Assignments:

1. Add another feature called Preferential Attachment with followers and followees data of vertex. you can check about Preferential Attachment in below link http://be.amazd.com/link-prediction/
2. Add feature called svd_dot. you can calculate svd_dot as Dot product between sourse node svd and destination node svd features. you can read about this in below pdf https://storage.googleapis.com/kaggle-forum-message-attachments/2594/supervised_link_prediction.pdf
3. Tune hyperparameters for XG boost with all these features and check the error metric.

In [10]:

```
#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 sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint as sp_randint
from scipy.stats import uniform
from sklearn.metrics import confusion_matrix
```

In [27]:

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

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

1. Reading Data

In [28]:

```
if os.path.isfile('/content/gdrive/MyDrive/Facebook/data/after_eda/train_after_eda.csv'):
    train_graph=nx.read_edgelist('/content/gdrive/MyDrive/Facebook/data/after_eda/train_after_eda.csv'
```

```
    train_graph=nx.read_edgelist('/content/gdrive/MyDrive/facebook/data/after_eda/train_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: 1862196
Number of edges: 15100030
Average in degree:   8.1087
Average out degree:   8.1087
```

2. Similarity measures

2.1 Jaccard Distance:

http://www.statisticshowto.com/jaccard-index/

\begin{equation} j = \frac{|X\cap Y|}{|X \cup Y|} \end{equation}

In [29]:

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

In [30]:

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

```
0.0
```

In [31]:

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

```
0.0
```

In [32]:

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

In [33]:

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

0

In [34]:

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

0.0

2.2 Cosine distance

\begin{equation} CosineDistance = \frac{|X\cap Y|}{|X|\cdot|Y|} \end{equation}

In [35]:

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

In [36]:

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

0.0

In [37]:

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

0.0

In [38]:

```
def cosine_for_followers(a,b):
    try:

        if len(set(train_graph.predecessors(a))) == 0  | len(set(train_graph.predecessors(b))) == 0:
            return 0
        sim = (len(set(train_graph.predecessors(a)).intersection(set(train_graph.predecessors(b)))))/\
                                     (math.sqrt(len(set(train_graph.predecessors(a))))*(len(set(train_g
raph.predecessors(b)))))
        return sim
    except:
        return 0
```

In [39]:

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

0.02041241452319315

```
0.020412414523193152
```

In [40]:

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

```
0.0
```

3. Ranking Measures

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.**

3.1 Page Ranking

https://en.wikipedia.org/wiki/PageRank

In [41]:

```
if not os.path.isfile('/content/gdrive/MyDrive/case_study_2/data/page_rank.p'):
    pr = nx.pagerank(train_graph, alpha=0.85)
    pickle.dump(pr,open('/content/gdrive/MyDrive/case_study_2/data/page_rank.p','wb'))
else:
    pr = pickle.load(open('/content/gdrive/MyDrive/case_study_2/data/page_rank.p','rb'))
```

In [42]:

```
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 8.216658719151825e-08
max 1.2201861685116973e-05
mean 5.370004016764232e-07
```

In [43]:

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

```
5.370004016764232e-07
```

4. Other Graph Features

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 [44]:

```
#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 [45]:

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

Out[45]:

```
7
```

In [46]:

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

Out[46]:

```
8
```

4.2 Checking for same community

In [47]:

```
#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 [48]:

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

Out[48]:

1

In [49]:

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

Out[49]:

1

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 [50]:

```
#adar index
def calc_adar_in(a,b):
    sum=0
    try:
        n=list(set(train_graph.successors(a)).intersection(set(train_graph.successors(b))))
        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 [51]:

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

Out[51]:

0

In [52]:

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

Out[52]:

0

4.4 Is persion was following back:

In [53]:

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

In [54]:

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

Out[54]:

```
1
```

In [55]:

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

Out[55]:

```
0
```

4.5 Katz Centrality:

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

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 [56]:

```
if not os.path.isfile('/content/gdrive/MyDrive/case_study_2/data/katz.p'):
    katz = nx.katz.katz_centrality(train_graph,alpha=0.005,beta=1)
    pickle.dump(katz,open('/content/gdrive/MyDrive/case_study_2/data/katz.p','wb'))
else:
    katz = pickle.load(open('/content/gdrive/MyDrive/case_study_2/data/katz.p','rb'))
```

In [57]:

```
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 0.0007010497950790909
max 0.003321131818137773504
mean 0.0007318221196993401
```

In [58]:

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

```
0.0007318221196993401
```

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

In [59]:

In [59]:

```
if not os.path.isfile('/content/gdrive/MyDrive/case_study_2/data/hits.p'):
    hits = nx.hits(train_graph, max_iter=100, tol=1e-08, nstart=None, normalized=True)
    pickle.dump(hits,open('/content/gdrive/MyDrive/case_study_2/data/hits.p','wb'))
else:
    hits = pickle.load(open('/content/gdrive/MyDrive/case_study_2/data/hits.p','rb'))
```

In [60]:

```
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 0.004727078856184644
mean 5.370004016762928e-07
```

5. Featurization

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

In [61]:

```
import random
if os.path.isfile('/content/gdrive/MyDrive/Facebook/data/after_eda/train_after_eda.csv'):
    filename = "/content/gdrive/MyDrive/Facebook/data/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 (excludes header)
    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 [62]:

```
if os.path.isfile('/content/gdrive/MyDrive/Facebook/data/after_eda/test_after_eda.csv'):
    filename = "/content/gdrive/MyDrive/Facebook/data/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 (excludes header)
    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 [63]:

```
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_train))
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_test))
```

```
Number of rows in the train data file: 15100030
Number of rows we are going to elimiate in train data are 15000030
Number of rows in the test data file: 3775008
Number of rows we are going to elimiate in test data are 3725008
```

In [64]:

```
df_final_train = pd.read_csv('/content/gdrive/MyDrive/Facebook/data/after_eda/train_after_eda.csv', ski
prows=skip_train, names=['source_node', 'destination_node'])
df_final_train['indicator_link'] = pd.read_csv('/content/gdrive/MyDrive/Facebook/data/train_y.csv', ski
prows=skip_train, names=['indicator_link'])
print("Our train matrix size ",df_final_train.shape)
df_final_train.head(2)
```

```
Our train matrix size  (100001, 3)
```

Out[64]:

	source_node	destination_node	indicator_link
0	273084	1505602	1
1	1548737	1664354	1

In [65]:

```
df_final_test = pd.read_csv('/content/gdrive/MyDrive/Facebook/data/after_eda/test_after_eda.csv', skipr
ows=skip_test, names=['source_node', 'destination_node'])
df_final_test['indicator_link'] = pd.read_csv('/content/gdrive/MyDrive/Facebook/data/test_y.csv', skipr
ows=skip_test, names=['indicator_link'])
print("Our test matrix size ",df_final_test.shape)
df_final_test.head(2)
```

```
Our test matrix size  (50001, 3)
```

Out[65]:

	source_node	destination_node	indicator_link
0	848424	784690	1
1	354852	1311092	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 [66]:

```
if not os.path.isfile('/content/gdrive/MyDrive/case_study_2/data/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_node'],row['destination_n
ode']),axis=1)
    df_final_test['jaccard_followers'] = df_final_test.apply(lambda row:
                                            jaccard_for_followers(row['source_node'],row['destination_n
ode']),axis=1)

    #mapping jaccrd followees to train and test data
    df_final_train['jaccard_followees'] = df_final_train.apply(lambda row:
                                            jaccard_for_followees(row['source_node'],row['destination_n
ode']),axis=1)
    df_final_test['jaccard_followees'] = df_final_test.apply(lambda row:
                                            jaccard_for_followees(row['source_node'],row['destination_n
ode']),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_no
de']),axis=1)
    df_final_test['cosine_followers'] = df_final_test.apply(lambda row:
                                                    cosine_for_followers(row['source_node'],row['destination_no
de']),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_no
de']),axis=1)
    df_final_test['cosine_followees'] = df_final_test.apply(lambda row:
                                                    cosine_for_followees(row['source_node'],row['destination_no
de']),axis=1)
```

In [67]:

```
def compute_features_stage1(df_final):
    #calculating no of followers followees for source and destination
    #calculating intersection of followers and followees for source and destination
    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_d, inter_followers, inter_f
ollowees
```

In [68]:

```
if not os.path.isfile('/content/gdrive/MyDrive/case_study_2/data/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']= compute_features_stage1(df_fi
nal_train)

    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']= compute_features_stage1(df_fina
l_test)

    hdf = HDFStore('/content/gdrive/MyDrive/case_study_2/data/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('/content/gdrive/MyDrive/case_study_2/data/storage_sample_stage1.h5', 'tr
ain_df',mode='r')
    df_final_test = read_hdf('/content/gdrive/MyDrive/case_study_2/data/storage_sample_stage1.h5', 'tes
t_df',mode='r')
```

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 [69]:

```
if not os.path.isfile('/content/gdrive/MyDrive/case_study_2/data/storage_sample_stage2.h5'):
    #mapping adar index on train
    df_final_train['adar_index'] = df_final_train.apply(lambda row: calc_adar_in(row['source_node'],row
['destination_node']),axis=1)
    #mapping adar index on test
    df_final_test['adar_index'] = df_final_test.apply(lambda row: calc_adar_in(row['source_node'],row['
destination_node']),axis=1)

    #---------------------------------------------------------------------------------------------------
------
    #mapping followback or not on train
    df_final_train['follows_back'] = df_final_train.apply(lambda row: follows_back(row['source_node'],r
ow['destination_node']),axis=1)

    #mapping followback or not on test
    df_final_test['follows_back'] = df_final_test.apply(lambda row: follows_back(row['source_node'],row
['destination_node']),axis=1)

    #---------------------------------------------------------------------------------------------------
------
    #mapping same component of wcc or not on train
    df_final_train['same_comp'] = df_final_train.apply(lambda row: belongs_to_same_wcc(row['source_node
'],row['destination_node']),axis=1)

    ##mapping same component of wcc or not on train
    df_final_test['same_comp'] = df_final_test.apply(lambda row: belongs_to_same_wcc(row['source_node']
,row['destination_node']),axis=1)

    #---------------------------------------------------------------------------------------------------
------
    #mapping shortest path on train
    df_final_train['shortest_path'] = df_final_train.apply(lambda row: compute_shortest_path_length(row
['source_node'],row['destination_node']),axis=1)
    #mapping shortest path on test
    df_final_test['shortest_path'] = df_final_test.apply(lambda row: compute_shortest_path_length(row['
source_node'],row['destination_node']),axis=1)

    hdf = HDFStore('/content/gdrive/MyDrive/case_study_2/data/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('/content/gdrive/MyDrive/case_study_2/data/storage_sample_stage2.h5', 'tr
ain_df',mode='r')
    df_final_test = read_hdf('/content/gdrive/MyDrive/case_study_2/data/storage_sample_stage2.h5', 'tes
t_df',mode='r')
```

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

\begin{equation} W = \frac{1}{\sqrt{1+|X|}} \end{equation}

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

In [70]:

```
#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%|██████████| 1862196/1862196 [00:22<00:00, 84555.42it/s]
```

In [71]:

```
if not os.path.isfile('/content/gdrive/MyDrive/case_study_2/data/storage_sample_stage3.h5'):
    #mapping to pandas train
    df_final_train['weight_in'] = df_final_train.destination_node.apply(lambda x: Weight_in.get(x,mean_weight_in))
    df_final_train['weight_out'] = df_final_train.source_node.apply(lambda x: Weight_out.get(x,mean_weight_out))

    #mapping to pandas test
    df_final_test['weight_in'] = df_final_test.destination_node.apply(lambda x: Weight_in.get(x,mean_weight_in))
    df_final_test['weight_out'] = df_final_test.source_node.apply(lambda x: Weight_out.get(x,mean_weight_out))


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

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

In [72]:

```
if not os.path.isfile('/content/gdrive/MyDrive/case_study_2/data/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:pr.get(x,mean_pr))
    df_final_train['page_rank_d'] = df_final_train.destination_node.apply(lambda x:pr.get(x,mean_pr))

    df_final_test['page_rank_s'] = df_final_test.source_node.apply(lambda x:pr.get(x,mean_pr))
    df_final_test['page_rank_d'] = df_final_test.destination_node.apply(lambda x:pr.get(x,mean_pr))
    #================================================================================

    #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: katz.get(x,mean_katz))
    df_final_train['katz_d'] = df_final_train.destination_node.apply(lambda x: katz.get(x,mean_katz))

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

    #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: hits[0].get(x,0))
    df_final_train['hubs_d'] = df_final_train.destination_node.apply(lambda x: hits[0].get(x,0))

    df_final_test['hubs_s'] = df_final_test.source_node.apply(lambda x: hits[0].get(x,0))
    df_final_test['hubs_d'] = df_final_test.destination_node.apply(lambda x: hits[0].get(x,0))
    #================================================================================

    #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 x: hits[1].get(x,0))
    df_final_train['authorities_d'] = df_final_train.destination_node.apply(lambda x: hits[1].get(x,0))

    df_final_test['authorities_s'] = df_final_test.source_node.apply(lambda x: hits[1].get(x,0))
    df_final_test['authorities_d'] = df_final_test.destination_node.apply(lambda x: hits[1].get(x,0))
    #================================================================================

    hdf = HDFStore('/content/gdrive/MyDrive/case_study_2/data/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('/content/gdrive/MyDrive/case_study_2/data/storage_sample_stage3.h5', 'tr
ain_df',mode='r')
    df_final_test = read_hdf('/content/gdrive/MyDrive/case_study_2/data/storage_sample_stage3.h5', 'tes
t_df',mode='r')
```

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 [73]:

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

In [74]:

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

```
    except:
        return [0,0,0,0,0,0]
```

In [75]:

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

In [76]:

```
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 (1862196, 1862196)
U Shape (1862196, 6)
V Shape (6, 1862196)
s Shape (6,)
```

In [77]:

```
if not os.path.isfile('/content/gdrive/MyDrive/case_study_2/data/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_s_5', 'svd_u_s_6']] = \
    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_u_d_5','svd_u_d_6']] = \
    df_final_train.destination_node.apply(lambda x: svd(x, U)).apply(pd.Series)
    #==========================================================================================
=

    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',]] = \
    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_v_d_5','svd_v_d_6']] = \
    df_final_train.destination_node.apply(lambda x: svd(x, V.T)).apply(pd.Series)
    #==========================================================================================
=

    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']] = \
    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_d_5','svd_u_d_6']] = \
    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_s_5', 'svd_v_s_6',]] = \
    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_d_5','svd_v_d_6']] = \
    df_final_test.destination_node.apply(lambda x: svd(x, V.T)).apply(pd.Series)
    #==========================================================================================
=

    hdf = HDFStore('/content/gdrive/MyDrive/case_study_2/data/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()
else:
    df_final_train = read_hdf('/content/gdrive/MyDrive/case_study_2/data/storage_sample_stage4.h5', 'tr
ain_df',mode='r')
    df_final_test = read_hdf('/content/gdrive/MyDrive/case_study_2/data/storage_sample_stage4.h5', 'tes
t_df',mode='r')
```

preferential attachment:

preferential attachment:

Preferential Attachment with followers and followees data of vertex. you can check about Preferential Attachment in below link
http://be.amazd.com/link-prediction/

In [78]:

```
df_final_train.columns
```

Out[78]:

```
Index(['source_node', 'destination_node', 'indicator_link',
       'jaccard_followers', 'jaccard_followees', 'cosine_followers',
       'cosine_followees', 'num_followers_s', 'num_followers_d',
       'num_followees_s', 'num_followees_d', 'inter_followers',
       'inter_followees', 'adar_index', 'follows_back', 'same_comp',
       'shortest_path', 'weight_in', 'weight_out', 'weight_f1', 'weight_f2',
       'weight_f3', 'weight_f4', 'page_rank_s', 'page_rank_d', 'katz_s',
       'katz_d', 'hubs_s', 'hubs_d', 'authorities_s', 'authorities_d',
       'svd_u_s_1', 'svd_u_s_2', 'svd_u_s_3', 'svd_u_s_4', 'svd_u_s_5',
       'svd_u_s_6', 'svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3', 'svd_u_d_4',
       'svd_u_d_5', 'svd_u_d_6', 'svd_v_s_1', 'svd_v_s_2', 'svd_v_s_3',
       'svd_v_s_4', 'svd_v_s_5', 'svd_v_s_6', 'svd_v_d_1', 'svd_v_d_2',
       'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5', 'svd_v_d_6'],
      dtype='object')
```

In [79]:

```
def compute_features_stage5(df_final):
    #calculating the product of followers for source and destination
    #calculating the product of followees for source and destination
    num_followers_s=[]
    num_followees_s=[]
    num_followers_d=[]
    num_followees_d=[]
    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))

        followers_attachment = np.array(num_followers_s)*np.array(num_followers_d)
        followees_attachment = np.array(num_followees_s)*np.array(num_followees_d)



    return followers_attachment,followees_attachment
```

In [80]:

```
if not os.path.isfile('/content/gdrive/MyDrive/case_study_2/data/storage_sample_stage5.h5'):
    df_final_train['prefer_followers'], df_final_train['prefer_followees'] = compute_features_stage5(df
_final_train)

    df_final_test['prefer_followers'], df_final_test['prefer_followees']= compute_features_stage5(df_fi
nal_test)

    hdf = HDFStore('/content/gdrive/MyDrive/case_study_2/data/storage_sample_stage5.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('/content/gdrive/MyDrive/case_study_2/data/storage_sample_stage5.h5', 'tr
ain_df',mode='r')
    df_final_test = read_hdf('/content/gdrive/MyDrive/case_study_2/data/storage_sample_stage5.h5', 'tes
t_df',mode='r')
```

In [81]:

```
df_final_train.columns
```

Out[81]:

```
Index(['source_node', 'destination_node', 'indicator_link',
       'jaccard_followers', 'jaccard_followees', 'cosine_followers',
       'cosine_followees', 'num_followers_s', 'num_followers_d',
       'num_followees_s', 'num_followees_d', 'inter_followers',
       'inter_followees', 'adar_index', 'follows_back', 'same_comp',
       'shortest_path', 'weight_in', 'weight_out', 'weight_f1', 'weight_f2',
       'weight_f3', 'weight_f4', 'page_rank_s', 'page_rank_d', 'katz_s',
       'katz_d', 'hubs_s', 'hubs_d', 'authorities_s', 'authorities_d',
       'svd_u_s_1', 'svd_u_s_2', 'svd_u_s_3', 'svd_u_s_4', 'svd_u_s_5',
       'svd_u_s_6', 'svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3', 'svd_u_d_4',
       'svd_u_d_5', 'svd_u_d_6', 'svd_v_s_1', 'svd_v_s_2', 'svd_v_s_3',
       'svd_v_s_4', 'svd_v_s_5', 'svd_v_s_6', 'svd_v_d_1', 'svd_v_d_2',
       'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5', 'svd_v_d_6', 'prefer_followers',
       'prefer_followees'],
      dtype='object')
```

svd_dot

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

In [82]:

```
# function to calculate svd_dot product of source and destination features for matrix 'U' and 'V.T' sep
erately :
def compute_features_stage6(df_final,U):
  source_features      = df_final.source_node.apply(lambda x: svd(x, U)).apply(pd.Series)
  destination_features = df_final.destination_node.apply(lambda x: svd(x, U)).apply(pd.Series)

  # convert to numpy arrays to do dot product of source and destination features:
  source_features      = [row.to_numpy() for i,row in source_features.iterrows()]
  destination_features = [row.to_numpy() for i,row in destination_features.iterrows()]

  L = len(source_features)

  result_dot = []

  for i in range(L):
    result_dot.append(np.dot(source_features[i],destination_features[i]))

  result_dot = np.array(result_dot).reshape(-1,1)
  return result_dot
```

In [83]:

```
if not os.path.isfile('/content/gdrive/MyDrive/case_study_2/data/storage_sample_stage6.h5'):

    # svd_dot of source and destination in matrix 'U' in train
    df_final_train['svdot_u_sd'] = compute_features_stage6(df_final_train,U)
    # svd_dot of source and destination in matrix 'V.T' in train
    df_final_train['svdot_v_sd'] = compute_features_stage6(df_final_train,V.T)
    # svd_dot of source and destination in matrix 'U' in test
    df_final_test['svdot_u_sd'] =  compute_features_stage6(df_final_test,U)
    # svd_dot of source and destination in matrix 'V.T' in test
    df_final_test['svdot_v_sd'] = compute_features_stage6(df_final_test,V.T)
```

```
    hdf = HDFStore('/content/gdrive/MyDrive/case_study_2/data/storage_sample_stage6.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('/content/gdrive/MyDrive/case_study_2/data/storage_sample_stage6.h5', 'train_df',mode='r')
    df_final_test = read_hdf('/content/gdrive/MyDrive/case_study_2/data/storage_sample_stage6.h5', 'test_df',mode='r')
```

Training Model : xgboost

In [1]:

```
#reading
from pandas import read_hdf
df_final_train = read_hdf('/content/gdrive/MyDrive/case_study_2/data/storage_sample_stage6.h5', 'train_df',mode='r')
df_final_test = read_hdf('/content/gdrive/MyDrive/case_study_2/data/storage_sample_stage6.h5', 'test_df',mode='r')
```

In [37]:

```
df_final_train[['prefer_followers','prefer_followees','svdot_u_sd','svdot_v_sd']].head(2)
```

Out[37]:

	prefer_followers	prefer_followees	svdot_u_sd	svdot_v_sd
0	130	187	1.123026e-11	2.334976e-12
1	306	169	5.797253e-17	4.876798e-17

In [3]:

```
df_final_train.columns
```

Out[3]:

```
Index(['source_node', 'destination_node', 'indicator_link',
       'jaccard_followers', 'jaccard_followees', 'cosine_followers',
       'cosine_followees', 'num_followers_s', 'num_followers_d',
       'num_followees_s', 'num_followees_d', 'inter_followers',
       'inter_followees', 'adar_index', 'follows_back', 'same_comp',
       'shortest_path', 'weight_in', 'weight_out', 'weight_f1', 'weight_f2',
       'weight_f3', 'weight_f4', 'page_rank_s', 'page_rank_d', 'katz_s',
       'katz_d', 'hubs_s', 'hubs_d', 'authorities_s', 'authorities_d',
       'svd_u_s_1', 'svd_u_s_2', 'svd_u_s_3', 'svd_u_s_4', 'svd_u_s_5',
       'svd_u_s_6', 'svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3', 'svd_u_d_4',
       'svd_u_d_5', 'svd_u_d_6', 'svd_v_s_1', 'svd_v_s_2', 'svd_v_s_3',
       'svd_v_s_4', 'svd_v_s_5', 'svd_v_s_6', 'svd_v_d_1', 'svd_v_d_2',
       'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5', 'svd_v_d_6', 'prefer_followers',
       'prefer_followees', 'svdot_u_sd', 'svdot_v_sd'],
      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=1,inplace=True)
df_final_test.drop(['source_node', 'destination_node','indicator_link'],axis=1,inplace=True)
```

1. working on balanced sampled data

1. Working on balanced sampled data

In [6]:

```
y_train.value_counts()
```

Out[6]:

```
1    50012
0    49989
Name: indicator_link, dtype: int64
```

In [7]:

```
y_test.value_counts()
```

Out[7]:

```
0    25021
1    24980
Name: indicator_link, dtype: int64
```

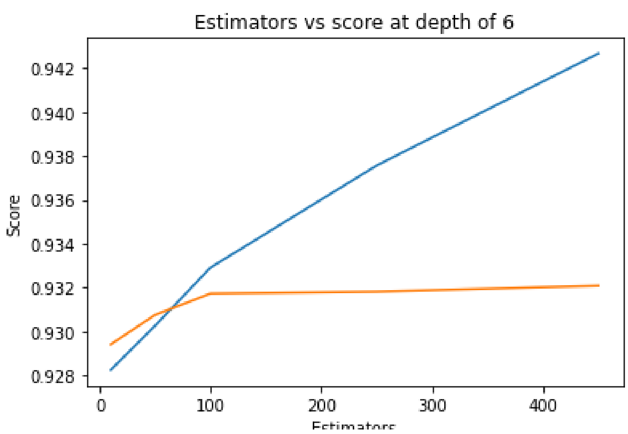
In [12]:

```
# to find best estimator
estimators = [10,50,100,250,450]
train_scores = []
test_scores = []
for i in estimators:
    clf = xgb.XGBClassifier(n_estimators=i,max_depth=6,learning_rate=0.1,verbosity=0,objective='binary:
logistic',booster='gbtree',n_jobs=-1,
                            min_child_weight=15,colsample_bytree=0.8,colsample_bylevel=0.8,random_state
=39)
    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 6')
```

```
Estimators =  10 Train Score 0.9282376749557644 test Score 0.9294025390500816
Estimators =  50 Train Score 0.9302547365744077 test Score 0.9307539093952621
Estimators =  100 Train Score 0.9329078209043546 test Score 0.9317263981393383
Estimators =  250 Train Score 0.937567855298247 test Score 0.9318105049822664
Estimators =  450 Train Score 0.9426742217970417 test Score 0.93208144319021
```

Out[12]:

```
Text(0.5, 1.0, 'Estimators vs score at depth of 6')
```

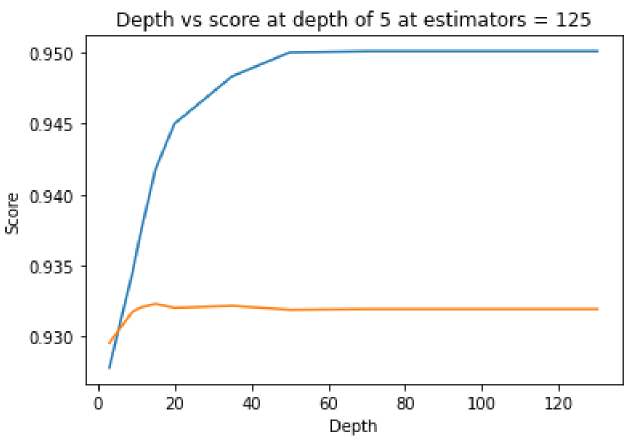


Estimators

In [13]:

```
# to find best depth
depths = [3,9,11,15,20,35,50,70,130]
train_scores = []
test_scores = []
for i in depths:
    clf = xgb.XGBClassifier(n_estimators=125,max_depth=i,learning_rate=0.05,verbosity=0,objective='bina
ry:logistic',booster='gbtree',n_jobs=-1,
                            min_child_weight=15,colsample_bytree=0.8,colsample_bylevel=0.8,random_state
=39)
    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 = 125')
plt.show()
```

```
depth =  3 Train Score 0.9278002125398513 test Score 0.9295451649891707
depth =  9 Train Score 0.9344705857484094 test Score 0.9317450914014895
depth =  11 Train Score 0.9370970294940115 test Score 0.932041709850039
depth =  15 Train Score 0.9417582995738413 test Score 0.9323158873763422
depth =  20 Train Score 0.9449949613705072 test Score 0.9320310352846664
depth =  35 Train Score 0.9483195476913412 test Score 0.932172590119596
depth =  50 Train Score 0.95000731804211 test Score 0.9318964606444796
depth =  70 Train Score 0.9500878220140516 test Score 0.9319415978194265
depth =  130 Train Score 0.9500878220140516 test Score 0.9319415978194265
```



In [14]:

```
# cross validation using RandomizedSearchCV to find best model parameters

param_dist = {"n_estimators":sp_randint(100,120),
              "max_depth": sp_randint(6,12),
              "min_child_weight": sp_randint(10,20)}

clf = xgb.XGBClassifier(random_state=39,n_jobs=-1)

xgb_random = RandomizedSearchCV(clf, param_distributions=param_dist,
                                   n_iter=5,cv=10,scoring='f1',random_state=39,return_train_score=True)

xgb_random.fit(df_final_train,y_train)
print('mean test scores',xgb_random.cv_results_['mean_test_score'])
print('mean train scores',xgb_random.cv_results_['mean_train_score'])
```

```
mean test scores [0.93034334 0.93021947 0.93016452 0.93041955 0.93045409]
mean train scores [0.93478639 0.93359079 0.93910283 0.93834854 0.93866433]
```

In [23]:

```
print(xgb_random.best_estimator_)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=10,
              min_child_weight=19, missing=None, n_estimators=100, n_jobs=-1,
              nthread=None, objective='binary:logistic', random_state=39,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

In [24]:

```
# best model
clf = xgb.XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=10,
              min_child_weight=19, missing=None, n_estimators=100, n_jobs=-1,
              nthread=None, objective='binary:logistic', random_state=39,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

In [25]:

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

In [26]:

```
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 0.937916934523496
Test f1 score 0.932071245959137
```

In [27]:

```
# function taken from assignment reference:

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
```

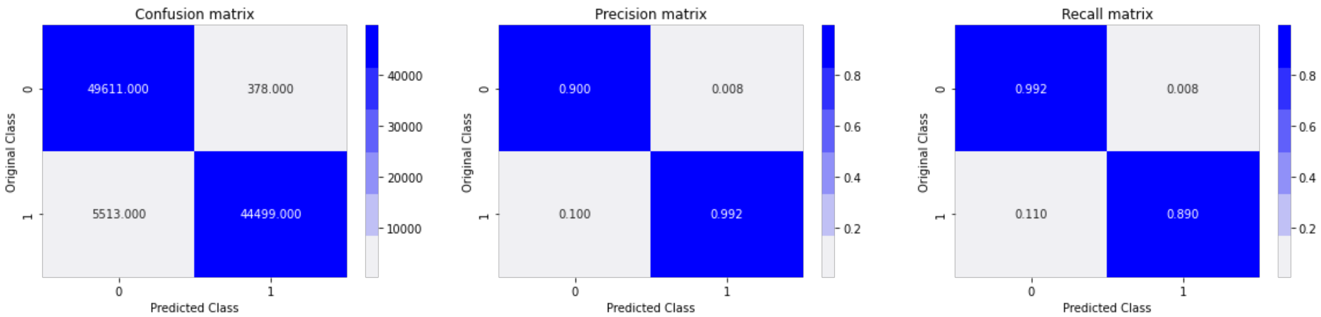
```
    sns.heatmap(A, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()
```

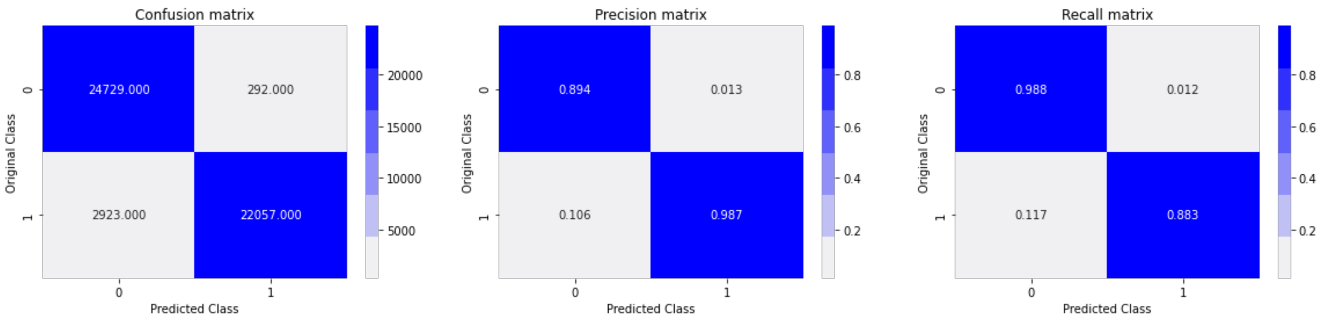
In [28]:

```
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



Observations:

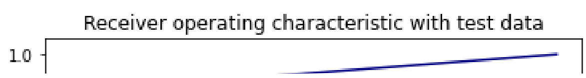
1. for both Train and Data data:

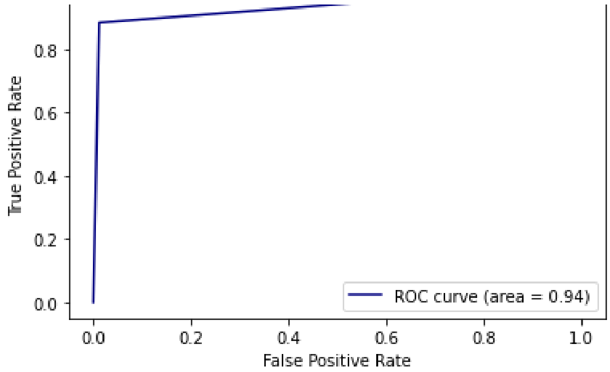
```
True positive is slightly high than true negative in precision
true negative is slightly high than true positive in recall
```

1. similary in confusion matrix True negative is slightly higher than True positive in both Train and Test data

In [29]:

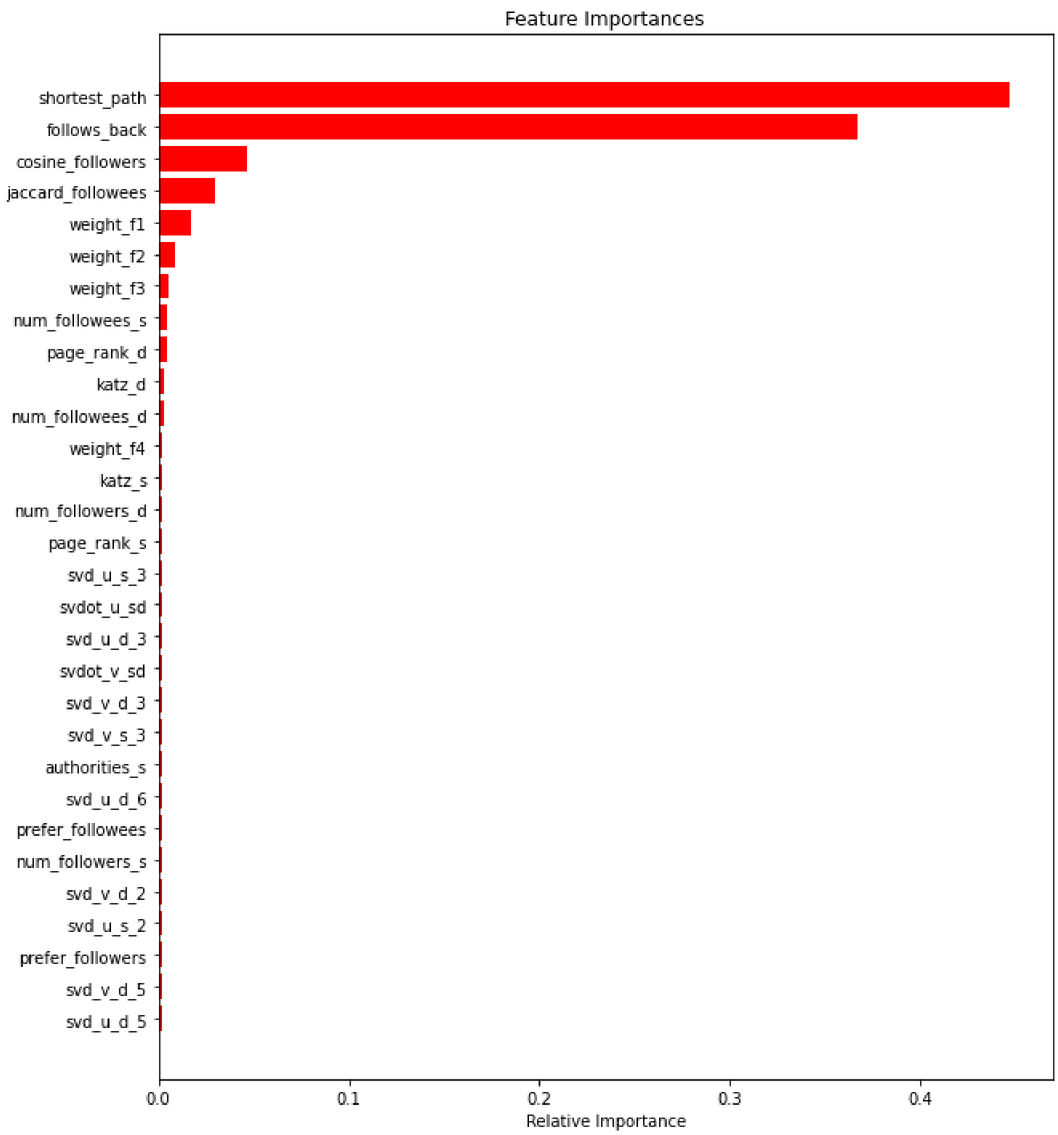
```
from sklearn.metrics import roc_curve, auc
fpr,tpr,ths = roc_curve(y_test,y_test_pred)
auc_sc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='navy',label='ROC curve (area = %0.2f)' % auc_sc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic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





In [38]:

```
# top 30 features
features = df_final_train.columns
importances = clf.feature_importances_
indices = (np.argsort(importances))[-30:]
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()
```



Observations:

1. on adding two new features (preferential attachment,svd_dot), there is a some improvement in both test f1_scores and test Auc score.
2. and F1 scores for both Train and Test and AUC score

```
            Train f1 score = 93%
            Test f1 score  = 93%
                AUC  score = 94%

      very close and overall best score without overfitting
```

3. we can see from the best features, in which our newly added features are also got some amount of feature importance (prefer_followees,prefer_followers,svd_dot_u_sd,svd_dot_v_sd)

In [ ]:

```

```