Social network Graph Link Prediction - Facebook Challenge

Problem statement:

Given a directed social graph, have to predict missing links to recommend users (Link Prediction in graph)

Data Overview

Taken data from facebook's recruting challenge on kaggle https://www.kaggle.com/c/FacebookRecruiting
data contains two columns source and destination eac edge in graph

```
- Data columns (total 2 columns):
- source_node         int64
- destination_node    int64
```

Mapping the problem into supervised learning problem:

- Generated training samples of good and bad links from given directed graph and for each link got some features like no of followers, is he followed back, page rank, katz score, adar index, some svd fetures of adj matrix, some weight features etc. and trained ml model based on these features to predict link.
- Some reference papers and videos :
  - https://www.cs.cornell.edu/home/kleinber/link-pred.pdf
  - https://www3.nd.edu/~dial/publications/lichtenwalter2010new.pdf
  - https://kaggle2.blob.core.windows.net/forum-message-attachments/2594/supervised_link_prediction.pdf
  - https://www.youtube.com/watch?v=2M77Hgy17cg

Business objectives and constraints:

- No low-latency requirement.
- Probability of prediction is useful to recommend ighest probability links

Performance metric for supervised learning:

- Both precision and recall is important so F1 score is good choice
- Confusion matrix

In [0]:

```
#Importing Libraries
# please do go through this python notebook:
import warnings
warnings.filterwarnings("ignore")

import csv
import pandas as pd#pandas to create small dataframes
import datetime #Convert to unix time
import time #Convert to unix time
# if numpy is not installed already : pip3 install numpy
import numpy as np#Do aritmetic operations on arrays
# matplotlib: used to plot graphs
import matplotlib
import matplotlib.pylab as plt
import seaborn as sns#Plots
from matplotlib import rcParams#Size of plots
from sklearn.cluster import MiniBatchKMeans, KMeans#Clustering
import math
import pickle
import os
# to install xgboost: pip3 install xgboost
import xgboost as xgb

import warnings
import networkx as nx
```

```
import networkx as nx
import pdb
import pickle
from sklearn.model_selection import GridSearchCV
```

In [0]:

```
traincsv=pd.read_csv("/content/drive/My Drive/train.csv")
```

Out[0]:

	source_node	destination_node

In [0]:

```
#reading graph
if not os.path.isfile('/content/drive/My Drive/train_woheader.csv'):
  traincsv=pd.read_csv('/content/drive/My Drive/train.csv')
  print(traincsv[traincsv.isna().any(1)])
  print(traincsv.info())
  print("Number of Duplicate Entries are :", sum(traincsv.duplicated()))
  traincsv.to_csv("/content/drive/My Drive/train_woheader.csv" , header = False , index = False)
  print("saved data into the file")
else:
  g = nx.read_edgelist("/content/drive/My Drive/train_woheader.csv", delimiter= ',',create_using= n
x.DiGraph() ,nodetype=int)
  print(nx.info(g))
```

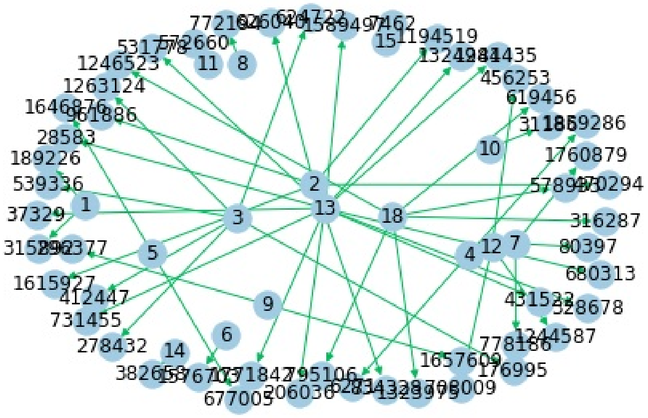
```
Empty DataFrame
Columns: [source_node, destination_node]
Index: []
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9437519 entries, 0 to 9437518
Data columns (total 2 columns):
 #   Column            Dtype
---  ------            -----
 0   source_node       int64
 1   destination_node  int64
dtypes: int64(2)
memory usage: 144.0 MB
None
Number of Duplicate Entries are : 0
saved data into the file
```

> Displaying a sub graph

In [0]:

```
if not os.path.isfile('/content/drive/My Drive/train_woheader_sample.csv'):
  pd.read_csv('/content/drive/My Drive/train_woheader.csv',nrows=50).to_csv("/content/drive/My
Drive/train_woheader_sample.csv",index=False,header=False)

subgraph=nx.read_edgelist("/content/drive/My
Drive/train_woheader_sample.csv",delimiter=',',create_using=nx.DiGraph(),nodetype=int)
pos=nx.spring_layout(subgraph)
nx.draw(subgraph,pos,node_color='#A0CBE2',edge_color='#00bb5e',width=1,edge_cmap=plt.cm.Blues,with_
labels=True)
plt.savefig("graph_sample.pdf")
print(nx.info(subgraph))
```

```
Name:
Type: DiGraph
Number of nodes: 66
Number of edges: 50
Average in degree:   0.7576
Average out degree:   0.7576
```



1. Exploratory Data Analysis

In [0]:

```
# graph creation
g = nx.read_edgelist("/content/drive/My Drive/train_woheader.csv", delimiter= ',',create_using= nx.
DiGraph() ,nodetype=int)
```

In [0]:

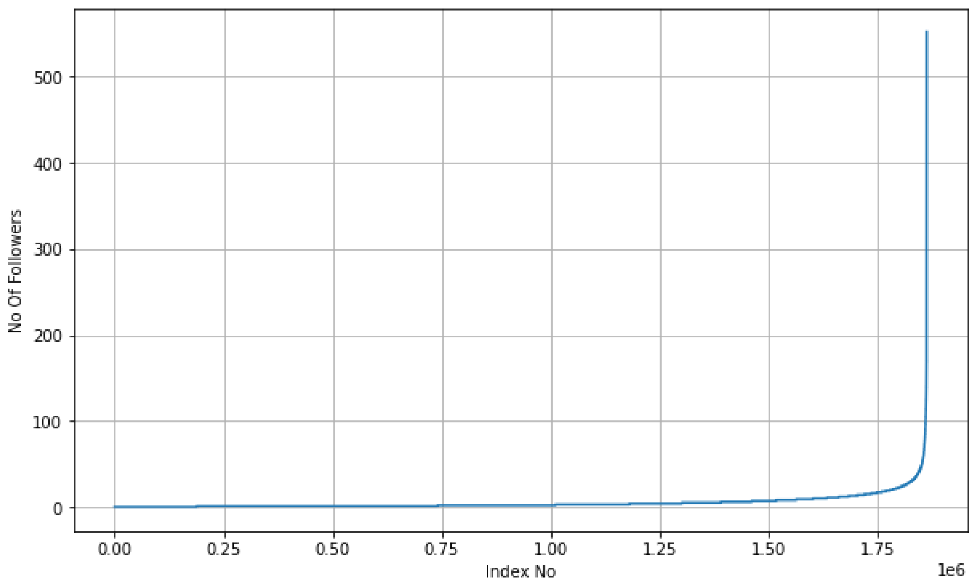
```
# No of Unique persons
print("The number of unique persons are:{count}".format(count = len(g.nodes())))
```

```
The number of unique persons are:1862220
```

1.1 No of followers for each person

In [0]:

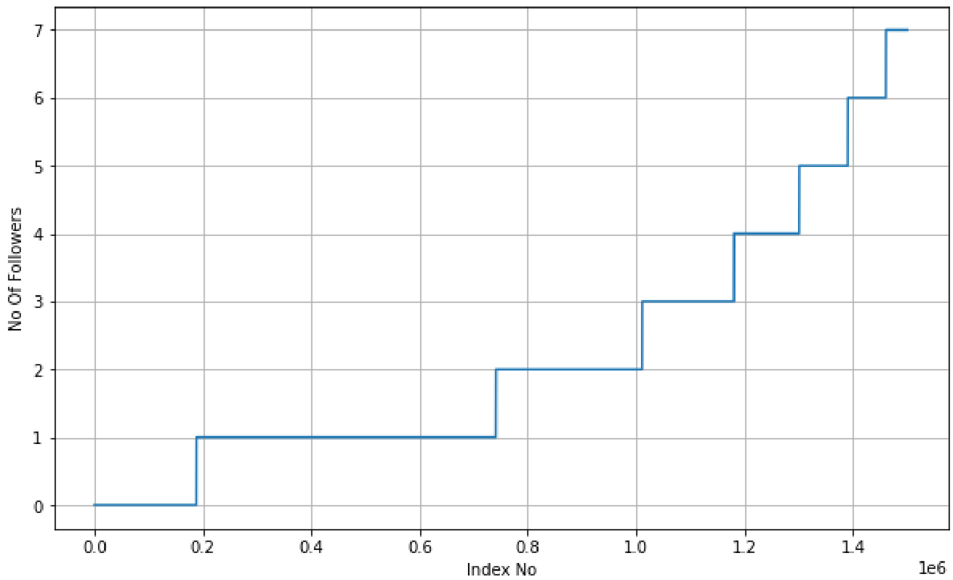
```
indegree_dist = list(dict(g.in_degree()).values())
indegree_dist.sort()
plt.figure(figsize=(10,6))
plt.plot(indegree_dist)
plt.xlabel('Index No')
plt.grid('box')
plt.ylabel('No Of Followers')
plt.show()
```



Most of the users have followers less than 50 but at last there are few users who have more than 50 followers

In [0]:

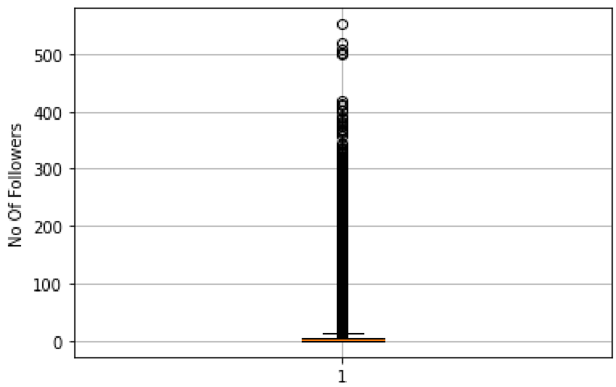
```
indegree_dist = list(dict(g.in_degree()).values())
indegree_dist.sort()
plt.figure(figsize=(10,6))
plt.plot(indegree_dist[0:1500000])
plt.xlabel('Index No')
plt.grid('box')
plt.ylabel('No Of Followers')
plt.show()
```



we can observe that until 1.5 M users have only atmost 7 followers and starting until 2 lakh followers have 0 followers

In [0]:

```
# boxplot
plt.boxplot(indegree_dist)
plt.ylabel('No Of Followers')
plt.grid('box')
plt.show()
```



By seeing the above box plot we cant notice 25,50,75 percentiles just by seeing and most of them have 0-10 followers just by looking in the graph,Rest of the portion up are outliers who has crazy numbe of followers

Lets check out some of the percentiles to get the sense of the number of followers:-

In [0]:

```
for i in range(90,101):
  print(i," Percentile of users have followers less than or equal to:" , np.percentile(indegree_dis
t , i))
```

```
90  Percentile of users have followers less than or equal to: 12.0
91  Percentile of users have followers less than or equal to: 13.0
```

```
92  Percentile of users have followers less than or equal to: 14.0
93  Percentile of users have followers less than or equal to: 15.0
94  Percentile of users have followers less than or equal to: 17.0
95  Percentile of users have followers less than or equal to: 19.0
96  Percentile of users have followers less than or equal to: 21.0
97  Percentile of users have followers less than or equal to: 24.0
98  Percentile of users have followers less than or equal to: 29.0
99  Percentile of users have followers less than or equal to: 40.0
100  Percentile of users have followers less than or equal to: 552.0
```

By seeing the above data we can notice that 90 percentile just have followers lessthan equal to 12 and there is one user with 552 followers lets look at 99-100 percentiles data

In [0]:

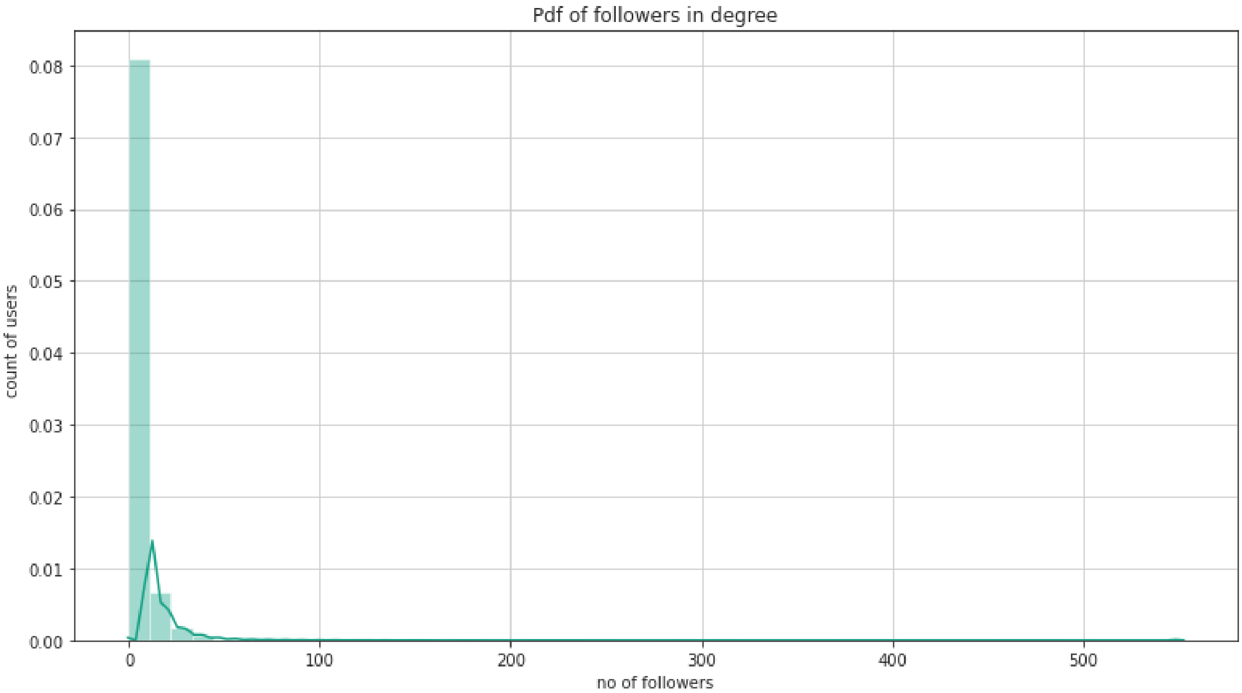
```
for i in range(1,10):
  print(99+(i/10)," Percentile of users have followers less than or equal to:" , np.percentile(inde
gree_dist , 99+(i/10)))
print(100,"  Percentile of users have followers less than or equal to:" , np.percentile(indegree_di
st ,100))
```

```
99.1  Percentile of users have followers less than or equal to: 42.0
99.2  Percentile of users have followers less than or equal to: 44.0
99.3  Percentile of users have followers less than or equal to: 47.0
99.4  Percentile of users have followers less than or equal to: 50.0
99.5  Percentile of users have followers less than or equal to: 55.0
99.6  Percentile of users have followers less than or equal to: 61.0
99.7  Percentile of users have followers less than or equal to: 70.0
99.8  Percentile of users have followers less than or equal to: 84.0
99.9  Percentile of users have followers less than or equal to: 112.0
100   Percentile of users have followers less than or equal to: 552.0
```

By seeing the above data we can notice that 99.1 - 99.9 percentile have followers lessthan equal to 112 and there is one of user with 552 followers

In [0]:

```
plt.figure(figsize=(13,7))
sns.distplot(indegree_dist, color='#16A085')
plt.title("Pdf of followers in degree")
plt.xlabel("no of followers")
plt.ylabel("count of users ")
plt.grid('box')
plt.show()
```

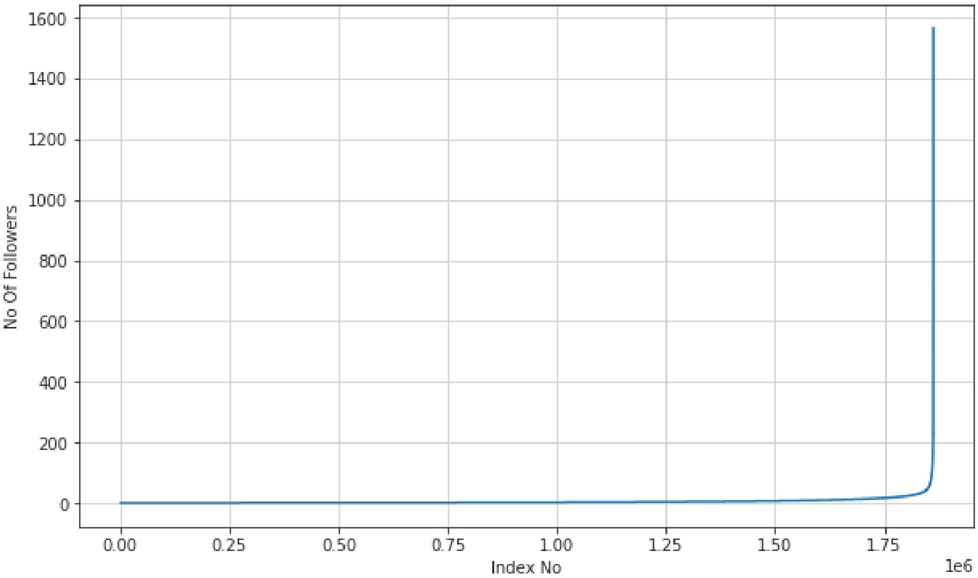


- By observing the pdf we can see that only 0.001 percent of the users has 500 plus followers

1.2 No of people each person is following

In [0]:

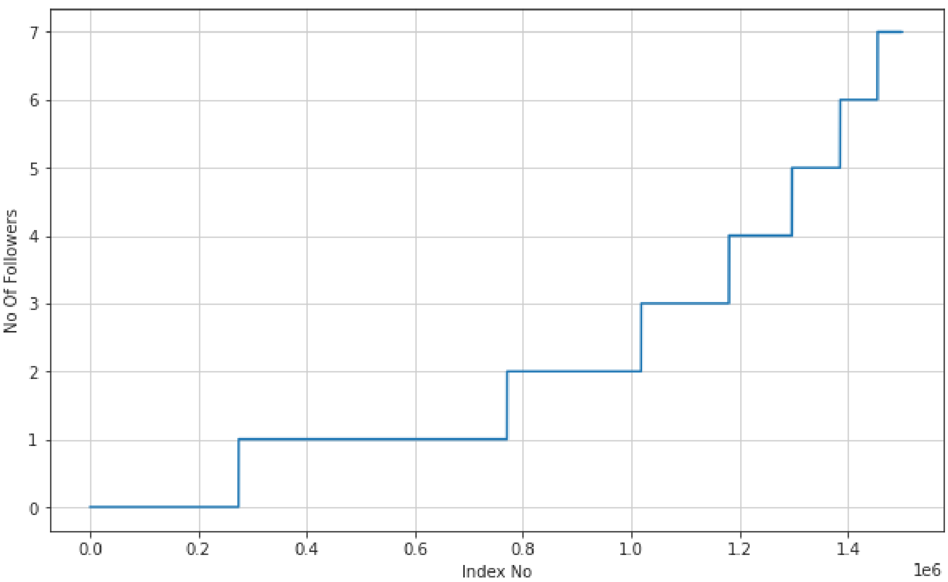
```
outdegree_dist = list(dict(g.out_degree()).values())
outdegree_dist.sort()
plt.figure(figsize=(10,6))
plt.plot(outdegree_dist)
plt.xlabel('Index No')
plt.grid('box')
plt.ylabel('No Of Followers')
plt.show()
```



Most of the 1.5m users are following less than 30 but at last there are few users who are following more than 50 followers

In [0]:

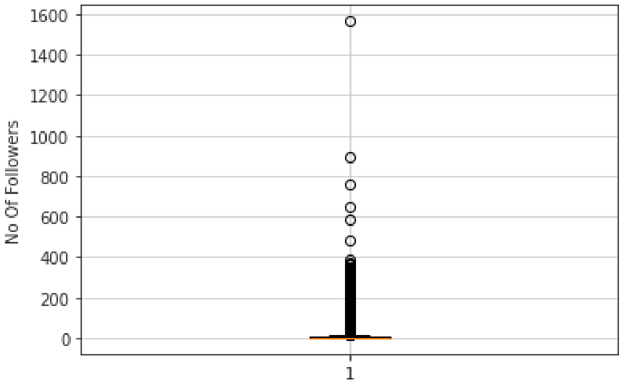
```
outdegree_dist = list(dict(g.out_degree()).values())
outdegree_dist.sort()
plt.figure(figsize=(10,6))
plt.plot(outdegree_dist[0:1500000])
plt.xlabel('Index No')
plt.grid('box')
plt.ylabel('No Of Followers')
plt.show()
```



we can observe that until 1.5 M users have only atmost 7 followers and starting 2 lakh users are following 0 users

In [0]:

```
# boxplot
plt.boxplot(outdegree_dist)
plt.ylabel('No Of Followers')
plt.grid('box')
plt.show()
```



By seeing the above box plot we can't notice 25,50,75 percentiles just by seeing and most of them are 0-1 followers just by looking in the graph,Rest of the portion up are outliers who are some following

Lets check out some of the percentiles to get the sense of the number following:-

In [0]:

```
for i in range(90,101):
  print(i," Percentile of users  following less than or equal to:" , np.percentile(outdegree_dist ,
i))
```

```
90  Percentile of users  following less than or equal to: 12.0
91  Percentile of users  following less than or equal to: 13.0
92  Percentile of users  following less than or equal to: 14.0
93  Percentile of users  following less than or equal to: 15.0
94  Percentile of users  following less than or equal to: 17.0
95  Percentile of users  following less than or equal to: 19.0
96  Percentile of users  following less than or equal to: 21.0
97  Percentile of users  following less than or equal to: 24.0
98  Percentile of users  following less than or equal to: 29.0
99  Percentile of users  following less than or equal to: 40.0
100  Percentile of users  following less than or equal to: 1566.0
```

By seeing the above data we can notice that 90 percentile users are following lessthan equal to 12 and there is one user with 1500+ followers lets look at 99-100 percentiles data

In [0]:

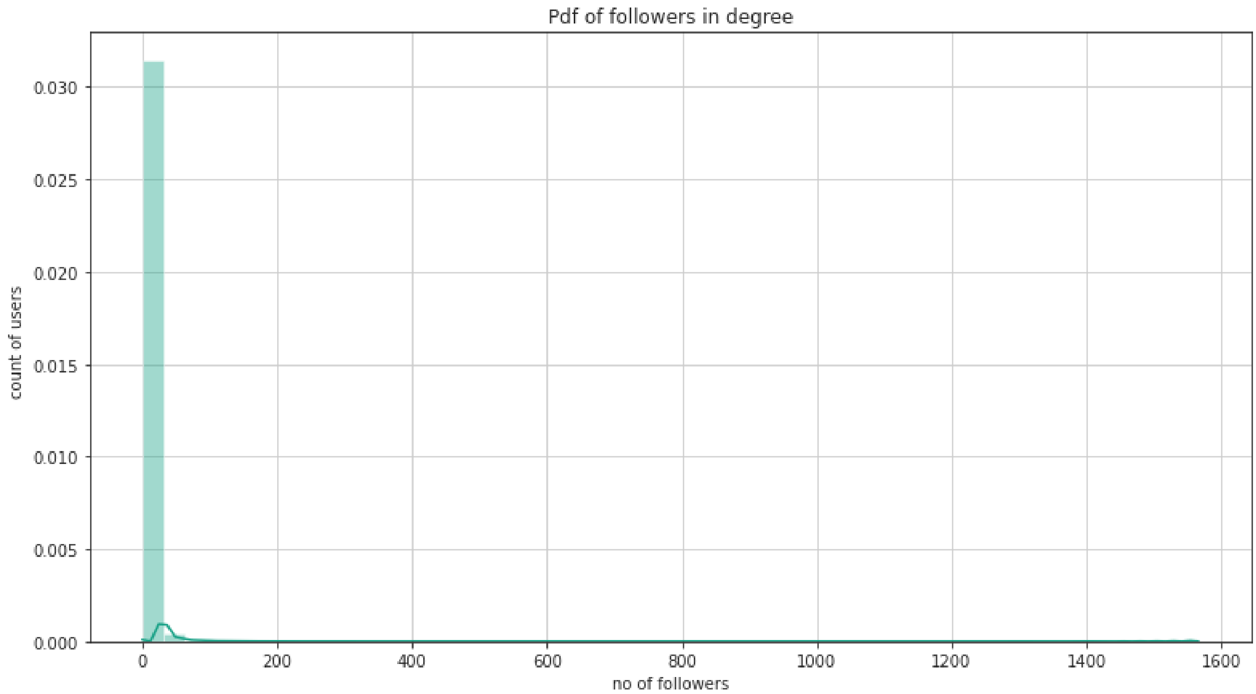
```
for i in range(1,10):
  print(99+(i/10)," Percentile of users have followers less than or equal to:" , np.percentile(outd
egree_dist , 99+(i/10)))
print(100,"  Percentile of users have followers less than or equal to:" , np.percentile(outdegree_d
ist ,100))
```

```
99.1  Percentile of users have followers less than or equal to: 42.0
99.2  Percentile of users have followers less than or equal to: 45.0
99.3  Percentile of users have followers less than or equal to: 48.0
99.4  Percentile of users have followers less than or equal to: 52.0
99.5  Percentile of users have followers less than or equal to: 56.0
99.6  Percentile of users have followers less than or equal to: 63.0
99.7  Percentile of users have followers less than or equal to: 73.0
99.8  Percentile of users have followers less than or equal to: 90.0
99.9  Percentile of users have followers less than or equal to: 123.0
100   Percentile of users have followers less than or equal to: 1566.0
```

By seeing the above data we can notice that 99.1 - 99.9 percentile are following lessthan equal to 123 and there is one user following 1566 users

In [0]:

```
plt.figure(figsize=(13,7))
sns.distplot(outdegree_dist, color='#16A085')
plt.title("Pdf of followers in degree")
plt.xlabel("no of followers")
plt.ylabel("count of users ")
plt.grid('box')
plt.show()
```



- By observing the pdf we can see that only 0.001 percent of the user is following 1500 plus users.

In [0]:

```
print('No of persons those are not following anyone are' ,sum(np.array(outdegree_dist)==0),'and % is',
                                sum(np.array(outdegree_dist)==0)*100/len(outdegree_dist) )
```

```
No of persons those are not following anyone are 274512 and % is 14.741115442858524
```

In [0]:

```
print('No of persons having zero followers are' ,sum(np.array(indegree_dist)==0),'and % is',
                                sum(np.array(indegree_dist)==0)*100/len(indegree_dist) )
```

```
No of persons having zero followers are 188043 and % is 10.097786512871734
```

1.3 both followers + following

In [0]:

```
count=0
for i in g.nodes():
    if len(list(g.predecessors(i)))==0 :
        if len(list(g.successors(i)))==0:
            count+=1
print('No of persons those are not not following anyone and also not having any followers are',count)
```

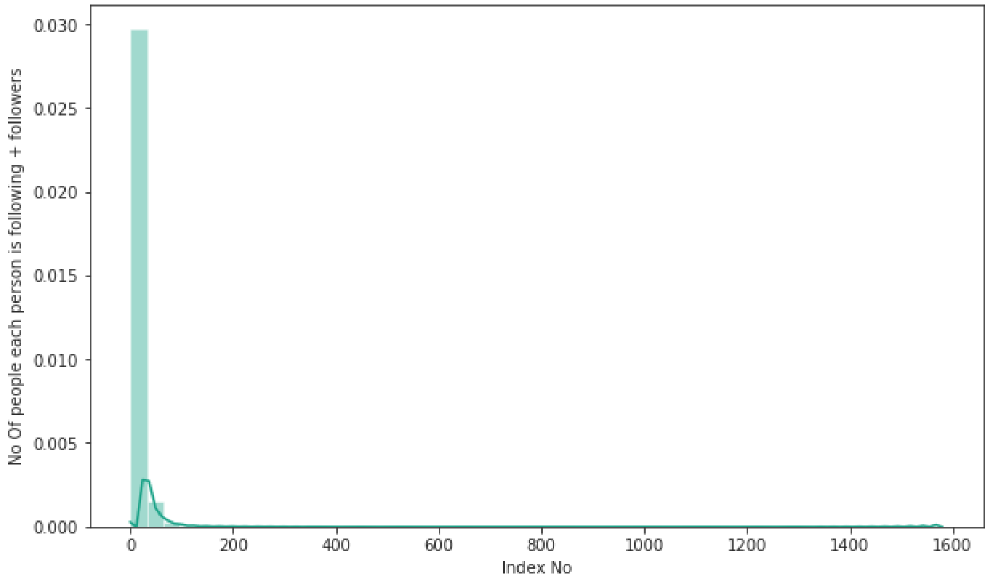
No of persons those are not not following anyone and also not having any followers are 0

In [0]:

```
from collections import Counter
dict_in = dict(g.in_degree())
dict_out = dict(g.out_degree())
d = Counter(dict_in) + Counter(dict_out)
in_out_degree = np.array(list(d.values()))
```

In [0]:

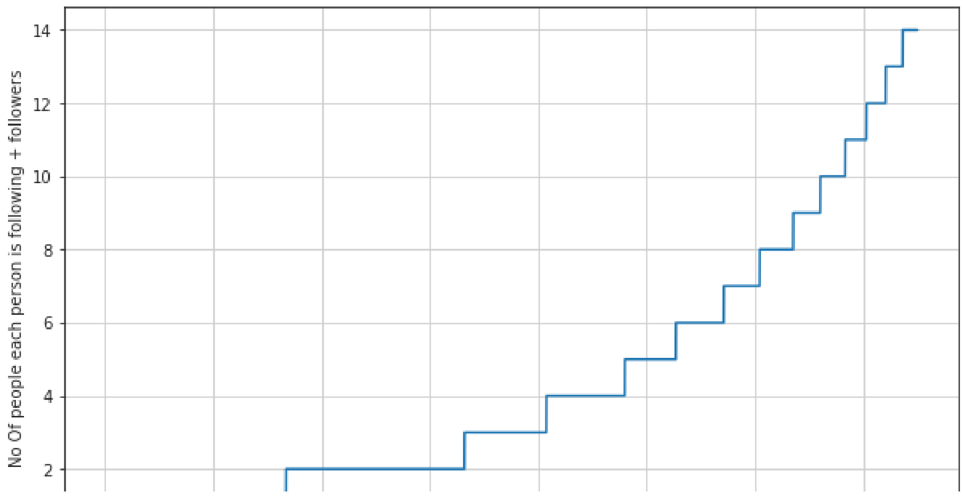
```
in_out_degree_sort = sorted(in_out_degree)
plt.figure(figsize=(10,6))
sns.distplot(in_out_degree_sort, color='#16A085')
plt.xlabel('Index No')
plt.ylabel('No Of people each person is following + followers')
plt.show()
```



By graph we can see that very few people are following n has less follwers less users and only one user is following and has followers more than 1500

In [0]:

```
in_out_degree_sort = sorted(in_out_degree)
plt.figure(figsize=(10,6))
plt.plot(in_out_degree_sort[0:1500000])
plt.xlabel('Index No')
plt.grid('box')
plt.ylabel('No Of people each person is following + followers')
plt.show()
```



By graph we can see that less than 1.4m people has 14 or less number of follwers and following

In [0]:

```
### 90-100 percentile
for i in range(0,11):
    print(90+i,'percentile value is',np.percentile(in_out_degree_sort,90+i))
```

```
90 percentile value is 24.0
91 percentile value is 26.0
92 percentile value is 28.0
93 percentile value is 31.0
94 percentile value is 33.0
95 percentile value is 37.0
96 percentile value is 41.0
97 percentile value is 48.0
98 percentile value is 58.0
99 percentile value is 79.0
100 percentile value is 1579.0
```

We can see that 99 percentile of the users are having and following less than or equal to 79 of the users

In [0]:

```
### 99-100 percentile
for i in range(10,110,10):
    print(99+(i/100),'percentile value is',np.percentile(in_out_degree_sort,99+(i/100)))
```

```
99.1 percentile value is 83.0
99.2 percentile value is 87.0
99.3 percentile value is 93.0
99.4 percentile value is 99.0
99.5 percentile value is 108.0
99.6 percentile value is 120.0
99.7 percentile value is 138.0
99.8 percentile value is 168.0
99.9 percentile value is 221.0
100.0 percentile value is 1579.0
```

We can see that only one user is having and following 1500 plus users

In [0]:

```
print('Min of no of followers + following is',in_out_degree.min())
print(np.sum(in_out_degree==in_out_degree.min()),' persons having minimum no of followers +
following')
```

```
Min of no of followers + following is 1
334291  persons having minimum no of followers + following
```

In [0]:

```
print('Max of no of followers + following is',in_out_degree.max())
print(np.sum(in_out_degree==in_out_degree.max()),' persons having maximum no of followers +
following')
```

```
Max of no of followers + following is 1579
1  persons having maximum no of followers + following
```

In [0]:

```
print('No of persons having followers + following less than 10 are',np.sum(in_out_degree<10))
```

```
No of persons having followers + following less than 10 are 1320326
```

No of persons having followers + following less than 10 are 1320326

In [0]:

```
print('No of weakly connected components',len(list(nx.weakly_connected_components(g))))
count=0
for i in list(nx.weakly_connected_components(g)):
    if len(i)==2:
        count+=1
print('weakly connected components with 2 nodes',count)
```

```
No of weakly connected components 45558
weakly connected components with 2 nodes 32195
```

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 [0]:

```
%%time
###generating bad edges from given graph
import random
if not os.path.isfile('/content/drive/My Drive/missing_edges_final.p'):
    #getting all set of edges
    r = csv.reader(open('/content/drive/My Drive/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('/content/drive/My Drive/missing_edges_final.p','wb'))
else:
    missing_edges = pickle.load(open('/content/drive/My Drive/missing_edges_final.p','rb'))
```

```
CPU times: user 5.09 s, sys: 812 ms, total: 5.91 s
Wall time: 6.07 s
```

In [0]:

```
len(missing_edges)
```

Out[0]:

```
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 [0]:

```
from sklearn.model_selection import train_test_split
if (not os.path.isfile('/content/drive/My Drive/train_pos_after_eda.csv')) and (not os.path.isfile(
'/content/drive/My Drive/test_pos_after_eda.csv')):
    #reading total data df
    df_pos = pd.read_csv('/content/drive/My Drive/train.csv')
    df_neg = pd.DataFrame(list(missing_edges), columns=['source_node', 'destination_node'])

    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 training data only for c
reating graph
    #and for feature generation
    X_train_pos, X_test_pos, y_train_pos, y_test_pos  = train_test_split(df_pos,np.ones(len(df_pos)
),test_size=0.2, random_state=9)
    X_train_neg, X_test_neg, y_train_neg, y_test_neg  = train_test_split(df_neg,np.zeros(len(df_neg
)),test_size=0.2, random_state=9)

    print('='*60)
    print("Number of nodes in the train data graph with edges", X_train_pos.shape[0],"=",y_train_po
s.shape[0])
    print("Number of nodes in the train data graph without edges", X_train_neg.shape[0],"=", y_trai
n_neg.shape[0])
    print('='*60)
    print("Number of nodes in the test data graph with edges", X_test_pos.shape[0],"=",y_test_pos.s
hape[0])
    print("Number of nodes in the test data graph without edges",
X_test_neg.shape[0],"=",y_test_neg.shape[0])

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

```
Number of nodes in the graph with edges 9437519
Number of nodes in the graph without edges 9437519
============================================================
Number of nodes in the train data graph with edges 7550015 = 7550015
Number of nodes in the train data graph without edges 7550015 = 7550015
============================================================
Number of nodes in the test data graph with edges 1887504 = 1887504
Number of nodes in the test data graph without edges 1887504 = 1887504
```

In [0]:

```
if (os.path.isfile('/content/drive/My Drive/train_pos_after_eda.csv')) and
(os.path.isfile('/content/drive/My Drive/test_pos_after_eda.csv')):
    train_graph=nx.read_edgelist('/content/drive/My
Drive/train_pos_after_eda.csv',delimiter=',',create_using=nx.DiGraph(),nodetype=int)
    test_graph=nx.read_edgelist('/content/drive/My
Drive/test_pos_after_eda.csv',delimiter=',',create_using=nx.DiGraph(),nodetype=int)
    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 data are {} %'.format(te
Y_trN/len(test_nodes_pos)*100))
```

```
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 --  717597
no of people present in test but not present in train --  81498
 % of people not there in Train but exist in Test in total Test data are 7.1200735962845405 %
```

> we have a cold start problem here

In [0]:

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

    X_train_pos = pd.read_csv('/content/drive/My Drive/train_pos_after_eda.csv', names=
['source_node', 'destination_node'])
    X_test_pos = pd.read_csv('/content/drive/My Drive/test_pos_after_eda.csv', names=
['source_node', 'destination_node'])
    X_train_neg = pd.read_csv('/content/drive/My Drive/train_neg_after_eda.csv', names=
['source_node', 'destination_node'])
    X_test_neg = pd.read_csv('/content/drive/My Drive/test_neg_after_eda.csv', names=
['source_node', 'destination_node'])

    print('='*60)
    print("Number of nodes in the train data graph with edges", X_train_pos.shape[0])
    print("Number of nodes in the train data graph without edges", X_train_neg.shape[0])
    print('='*60)
    print("Number of nodes in the test data graph with edges", X_test_pos.shape[0])
    print("Number of nodes in the test data graph without edges", X_test_neg.shape[0])

    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('/content/drive/My Drive/train_after_eda.csv',header=False,index=False)
    X_test.to_csv('/content/drive/My Drive/test_after_eda.csv',header=False,index=False)
    pd.DataFrame(y_train.astype(int)).to_csv('/content/drive/My
Drive/train_y.csv',header=False,index=False)
    pd.DataFrame(y_test.astype(int)).to_csv('/content/drive/My
Drive/test_y.csv',header=False,index=False)
```

```
The slowest run took 29.44 times longer than the fastest. This could mean that an intermediate res
ult is being cached.
1000 loops, best of 3: 242 µs per loop
```

In [0]:

```
X_train = pd.read_csv("/content/drive/My Drive/after_eda/train_after_eda.csv")
X_test = pd.read_csv("/content/drive/My Drive/after_eda/test_after_eda.csv")
y_train = pd.read_csv("/content/drive/My Drive/train_y.csv")
y_test = pd.read_csv("/content/drive/My Drive/test_y.csv")
```

In [0]:

```
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)
```

```
Data points in train data (15100029, 2)
Data points in test data (3775007, 2)
Shape of traget variable in train (15100029, 1)
Shape of traget variable in test (3775007, 1)
```

Feature Engineering

In [0]:

```
#creating train graph
train_graph=nx.read_edgelist('/content/drive/My
Drive/train_pos_after_eda.csv',delimiter=',',create_using=nx.DiGraph(),nodetype=int)
print(nx.info(train_graph))
```

```
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/

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

In [0]:

```
def jaccard_for_followees(a,b):

  try:
    if len(set(train_graph.successors(a))) == 0 | len(set(train_graph.successors(b))) == 0:
      return (0)

    else:
      result = len(set(train_graph.successors(a).intersection(set(train_graph.successors(b))))) /
\
      len(set(train_graph.successors(a).union(set(train_graph.successors(b)))))
      return (result)
  except:
    return (0)
```

In [0]:

```
print(jaccard_for_followees(273084,1505602))
```

```
0
```

In [0]:

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

0

In [0]:

```
def jaccard_for_followers(a,b):

  try:
    if len(set(train_graph.predecessors(a))) == 0 | len(set(train_graph.predecessors(b))) == 0:
      return (0)

    else:
      result = len(set(train_graph.predecessors(a).intersection(set(train_graph.predecessors(b)))))
/  \
      len(set(train_graph.predecessors(a).union(set(train_graph.predecessors(b)))))

  except:
    return (0)

  return (result)
```

In [0]:

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

0

In [0]:

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

0

2.2 Cosine distance

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

In [0]:

```
#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)
        result = (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 (result)
    except:
        return (0)
```

In [0]:

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

0

In [0]:

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

0.0

In [0]:

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

        if len(set(train_graph.predecessors(a))) == 0  | len(set(train_graph.predecessors(b))) == 0
:
            return (0)
        result = (len(set(train_graph.predecessors(a)).intersection(set(train_graph.predecessors(b)
))))/\
                                     (math.sqrt(len(set(train_graph.predecessors(a))))*(len(set(tra
n_graph.predecessors(b)))))
        return (result)
    except:
        return (0)
```

In [0]:

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

```
0.02886751345948129
```

In [0]:

```
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

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 [0]:

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

In [0]:

```
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 [0]:

```
#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

In [0]:

```
def shrtpath(a,b):
  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)
```

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 [0]:

```
#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 [0]:

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

Out[0]:

10

In [0]:

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

Out[0]:

-1

4.2 Checking for same community

In [0]:

```
#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 [0]:

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

Out[0]:

0

In [0]:

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

Out[0]:

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 [0]:

```
#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 [0]:

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

Out[0]:

0

In [0]:

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

Out[0]:

0

4.4 Is persion was following back:

In [0]:

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

In [0]:

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

Out[0]:

1

In [0]:

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

Out[0]:

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 [0]:

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

In [0]:

```
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 [0]:

```python
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

In [0]:

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

In [0]:

```python
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
```

5. Featurization

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

In [0]:

```python
import random
if os.path.isfile('/content/drive/My Drive/after_eda/train_after_eda.csv'):
    filename = "/content/drive/My Drive/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)
    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 [0]:

```python
if os.path.isfile('/content/drive/My Drive/after_eda/test_after_eda.csv'):
    filename = "/content/drive/My Drive/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)
    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 [0]:

In [0]:

```
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: 15100028
Number of rows we are going to elimiate in train data are 15000028
Number of rows in the test data file: 3775006
Number of rows we are going to elimiate in test data are 3725006
```

In [0]:

```
df_final_train = pd.read_csv('/content/drive/My Drive/after_eda/train_after_eda.csv',
skiprows=skip_train, names=['source_node', 'destination_node'])
df_final_train['indicator_link'] = pd.read_csv('/content/drive/My Drive/train_y.csv', skiprows=skip
_train, names=['indicator_link'])
print("Our train matrix size ",df_final_train.shape)
df_final_train.head(2)
```

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

Out[0]:

	source_node	destination_node	indicator_link
0	273084	1505602	1
1	1613640	1313162	1

In [0]:

```
df_final_test = pd.read_csv('/content/drive/My Drive/after_eda/test_after_eda.csv',
skiprows=skip_test, names=['source_node', 'destination_node'])
df_final_test['indicator_link'] = pd.read_csv('/content/drive/My Drive/test_y.csv', skiprows=skip_t
est, names=['indicator_link'])
print("Our test matrix size ",df_final_test.shape)
df_final_test.head(2)
```

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

Out[0]:

	source_node	destination_node	indicator_link
0	848424	784690	1
1	1790470	571834	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 [0]:

```
if not os.path.isfile('/content/drive/My Drive/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_node']),axis=1)
    df_final_test['jaccard_followers'] = df_final_test.apply(lambda row:

jaccard_for_followers(row['source_node'],row['destination_node']),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_node']),axis=1)
    df_final_test['jaccard_followees'] = df_final_test.apply(lambda row:

jaccard_for_followees(row['source_node'],row['destination_node']),axis=1)


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

cosine_for_followers(row['source_node'],row['destination_node']),axis=1)
    df_final_test['cosine_followers'] = df_final_test.apply(lambda row:

cosine_for_followers(row['source_node'],row['destination_node']),axis=1)

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

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

cosine_for_followees(row['source_node'],row['destination_node']),axis=1)
```

In [0]:

```
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, int
er_followees
```

In [0]:

```
if not os.path.isfile('/content/drive/My Drive/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(d
```

```
    df_final_train['inter_followers'], df_final_train['inter_followees']= compute_features_stage1(d
f_final_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_final_test)

    hdf = pd.HDFStore('/content/drive/My Drive/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/drive/My Drive/storage_sample_stage1.h5', 'train_df',mode='
r')
    df_final_test = read_hdf('/content/drive/My Drive/storage_sample_stage1.h5',
'test_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 [0]:

```
if not os.path.isfile('/content/drive/My Drive/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'],r
ow['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'],row['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_no
de'],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(r
ow['source_node'],row['destination_node']),axis=1)

    hdf = pd.HDFStore('/content/drive/My Drive/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/drive/My Drive/storage_sample_stage2.h5', 'train_df',mode='
r')
    df_final_test = read_hdf('/content/drive/My Drive/storage_sample_stage2.h5',
'test_df',mode='r')
```

```
test_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 [0]:

```
#weight for source and destination of each link
Weight_in = {}
Weight_out = {}
for i in  (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()))
```

In [0]:

```
if not os.path.isfile('/content/drive/My Drive/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 [0]:

```
if not os.path.isfile('/content/drive/My Drive/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 = pd.HDFStore('/content/drive/My Drive/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('data/fea_sample/storage_sample_stage3.h5', 'train_df',mode='r')
    df_final_test = read_hdf('data/fea_sample/storage_sample_stage3.h5', 'test_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 [0]:

```
def svd(x, S):
```

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

In [0]:

```python
#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 [0]:

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

In [0]:

```python
U, s, V = svd(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 [0]:

```python
if not os.path.isfile('/content/drive/My Drive/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 = pd.HDFStore('/content/drive/My Drive/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 [0]:

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

In [0]:

```
df_final_train.shape
```

Out[0]:

```
(100002, 54)
```

In [0]:

```
df_final_test.shape
```

Out[0]:

```
(50002, 54)
```

5.7 Adding new feature 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.

In [0]:

```
#function for getting the successors of user

def get_successors_train(data):
  """
  This Function is used to get the followers of each node
  """
  out_followers = len(set(train.successors(data)))
  return (out_followers)
```

In [0]:

```
#Applying the function to the data set column
df_final_train['successors_train_source_node'] = df_final_train['source_node'].apply(get_successors
_train)
df_final_train['successors_train_dest_node'] = df_final_train['destination_node'].apply(get_success
ors_train)

df_final_train['preferential_score(Ui,Uj)'] =
(df_final_train['successors_train_source_node'])*(df_final_train['successors_train_dest_node'])
```

In [0]:

```
def get_predecessors_train(data):
  """
  This Function is used to get the followers of each node
  """
  in_followers = len(set(train.predecessors(data)))
  return (in_followers)
```

In [0]:

```
#Applying the function to the data set column
df_final_train['pred_train_source_node'] =
df_final_train['source_node'].apply(get_predecessors_train)
df_final_train['pred_train_dest_node'] =
df_final_train['destination_node'].apply(get_predecessors_train)

df_final_train['preferential_score(Ui,Uj)pred'] = (df_final_train['pred_train_source_node'])*(df_fi
nal_train['pred_train_dest_node'])
```

In [0]:

```
#function for getting the successors of user
def get_successors_test(data):
  """
  This Function is used to get the followers of each node
  """
  out_followers = len(set(test.successors(data)))
  return (out_followers)
```

In [0]:

```
#Applying the function to the data set column
df_final_test['successors_test_source_node'] =
df_final_test['source_node'].apply(get_successors_test)
df_final_test['successors_test_dest_node'] =
df_final_test['destination_node'].apply(get_successors_test)

df_final_test['preferential_score(Ui,Uj)'] = (df_final_test['successors_test_source_node'])
*(df_final_test['successors_test_dest_node'])
```

In [0]:

```
def get_predecessors_test(data):
  """
  This Function is used to get the followers of each node
  """
  in_followers = len(set(test.predecessors(data)))
  return (in_followers)
```

In [0]:

```
#Applying the function to the data set column
df_final_test['pred_test_source_node'] = df_final_test['source_node'].apply(get_predecessors_test)
df_final_test['pred_test_dest_node'] =
df_final_test['destination_node'].apply(get_predecessors_test)

df_final_test['preferential_score(Ui,Uj)pred'] = (df_final_test['pred_test_source_node'])
*(df_final_test['pred_test_dest_node'])
```

In [0]:

```
df_final_test=df_final_test.drop(columns=['pred_test_source_node',
       'pred_test_dest_node'])
df_final_train=df_final_train.drop(columns=['pred_train_source_node',
       'pred_train_dest_node'])
```

In [38]:

```
df_final_test.columns
```

Out[38]:

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

5.8 Adding new Feature svd_dot

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 [0]:

```
#Performing the svd Dot Product for train data set
df_final_train["svd_dot_u"] =((df_final_train['svd_u_s_1']*df_final_train['svd_u_d_1']) +
                             (df_final_train['svd_u_s_2']*df_final_train['svd_u_d_2']) +
                             (df_final_train['svd_u_s_3']*df_final_train['svd_u_d_3']) +
                             (df_final_train['svd_u_s_4']*df_final_train['svd_u_d_4']) +
                             (df_final_train['svd_u_s_5']*df_final_train['svd_u_d_5']) +
                             (df_final_train['svd_u_s_6']*df_final_train['svd_u_d_6'])
)
```

In [0]:

```
df_final_train["svd_dot_v"] =((df_final_train['svd_v_s_1']*df_final_train['svd_v_d_1']) +
                             (df_final_train['svd_v_s_2']*df_final_train['svd_v_d_2']) +
                             (df_final_train['svd_v_s_3']*df_final_train['svd_v_d_3']) +
                             (df_final_train['svd_v_s_4']*df_final_train['svd_v_d_4']) +
                             (df_final_train['svd_v_s_5']*df_final_train['svd_v_d_5']) +
                             (df_final_train['svd_v_s_6']*df_final_train['svd_v_d_6']))
```

In [0]:

```
#Performing the svd Dot Product for test data set
df_final_test["svd_dot_u"] =((df_final_test['svd_u_s_1']*df_final_test['svd_u_d_1']) +
                            (df_final_test['svd_u_s_2']*df_final_test['svd_u_d_2']) +
                            (df_final_test['svd_u_s_3']*df_final_test['svd_u_d_3']) +
                            (df_final_test['svd_u_s_4']*df_final_test['svd_u_d_4']) +
                            (df_final_test['svd_u_s_5']*df_final_test['svd_u_d_5']) +
                            (df_final_test['svd_u_s_6']*df_final_test['svd_u_d_6']))
```

In [0]:

```
df_final_test["svd_dot_v"] =((df_final_test['svd_v_s_1']*df_final_test['svd_v_d_1']) +
                            (df_final_test['svd_v_s_2']*df_final_test['svd_v_d_2']) +
                            (df_final_test['svd_v_s_3']*df_final_test['svd_v_d_3']) +
                            (df_final_test['svd_v_s_4']*df_final_test['svd_v_d_4']) +
                            (df_final_test['svd_v_s_5']*df_final_test['svd_v_d_5']) +
                            (df_final_test['svd_v_s_6']*df_final_test['svd_v_d_6']) )
```

In [39]:

```
df_final_train.columns
```

Out[39]:

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

In [0]:

```
#Saving the final Dataframes
if not os.path.isfile("/content/drive/My Drive/df_final_test.csv"):
  df_final_test.to_csv("/content/drive/My Drive/df_final_test.csv")
else:
  df_final_test = pd.read_csv("/content/drive/My Drive/df_final_test.csv")
```

In [0]:

```
#Saving the final Dataframes
if not os.path.isfile("/content/drive/My Drive/df_final_train.csv"):
  df_final_test.to_csv("/content/drive/My Drive/df_final_train.csv")
else:
  df_final_test = pd.read_csv("/content/drive/My Drive/df_final_train.csv")
```

6.Modelling

6.1 Random forest model

In [0]:

```
df_final_test=pd.read_csv("/content/drive/My Drive/df_final_test.csv")
df_final_test=df_final_test.drop(columns='Unnamed: 0',axis=1)
```

In [0]:

```
df_final_train=pd.read_csv("/content/drive/My Drive/df_final_train.csv")
df_final_train=df_final_train.drop(columns='Unnamed: 0',axis=1)
```

In [0]:

```
df_train.to_csv("/content/drive/My Drive/df_train.csv",index=False,header=False)
```

In [0]:

```
df_test.to_csv("/content/drive/My Drive/df_test.csv",index=False,header=False)
```

In [0]:

```
train=nx.read_edgelist("/content/drive/My
Drive/df_train.csv",delimiter=',',create_using=nx.DiGraph(),nodetype=int)
test = nx.read_edgelist("/content/drive/My
Drive/df_test.csv",delimiter=',',create_using=nx.DiGraph(),nodetype=int)
```

In [0]:

```
df_train = df_final_train.drop(columns=['indicator_link',
       'jaccard_followers', 'jaccard_followees', 'cosine_followers',
```

```
       'cosine_followees', 'num_followers_s', 'num_followees_s',
       'num_followees_d', 'inter_followers', 'inter_followees', 'adar_index',
       'follows_back', 'same_comp', 'shortest_path', 'weight_in', 'weight_out',
       'weight_f1', 'weight_f2', 'weight_f3', 'weight_f4', 'page_rank_s',
       'page_rank_d', 'katz_s', 'katz_d', 'hubs_s', 'hubs_d', 'authorities_s',
       'authorities_d', 'svd_u_s_1', 'svd_u_s_2', 'svd_u_s_3', 'svd_u_s_4',
       'svd_u_s_5', 'svd_u_s_6', 'svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3',
       'svd_u_d_4', 'svd_u_d_5', 'svd_u_d_6', 'svd_v_s_1', 'svd_v_s_2',
       'svd_v_s_3', 'svd_v_s_4', 'svd_v_s_5', 'svd_v_s_6', 'svd_v_d_1',
       'svd_v_d_2', 'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5', 'svd_v_d_6',
       'preferential_score(Ui,Uj)', 'svd_dot'])
```

In [0]:

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

In [0]:

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

In [0]:

```
df_final_train.drop(['source_node', 'destination_node','indicator_link'],axis=1,inplace=True)
df_final_test.drop(['source_node', 'destination_node','indicator_link'],axis=1,inplace=True)
```

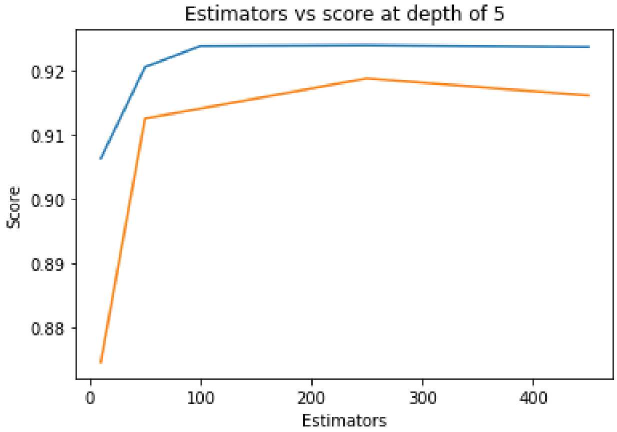
In [0]:

```
estimators = [10,50,100,250,450]
train_scores = []
test_scores = []
for i in estimators:
    clf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=5, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=52, min_samples_split=120,
            min_weight_fraction_leaf=0.0, n_estimators=i, n_jobs=-1,random_state=25,verbose=0,warm_
start=False)
    clf.fit(df_final_train,y_train)
    train_sc = f1_score(y_train,clf.predict(df_final_train))
    test_sc = f1_score(y_test,clf.predict(df_final_test))
    test_scores.append(test_sc)
    train_scores.append(train_sc)
    print('Estimators = ',i,'Train Score',train_sc,'test Score',test_sc)
plt.plot(estimators,train_scores,label='Train Score')
plt.plot(estimators,test_scores,label='Test Score')
plt.xlabel('Estimators')
plt.ylabel('Score')
plt.title('Estimators vs score at depth of 5')
```

```
Estimators =  10 Train Score 0.9063252121775113 test Score 0.8745605278006858
Estimators =  50 Train Score 0.9205725512208812 test Score 0.9125653355634538
Estimators =  100 Train Score 0.9238690848446947 test Score 0.9141199714153599
Estimators =  250 Train Score 0.9239789348046863 test Score 0.9188007232664732
Estimators =  450 Train Score 0.9237190618658074 test Score 0.9161507685828595
```

Out[0]:

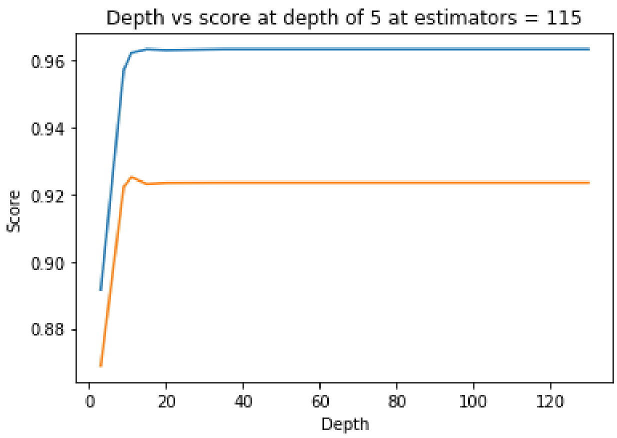
```
Text(0.5,1,'Estimators vs score at depth of 5')
```



In [0]:

```
depths = [3,9,11,15,20,35,50,70,130]
train_scores = []
test_scores = []
for i in depths:
    clf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=i, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=52, min_samples_split=120,
            min_weight_fraction_leaf=0.0, n_estimators=115, n_jobs=-1,random_state=25,verbose=0,war
m_start=False)
    clf.fit(df_final_train,y_train)
    train_sc = f1_score(y_train,clf.predict(df_final_train))
    test_sc = f1_score(y_test,clf.predict(df_final_test))
    test_scores.append(test_sc)
    train_scores.append(train_sc)
    print('depth = ',i,'Train Score',train_sc,'test Score',test_sc)
plt.plot(depths,train_scores,label='Train Score')
plt.plot(depths,test_scores,label='Test Score')
plt.xlabel('Depth')
plt.ylabel('Score')
plt.title('Depth vs score at depth of 5 at estimators = 115')
plt.show()
```

```
depth =  3 Train Score 0.8916120853581238 test Score 0.8687934859875491
depth =  9 Train Score 0.9572226298198419 test Score 0.9222953031452904
depth =  11 Train Score 0.9623451340902863 test Score 0.9252318758281279
depth =  15 Train Score 0.9634267621927706 test Score 0.9231288356496615
depth =  20 Train Score 0.9631629153051491 test Score 0.9235051024711141
depth =  35 Train Score 0.9634333127085721 test Score 0.9235601652753184
depth =  50 Train Score 0.9634333127085721 test Score 0.9235601652753184
depth =  70 Train Score 0.9634333127085721 test Score 0.9235601652753184
depth =  130 Train Score 0.9634333127085721 test Score 0.9235601652753184
```



In [0]:

```
from sklearn.metrics import f1_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
from sklearn.model_selection import RandomizedSearchCV
```

```
from scipy.stats import randint as sp_randint
from scipy.stats import uniform

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

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

rf_random = RandomizedSearchCV(clf, param_distributions=param_dist,
                                   n_iter=5,cv=10,scoring='f1',random_state=25)

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

```
mean test scores [0.96225043 0.96215493 0.96057081 0.96194015 0.96330005]
mean train scores [0.96294922 0.96266735 0.96115674 0.96263457 0.96430539]
```

In [0]:

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

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=14, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=28, min_samples_split=111,
            min_weight_fraction_leaf=0.0, n_estimators=121, n_jobs=-1,
            oob_score=False, random_state=25, verbose=0, warm_start=False)
```

In [0]:

```
clf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=14, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=28, min_samples_split=111,
            min_weight_fraction_leaf=0.0, n_estimators=121, n_jobs=-1,
            oob_score=False, random_state=25, verbose=0, warm_start=False)
```

In [0]:

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

In [0]:

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

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

In [0]:

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

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

    B =(C/C.sum(axis=0))
    plt.figure(figsize=(20,4))

    labels = [0,1]
    # representing A in heatmap format
    cmap=sns.light_palette("blue")
    plt.subplot(1, 3, 1)
    sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
```

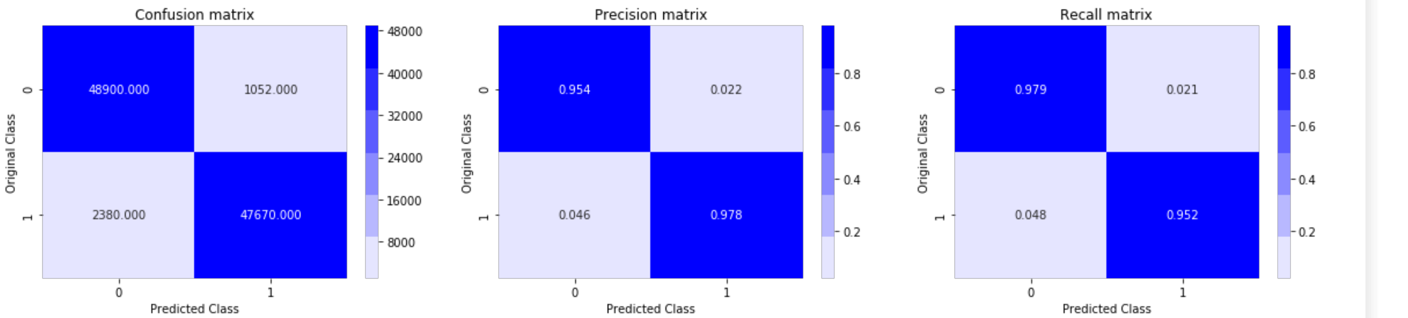
```
    sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Confusion matrix")

    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()
```

Confusion Matrix

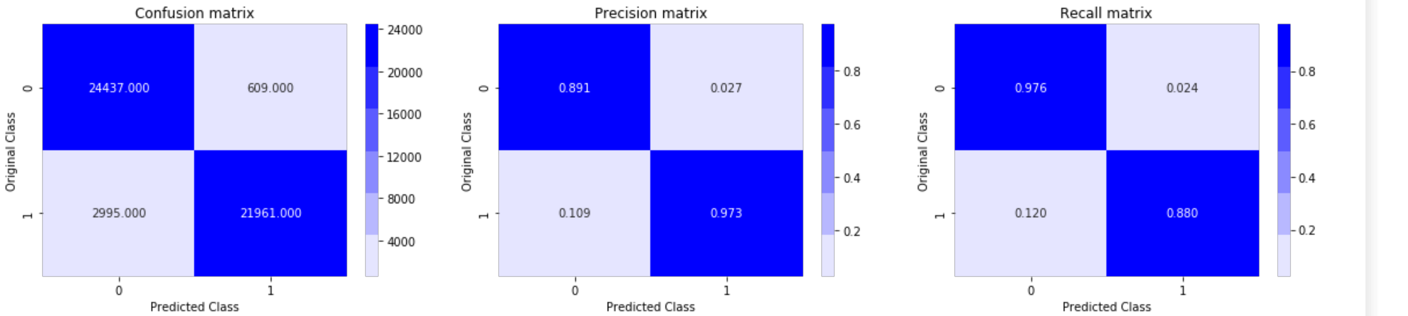
In [0]:

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

```
Train confusion_matrix
```



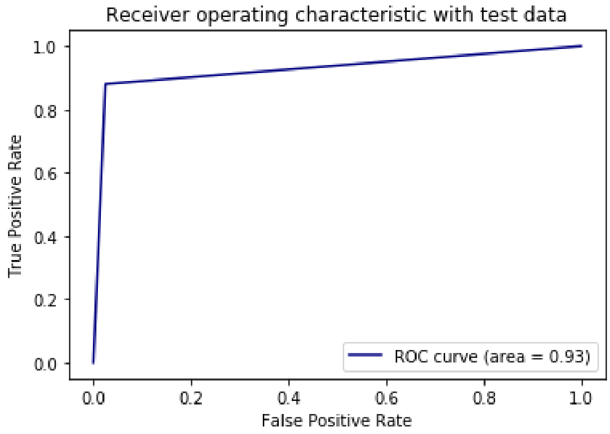

```
Test confusion_matrix
```



Plotting ROC Curve

In [0]:

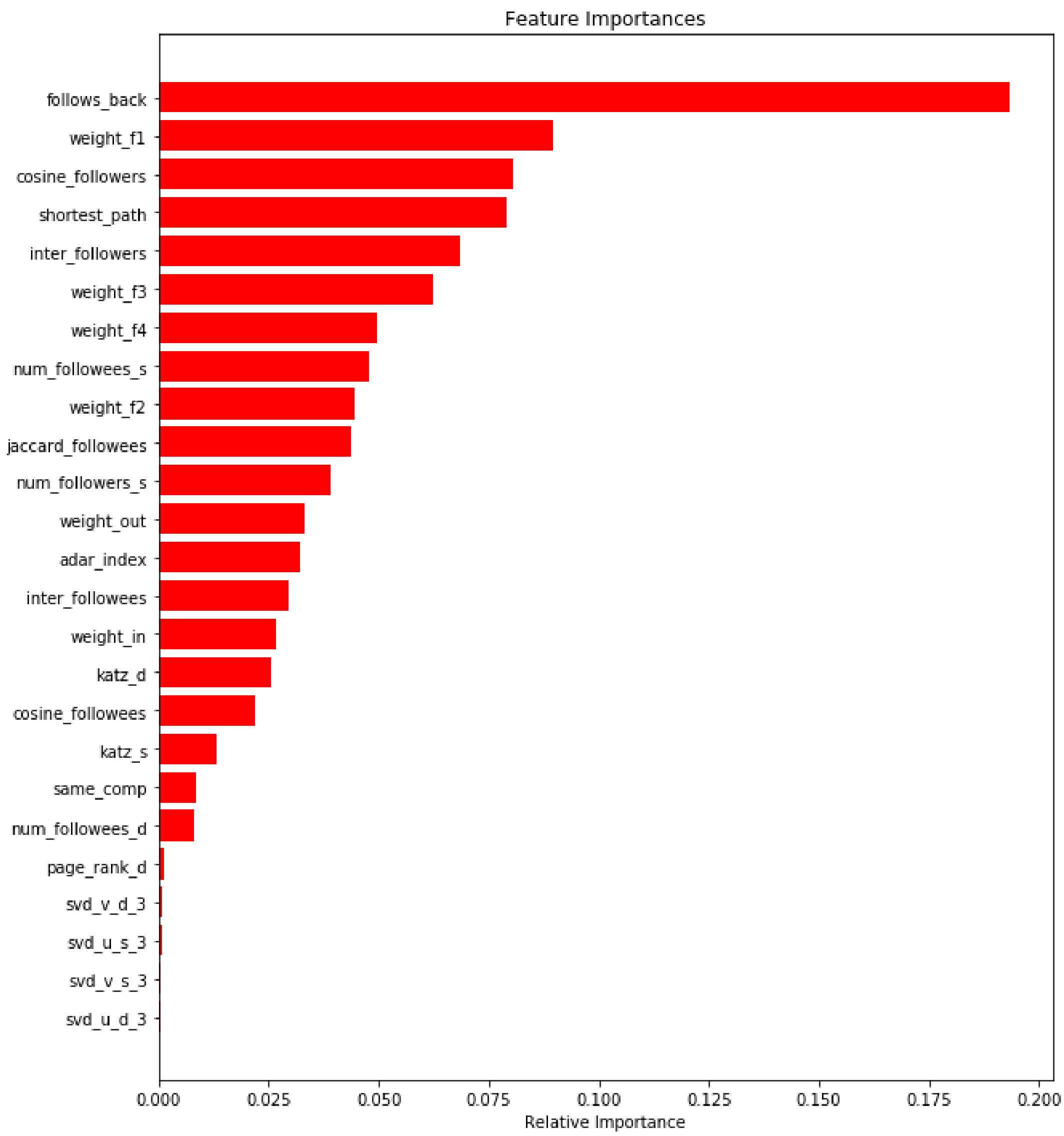
```
from sklearn.metrics import roc_curve, auc
fpr,tpr,ths = roc_curve(y_test,y_test_pred)
auc_sc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='navy',label='ROC curve (area = %0.2f)' % auc_sc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic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



Feature Importance

In [0]:

```
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()
```



*The most important feature of all of them is **follow back** feature*

XGBoost Model

In [7]:

```
model = xgb.XGBClassifier()
parameters = {'max_depth':[1,5,10],'n_estimators':[50,100,150,200], 'learning_rate':[0,0.1,0.5,1] }

clf = GridSearchCV(model, parameters,cv=5,scoring='f1',return_train_score=True)
clf.fit(df_final_train, y_train)
```

Out[7]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                     colsample_bylevel=1, colsample_bynode=1,
                                     colsample_bytree=1, gamma=0,
                                     learning_rate=0.1, max_delta_step=0,
                                     max_depth=3, min_child_weight=1,
                                     missing=None, n_estimators=100, n_jobs=1,
                                     nthread=None, objective='binary:logistic',
                                     random_state=0, reg_alpha=0, reg_lambda=1,
                                     scale_pos_weight=1, seed=None, silent=None,
                                     subsample=1, verbosity=1),
             iid='deprecated', n_jobs=None,
             param_grid={'learning_rate': [0, 0.1, 0.5, 1],
                         'max_depth': [1, 5, 10],
                         'n_estimators': [50, 100, 150, 200]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='f1', verbose=0)
```

In [8]:

```
results = pd.DataFrame.from_dict(clf.cv_results_)
results
```

Out[8]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_learning_rate	param_max_depth	param_n_estir
0	5.293298	0.372677	0.029432	0.000437	0	1	50
1	9.767770	0.130188	0.034827	0.000130	0	1	100
2	14.395171	0.201271	0.041009	0.000411	0	1	150
3	19.045856	0.249639	0.046329	0.000992	0	1	200
4	9.247545	0.134063	0.031119	0.002537	0	5	50
5	17.976655	0.207548	0.035097	0.000597	0	5	100
6	26.801553	0.351903	0.041343	0.000693	0	5	150
7	35.512280	0.320803	0.049025	0.003150	0	5	200

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_learning_rate	param_max_depth	param_n_esti
8	9.297703	0.075156	0.029134	0.000248	0	10	50
9	17.960515	0.204389	0.035154	0.000386	0	10	100
10	26.862908	0.307729	0.040693	0.000534	0	10	150
11	35.935152	0.601251	0.050276	0.003593	0	10	200
12	5.224301	0.032876	0.031630	0.002356	0.1	1	50
13	9.647452	0.109673	0.035888	0.000311	0.1	1	100
14	12.912822	0.081637	0.040854	0.002026	0.1	1	150
15	16.060659	0.098565	0.041892	0.000237	0.1	1	200
16	9.414715	0.039430	0.031090	0.002583	0.1	5	50
17	17.775616	0.058846	0.035776	0.000465	0.1	5	100
18	21.497155	0.068947	0.040027	0.000244	0.1	5	150
19	24.893124	0.096082	0.043248	0.000829	0.1	5	200
20	9.559423	0.033718	0.031076	0.001258	0.1	10	50
21	17.779846	0.021406	0.037105	0.001220	0.1	10	100
22	21.543649	0.070453	0.040314	0.000669	0.1	10	150
23	24.810440	0.063423	0.042254	0.000202	0.1	10	200

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_learning_rate	param_max_depth	param_n_estir
24	4.331822	0.025802	0.028689	0.000253	0.5	1	50
25	7.590193	0.031570	0.033358	0.003554	0.5	1	100
26	10.856622	0.017206	0.037413	0.003155	0.5	1	150
27	14.169348	0.047842	0.038107	0.000460	0.5	1	200
28	6.064367	0.029661	0.029169	0.000334	0.5	5	50
29	9.378761	0.042094	0.031505	0.000418	0.5	5	100
30	12.634556	0.026014	0.035485	0.000481	0.5	5	150
31	15.722478	0.043080	0.037554	0.000372	0.5	5	200
32	6.026124	0.038738	0.028696	0.000472	0.5	10	50
33	9.237901	0.023303	0.031758	0.000553	0.5	10	100
34	12.484442	0.069820	0.034907	0.000434	0.5	10	150
35	15.725432	0.035642	0.037854	0.000733	0.5	10	200
36	3.997875	0.019430	0.029214	0.002305	1	1	50
37	7.176247	0.023981	0.030736	0.000453	1	1	100
38	10.400242	0.036906	0.034983	0.001284	1	1	150

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_learning_rate	param_max_depth	param_n_estir
39	13.563983	0.016601	0.037238	0.001582	1	1	200
40	4.854039	0.023113	0.028086	0.000138	1	5	50
41	8.051349	0.016581	0.030970	0.000739	1	5	100
42	11.249002	0.033681	0.034361	0.001230	1	5	150
43	14.494715	0.045002	0.038054	0.002760	1	5	200
44	4.855808	0.029675	0.028406	0.000850	1	10	50
45	8.062193	0.051644	0.030944	0.000498	1	10	100
46	11.262729	0.048420	0.033710	0.000492	1	10	150
47	14.534151	0.046507	0.037512	0.000794	1	10	200

- By seeing the above results i can take max depth of 1 and learning rate of 0.1 and n_estimators count as 50. As 50,100,150 estimators are leading to same result.

In [19]:

```
clf.best_estimator_
```

Out[19]:

```
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=1,
              min_child_weight=1, missing=None, n_estimators=50, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

In [20]:

```
#Training the model
xgbmodel = 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=1,
              min_child_weight=1, missing=None, n_estimators=50, n_jobs=-1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
xgbmodel.fit(df_final_train, y_train)
```

Out[20]:

```
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=1,
              min_child_weight=1, missing=None, n_estimators=50, n_jobs=-1,
              nthread=None, objective='binary:logistic', random_state=0,
```

```
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

In [0]:

```python
#Predicting using model
y_train_pred = xgbmodel.predict(df_final_train)
y_test_pred = xgbmodel.predict(df_final_test)
```

In [22]:

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

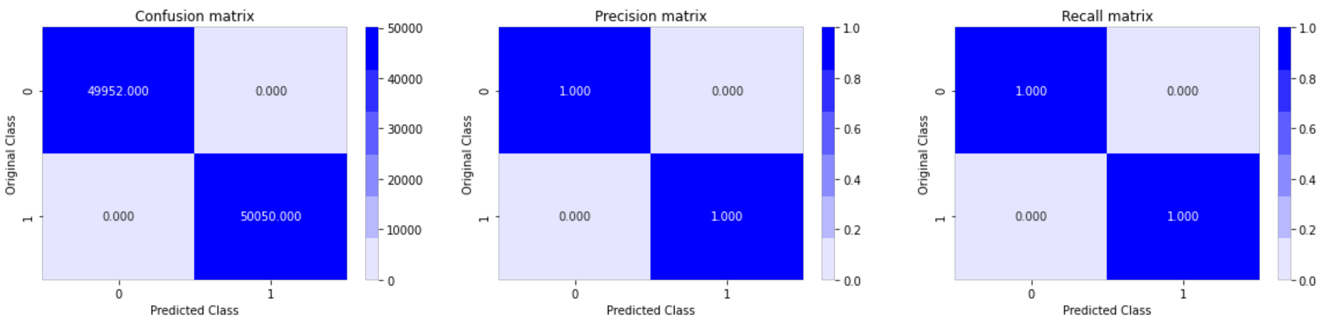
```
Train f1 score 1.0
Test f1 score 1.0
```

Confusion Matrix

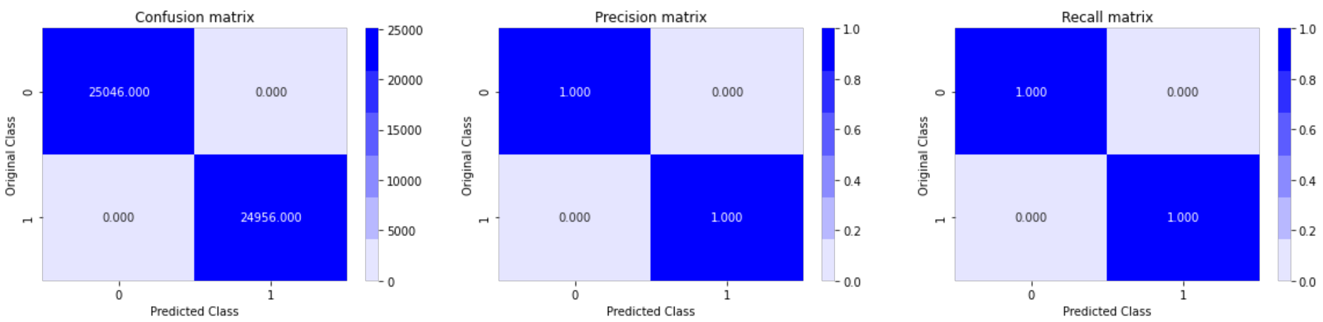
In [15]:

```python
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
```

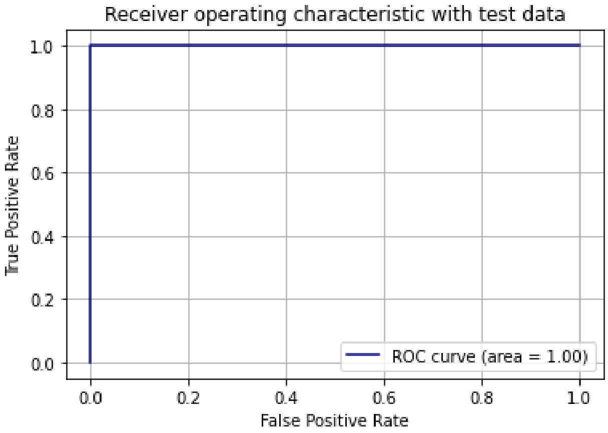


ROC curve

In [16]:

```python
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.grid()
plt.legend()
plt.show()
```

```
plt.show()
```


Receiver operating characteristic with test data
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 1.00)