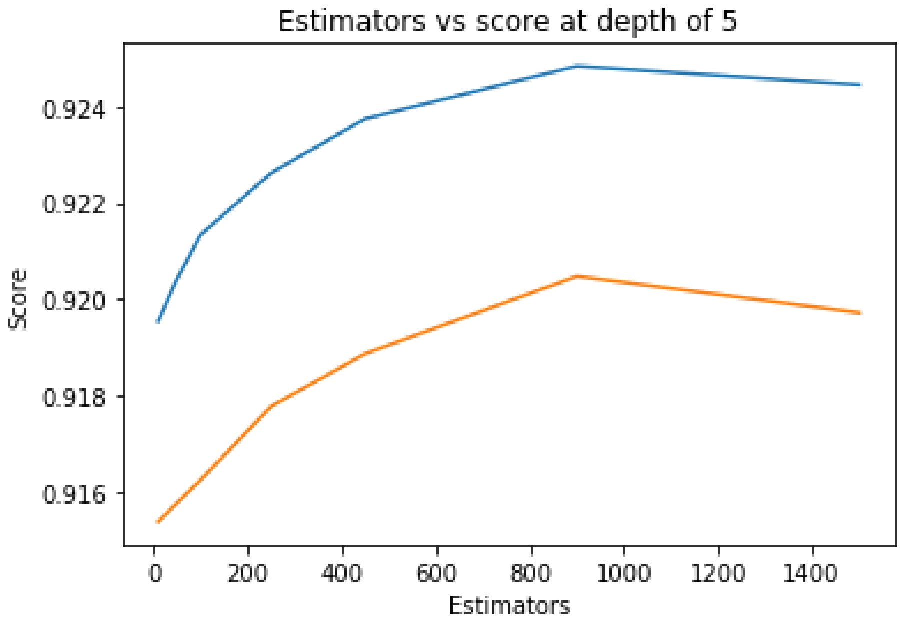
	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	nı
0	0	0.0	0.029161	0.0	
1	0	0.0	0.000000	0.0	

2 rows × 29 columns

In [13]:

```
estimators = [10,50,100,250,450,900,1500]
train_scores = []
test_scores = []
for i in estimators:
    clf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion
            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_st
    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')
plt.show()
```

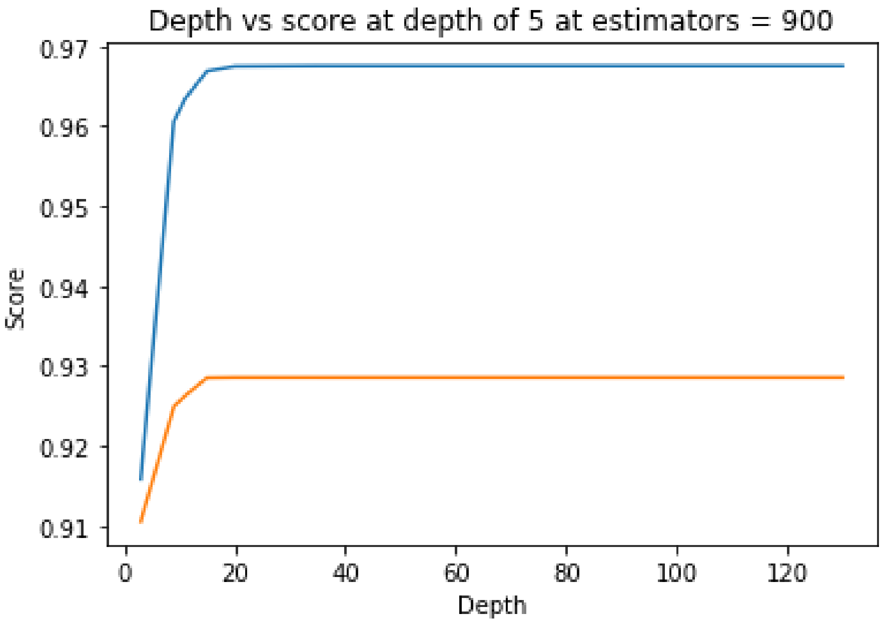
```
Estimators =  10 Train Score 0.9195402298850576 test Score 0.9
153693033001573
Estimators =  50 Train Score 0.9204132689912176 test Score 0.9
157598420234444
Estimators =  100 Train Score 0.9213368146214099 test Score 0.
916238599433903
Estimators =  250 Train Score 0.922622538751571 test Score 0.9
177687601185844
Estimators =  450 Train Score 0.9237528047476252 test Score 0.
9188665740389879
Estimators =  900 Train Score 0.9248462716951038 test Score 0.
9204763410264517
Estimators =  1500 Train Score 0.9244566585550055 test Score
0.9197194076383477
```


Estimators vs score at depth of 5
Score
0.924
0.922
0.920
0.918
0.916
0
200
400
600
800
1000
1200
1400
Estimators

In [14]:

```
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
            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=900, n_jobs=-1,random_
    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 = 900')
plt.show()
```

```
depth =  3 Train Score 0.9158752679708779 test Score 0.9105390
118116785
depth =  9 Train Score 0.9605794565560177 test Score 0.9249304
911955514
depth =  11 Train Score 0.963415748591156 test Score 0.9262581
596125501
depth =  15 Train Score 0.966892117991827 test Score 0.9285383
221674461
depth =  20 Train Score 0.9675006840361172 test Score 0.928581
9590994293
depth =  35 Train Score 0.9675510245444779 test Score 0.928578
950693991
depth =  50 Train Score 0.9675510245444779 test Score 0.928578
950693991
depth =  70 Train Score 0.9675510245444779 test Score 0.928578
950693991
depth =  130 Train Score 0.9675510245444779 test Score 0.92857
8950693991
```



In [15]:

```
rf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gir
                            max_depth=15, 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=900, n_jobs
rf.fit(df_final_train,y_train)
train_sc = f1_score(y_train,rf.predict(df_final_train))
test_sc = f1_score(y_test,rf.predict(df_final_test))
test_scores.append(test_sc)
train_scores.append(train_sc)
```

In [16]:

```
print('Train Score',train_sc,'test Score',test_sc)
```

```
Train Score 0.966892117991827 test Score 0.9285383221674461
```

In [20]:

```
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, return_tr
                                   n_iter=5,cv=10,scoring='f1',random_state=2

rf_random.fit(df_final_train,y_train)
```

Out[20]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=RandomForestClassifier(bootstr
ap=True,
                                                    ccp_alp
ha=0.0,
                                                    class_w
eight=None,
                                                    criteri
on='gini',
                                                    max_dep
th=None,
                                                    max_fea
tures='auto',
                                                    max_lea
f_nodes=None,
                                                    max_sam
ples=None,
                                                    min_imp
urity_decrease=0.0,
                                                    min_imp
urity_split=None,
                                                    min_sam
ples_leaf=1,
                                                    min_sam
ples_split=2,
                                                    min_wei
ght_fraction_leaf=0.0,
                                                    n_estim
```

```
ators=100, n_job...
                                        'min_samples_leaf':
<scipy.stats._distn_infrastructure.rv_frozen object at 0x00
00022426DC1748>,
                                        'min_samples_spli
t': <scipy.stats._distn_infrastructure.rv_frozen object at
0x0000022435367248>,
                                        'n_estimators': <sc
ipy.stats._distn_infrastructure.rv_frozen object at 0x00000
22435359048>},
                   pre_dispatch='2*n_jobs', random_state=2
5, refit=True,
                   return_train_score=True, scoring='f1', v
erbose=0)
```

In [21]:

```
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 [0.96468386 0.96423467 0.96230892 0.96371365
0.96599687]
mean train scores [0.96584897 0.96518664 0.96320689 0.96492208
0.96769218]
```

In [22]:

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

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_we
ight=None,
                       criterion='gini', max_depth=14, max_fea
tures='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_estimat
ors=121,
                       n_jobs=-1, oob_score=False, random_stat
e=25, verbose=0,
                       warm_start=False)
```

In [23]:

```
clf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gi
            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)
clf.fit(df_final_train,y_train)
y_train_pred = clf.predict(df_final_train)
y_test_pred = clf.predict(df_final_test)
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 0.9683115724596546
Test f1 score 0.9286105609170213
```

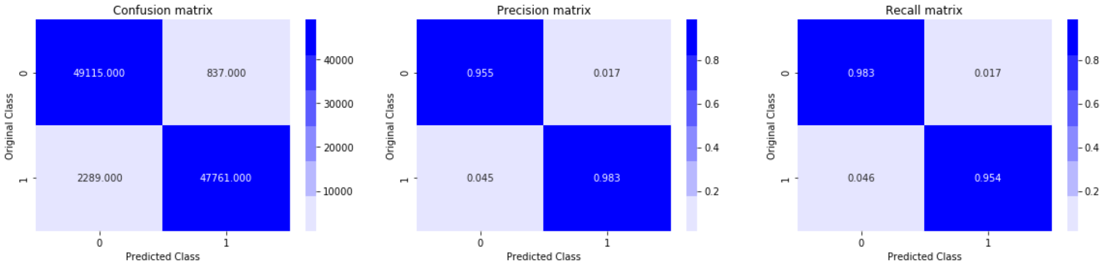
In [23]:

```
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, ytic
    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, ytic
    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, ytic
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()
```

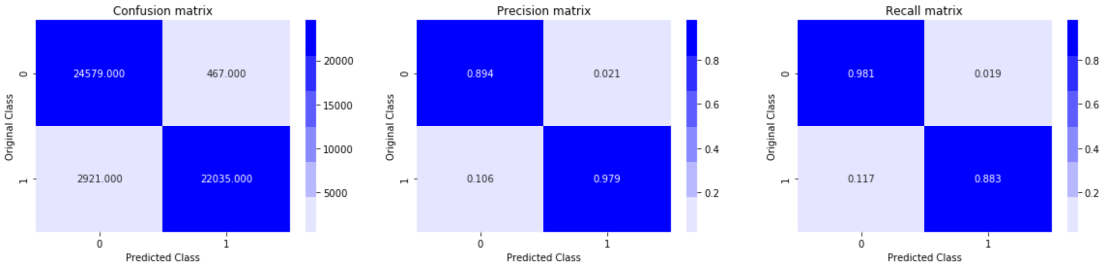
In [25]:

```
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

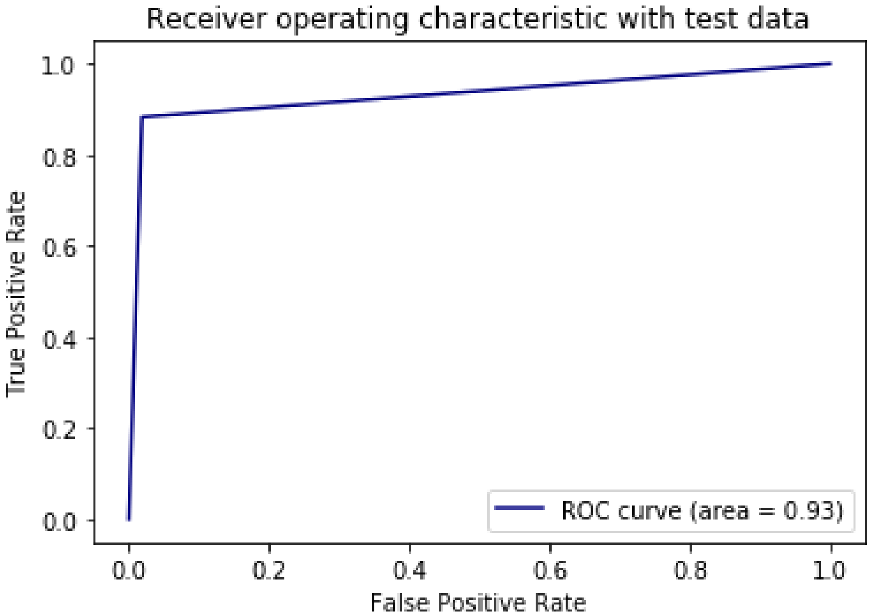


Test confusion_matrix



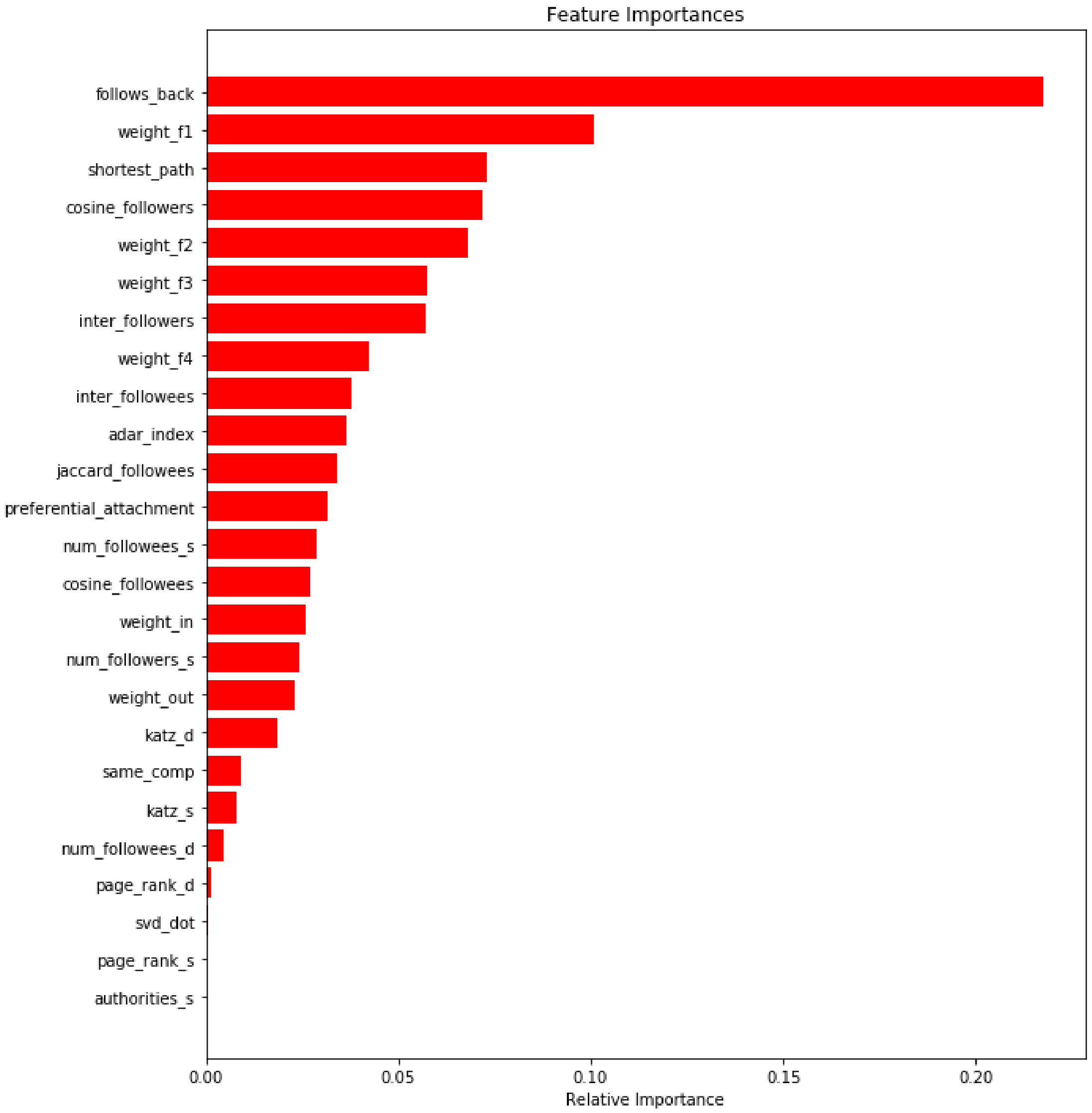
In [26]:

```
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 [27]:

```
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()
```



In [15]:

```
from sklearn.model_selection import GridSearchCV
parameters = {'max_depth' : [1, 2, 3, 4, 5, 6], 'n_estimators' : [50, 100, 20
xgbdt = xgb.XGBClassifier()
clf = GridSearchCV(xgbdt, parameters, cv=5, scoring='f1', return_train_score=
clf.fit(df_final_train, y_train)

results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_max_depth'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
K =  results['param_max_depth']
M = results['param_n_estimators']
```

In [16]:

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

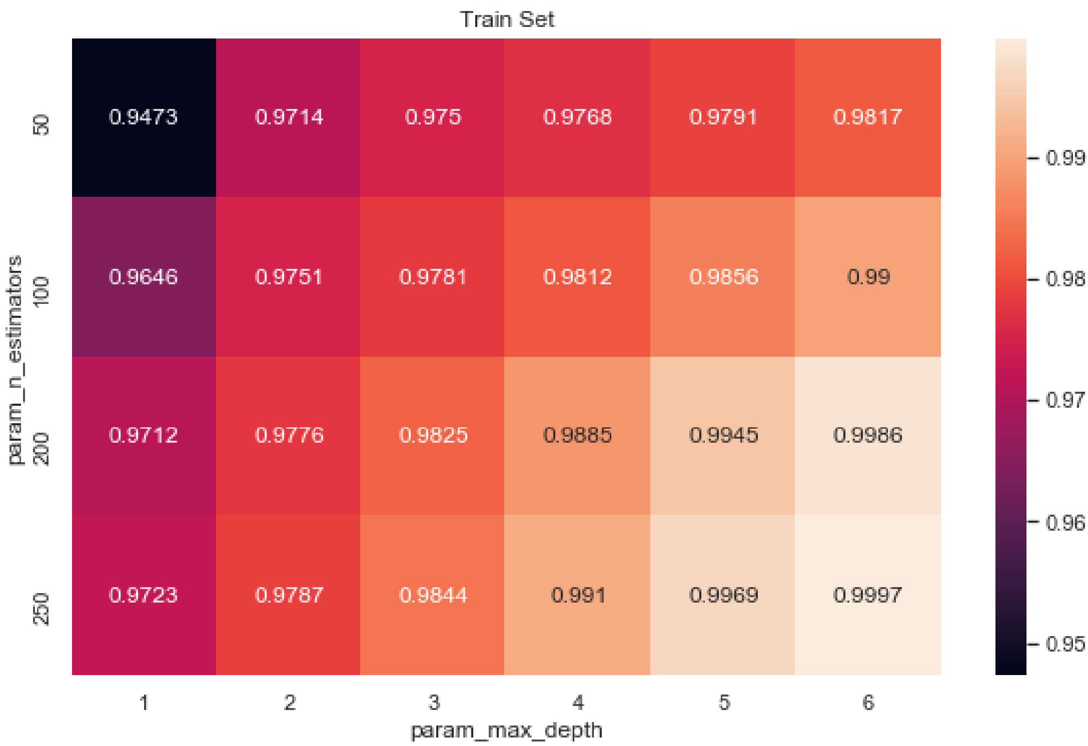
```
mean test scores [0.94735663 0.96434487 0.97088951 0.97192188
0.97124476 0.97457844
 0.97603335 0.97641587 0.9742475  0.97614307 0.97809365 0.9786
1316
 0.97526755 0.97735254 0.97839569 0.97863248 0.976341   0.9783
1572
 0.97902    0.97881794 0.97670599 0.97804677 0.97856133 0.9787
2454]
mean train scores [0.94734191 0.96459872 0.97122507 0.97230166
0.97144833 0.97510043
 0.97763043 0.97869985 0.97497693 0.97807448 0.98251409 0.9843
7387
 0.97681339 0.98119592 0.98848443 0.99101129 0.97905974 0.9856
1648
 0.99449126 0.99685666 0.98166075 0.99003781 0.9986299  0.9997
0271]
```

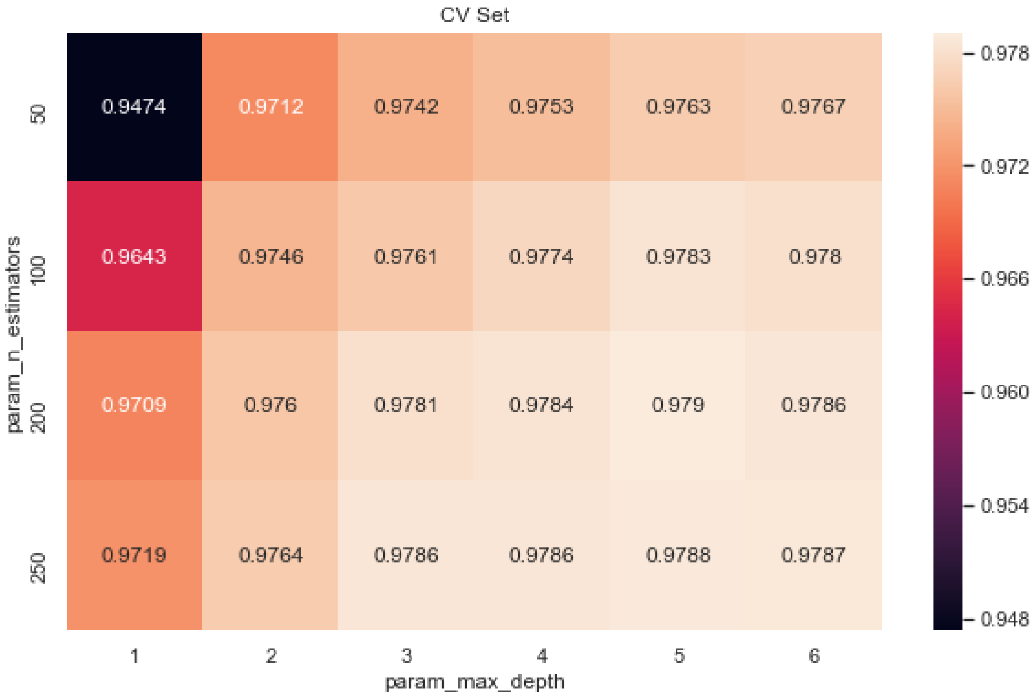
In [34]:

```
import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
trace1 = go.Scatter3d(x = K, y = M, z = train_auc, name = 'Train')
trace2 = go.Scatter3d(x = K, y = M, z = cv_auc, name = 'Cross Validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(xaxis = dict(title = 'max_depth'), yaxis = di
                              zaxis = dict(title = 'F1'),))

fig = go.Figure(data = data, layout = layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

In [19]:

```
import seaborn as sns; sns.set()
max_scores1 = pd.DataFrame(clf.cv_results_).groupby(['param_n_estimators', 'p
plt.figure(figsize=(10,6))
plt.title('Train Set')
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g')
plt.show()


plt.figure(figsize=(10,6))
plt.title('CV Set')
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g')
plt.show()
```





In [20]:

```
best_max_depth_tfidf_xgbdt = clf.best_params_['max_depth']
best_n_estimators_tfidf_xgbdt = clf.best_params_['n_estimators']
print('best value for max depth is {} and best value for n_estimators is {}'.
```

```
best value for max depth is 5 and best value for n_estimators
is 200
```

In [21]:

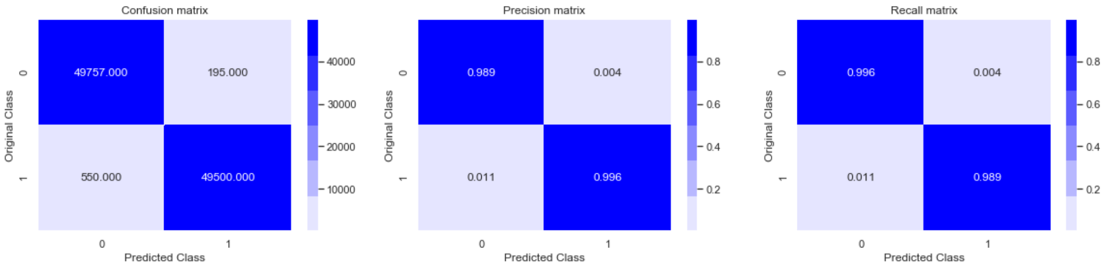
```
xgbdt = xgb.XGBClassifier(max_depth= best_max_depth_tfidf_xgbdt, n_estimators
xgbdt.fit(df_final_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability esti
# not the predicted outputs
y_train_pred = xgbdt.predict(df_final_train)
y_test_pred = xgbdt.predict(df_final_test)
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 0.992530953932528
Test f1 score 0.9051731583038258
```

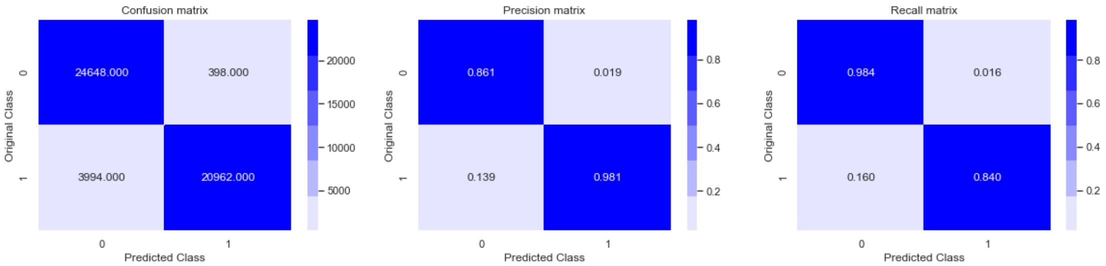
In [25]:

```
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

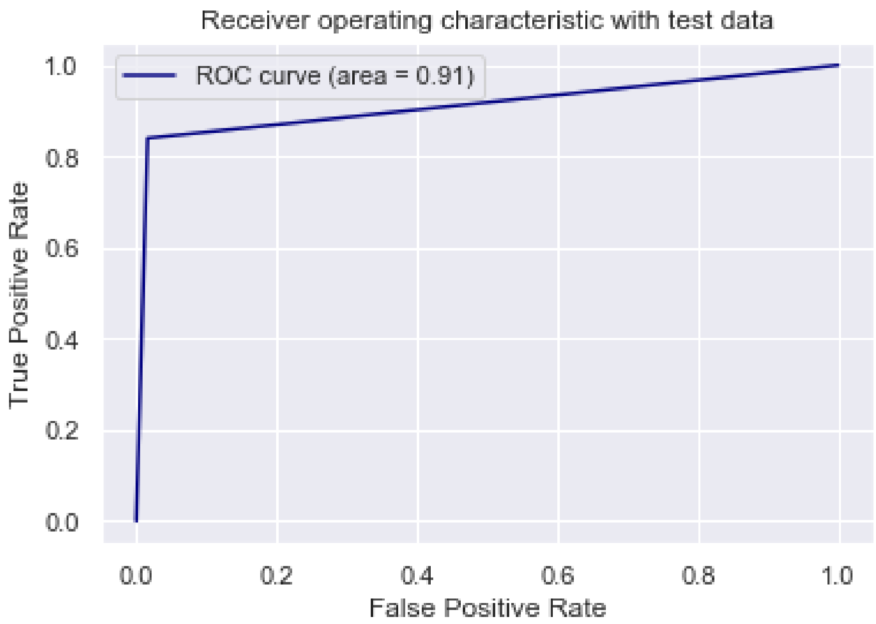


Test confusion_matrix



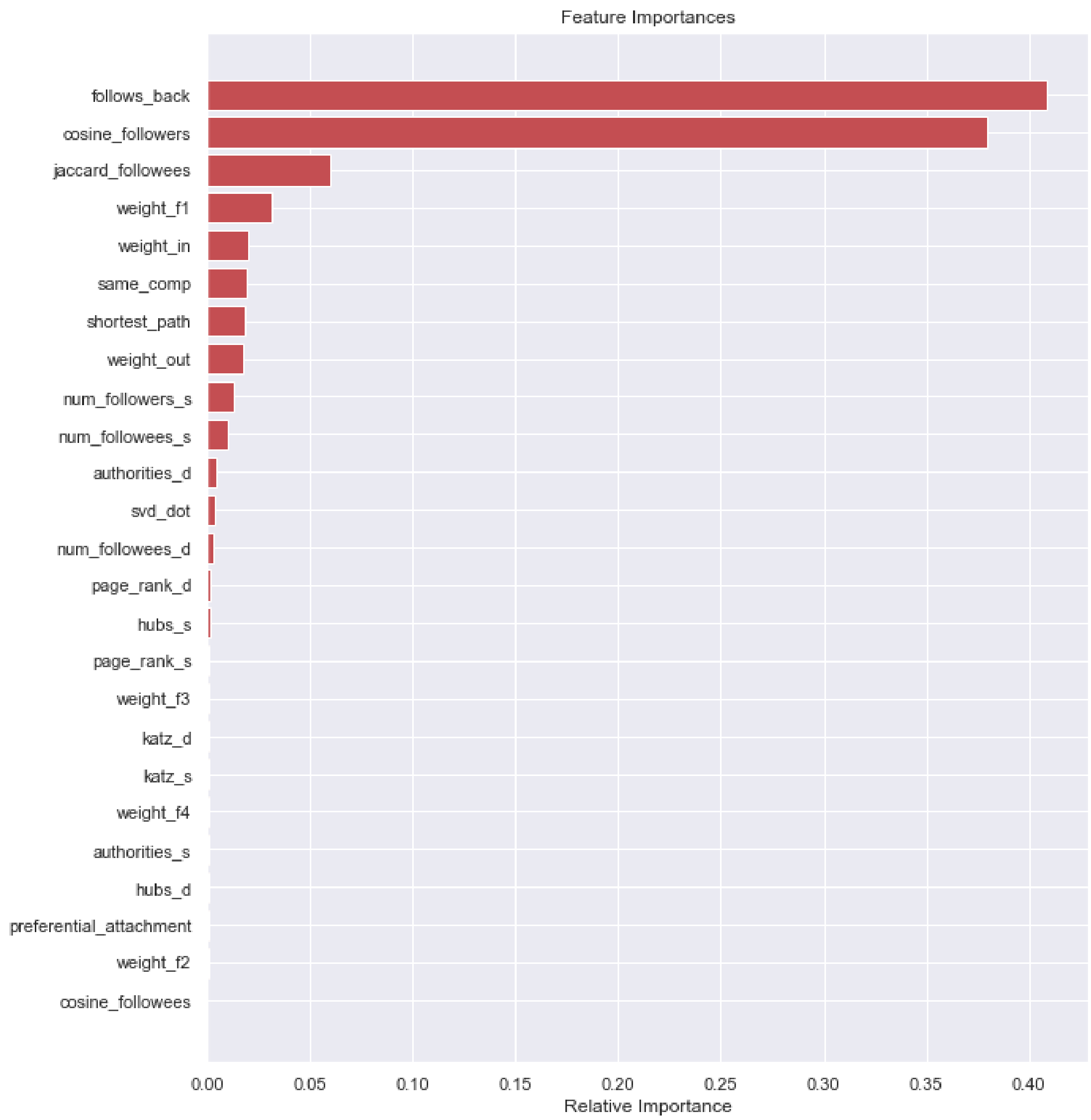
In [26]:

```
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 [27]:

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



In [29]:

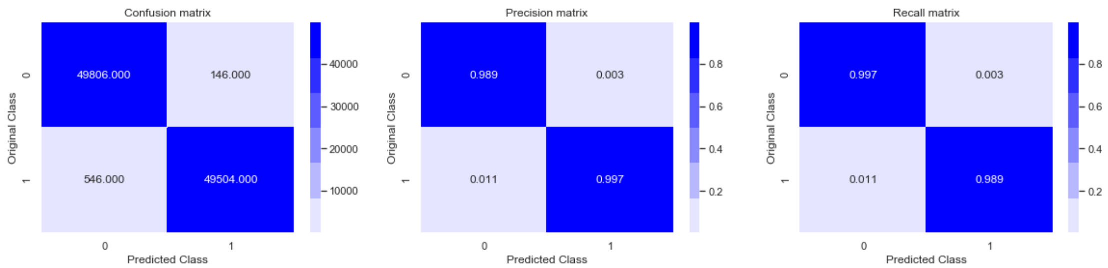
```
xgbdt = xgb.XGBClassifier(max_depth= 100, n_estimators=4)
xgbdt.fit(df_final_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability esti
# not the predicted outputs
y_train_pred = xgbdt.predict(df_final_train)
y_test_pred = xgbdt.predict(df_final_test)
print('Train f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))
```

```
Train f1 score 0.9930591775325978
Test f1 score 0.9320286416227453
```

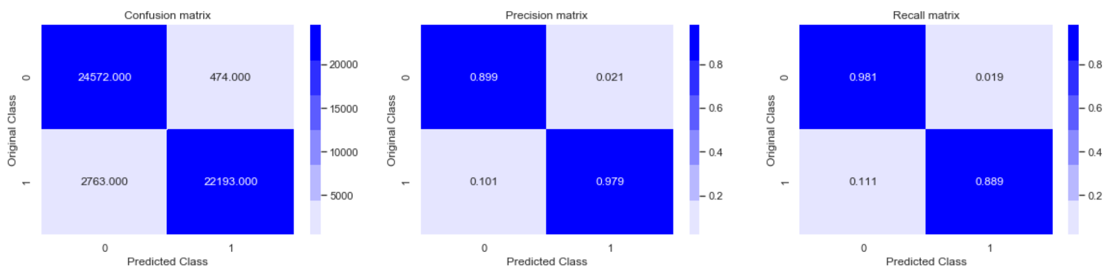
In [30]:

```
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
```

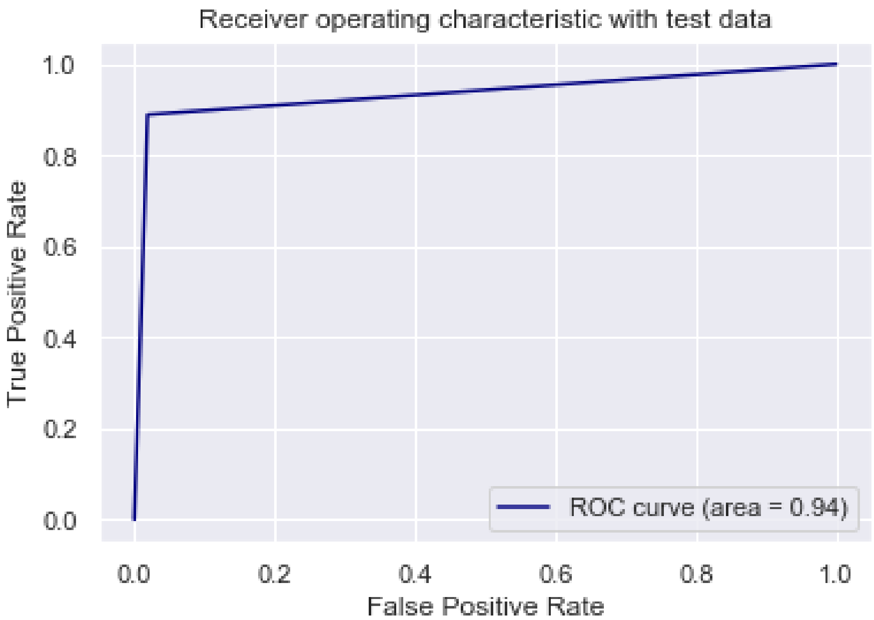



```
Test confusion_matrix
```



In [31]:

```
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()
```



Conclusion

In [33]:

```
# http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

table = PrettyTable()
table.field_names = ["Model", "Max Depth", "Estimators", "F1 Score"]

table.add_row(['Random Forest', 15, 900, 0.9285])
table.add_row(['Random Forest (RS)', 121, 14, 0.9286])
table.add_row(['GBDT (XGBoost)', 5, 200, 0.9051])
table.add_row(['GBDT (XGBoost)', 100, 4, 0.9320])
print(table)
```

```
+--------------------+-----------+------------+----------+
|       Model        | Max Depth | Estimators | F1 Score |
+--------------------+-----------+------------+----------+
|   Random Forest    |     15    |    900     |  0.9285  |
| Random Forest (RS) |    121    |     14     |  0.9286  |
|   GBDT (XGBoost)   |     5     |    200     |  0.9051  |
|   GBDT (XGBoost)   |    100    |     4      |  0.932   |
+--------------------+-----------+------------+----------+
```

Summary

- Added feature called Preferential Attachment and svd_dot.
- Trained Random forest and XG boost with all the features.
- Tuned hyperparameters for XG boost and Random Forest using GridSearchCV and RandomSearchCV.
- Plotted 3D plot, Heat map and other plots for understanding the model's behaviour
- XGBoost is the best model among the two
- Best Test f1 score obtained is 0.932