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 (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://www.cs.cornell.edu/home/kleinber/link-pred.pdf)
  - https://www3.nd.edu/~dial/publications/lichtenwalter2010new.pdf (https://www3.nd.edu/~dial/publications/lichtenwalter2010new.pdf)
  - https://www.youtube.com/watch?v=2M77Hgy17cg (https://www.youtube.com/watch?v=2M77Hgy17cg)

Business objectives and constraints:

- No low-latency requirement.
- Probability of prediction is useful to recommend highest 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 #library for computing on graphs or network
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
```

Reading Data

In [2]:

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

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/aut
h?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleu
sercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&respon
se_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fd
ocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%
2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2
fwww.googleapis.com%2fauth%2fpeopleapi.readonly (https://accounts.goog
le.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc
0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoaut
h%3a2.0%3aoob&response_type=code&scope=email%20https%3a%2f%2fwww.googl
eapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2faut
h%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.rea
donly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly)

Enter your authorization code:
··········
Mounted at /content/drive
```

In [3]:

```
g=nx.read_edgelist('/content/drive/My Drive/appliedai/facebook case study/data/afte
print(nx.info(g))
```

```
Name:
Type: DiGraph
Number of nodes: 1862220
Number of edges: 9437519
Average in degree:   5.0679
Average out degree:   5.0679
```

Displaying a sub graph

In [0]:

```
subgraph=nx.read_edgelist('/content/drive/My Drive/appliedai/facebook case study/da
# https://stackoverflow.com/questions/9402255/drawing-a-huge-graph-with-networkx-ar

pos=nx.spring_layout(subgraph)
nx.draw(subgraph,pos,node_color='#A0CBE2',edge_color='#00bb5e',width=1,edge_cmap=pl
plt.savefig("graph_sample.pdf")
print(nx.info(subgraph))
```

Exploratory Data Analysis

In [0]:

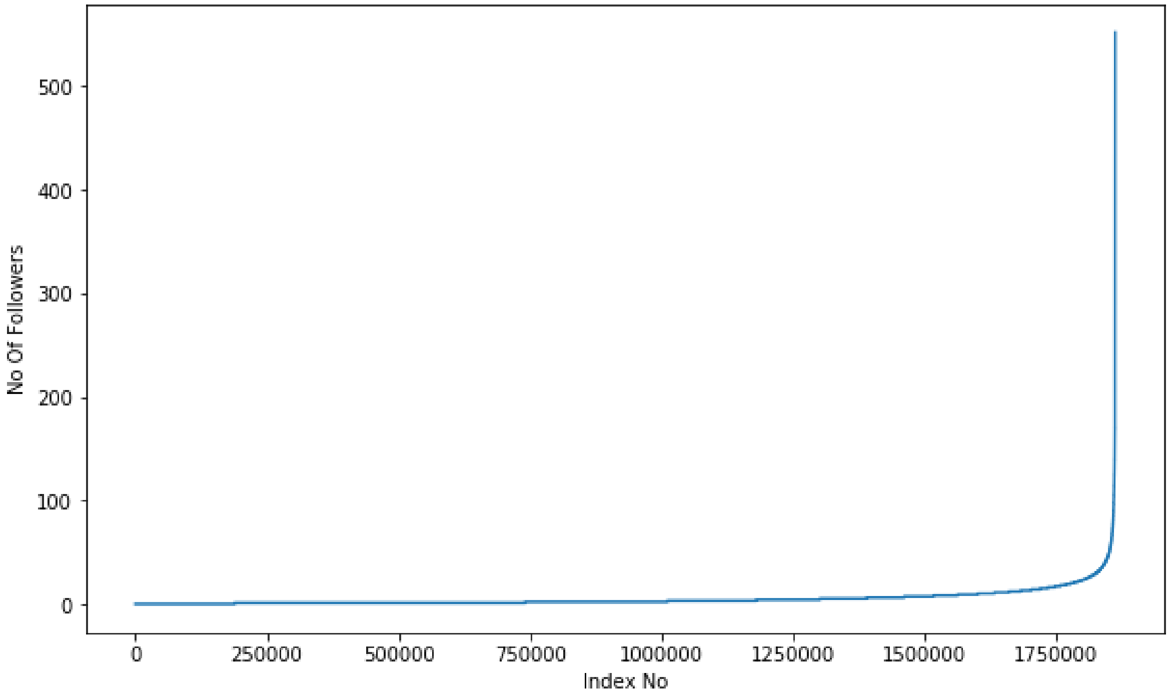
```
# No of Unique persons
print("The number of unique persons",len(g.nodes()))
```

```
The number of unique persons 1862220
```

No of followers for each person

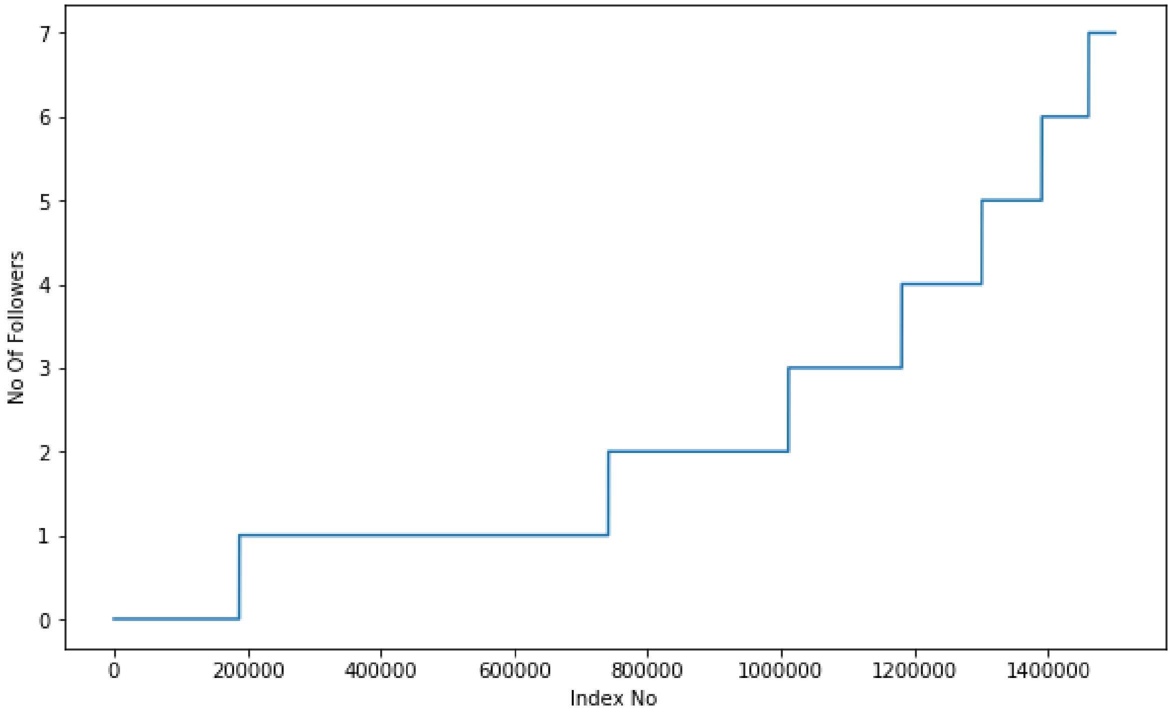
In [0]:

```
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.ylabel('No Of Followers')
plt.show()
```



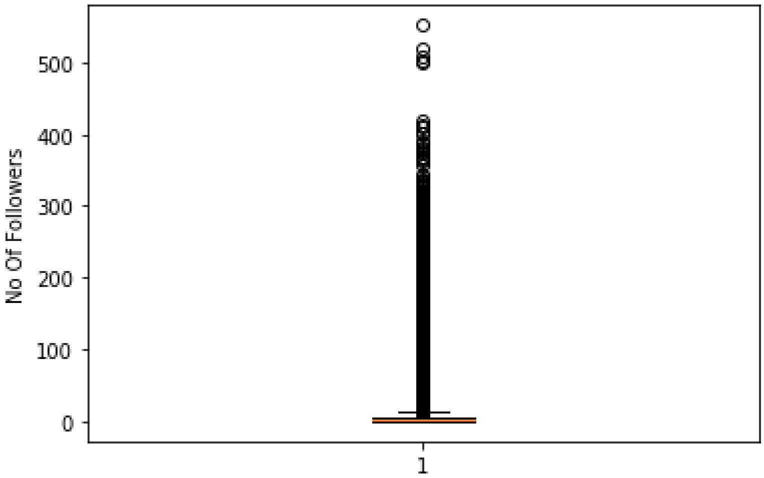
In [0]:

```
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.ylabel('No Of Followers')
plt.show()
```



In [0]:

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



In [0]:

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

```
90 percentile value is 12.0
91 percentile value is 13.0
92 percentile value is 14.0
93 percentile value is 15.0
94 percentile value is 17.0
95 percentile value is 19.0
96 percentile value is 21.0
97 percentile value is 24.0
98 percentile value is 29.0
99 percentile value is 40.0
100 percentile value is 552.0
```

99% of people have followers less than or equal to 40.

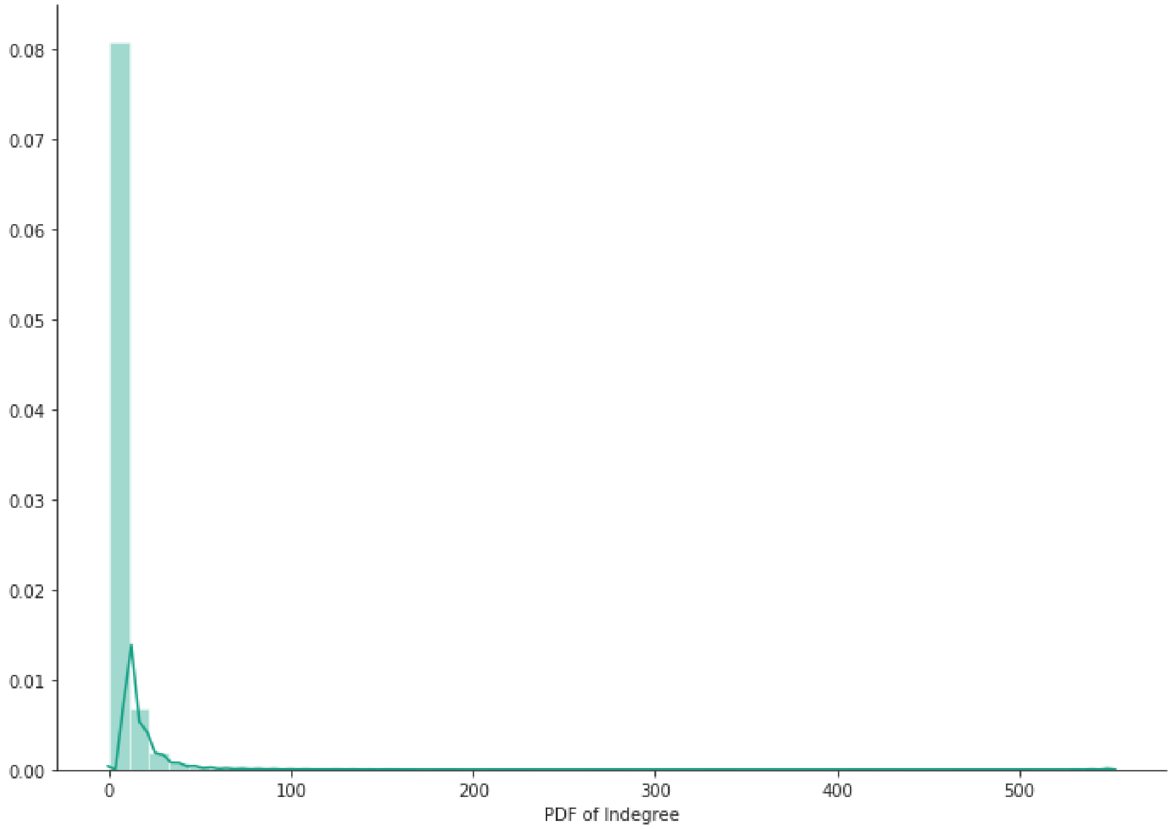
In [0]:

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

```
99.1 percentile value is 42.0
99.2 percentile value is 44.0
99.3 percentile value is 47.0
99.4 percentile value is 50.0
99.5 percentile value is 55.0
99.6 percentile value is 61.0
99.7 percentile value is 70.0
99.8 percentile value is 84.0
99.9 percentile value is 112.0
100.0 percentile value is 552.0
```

In [0]:

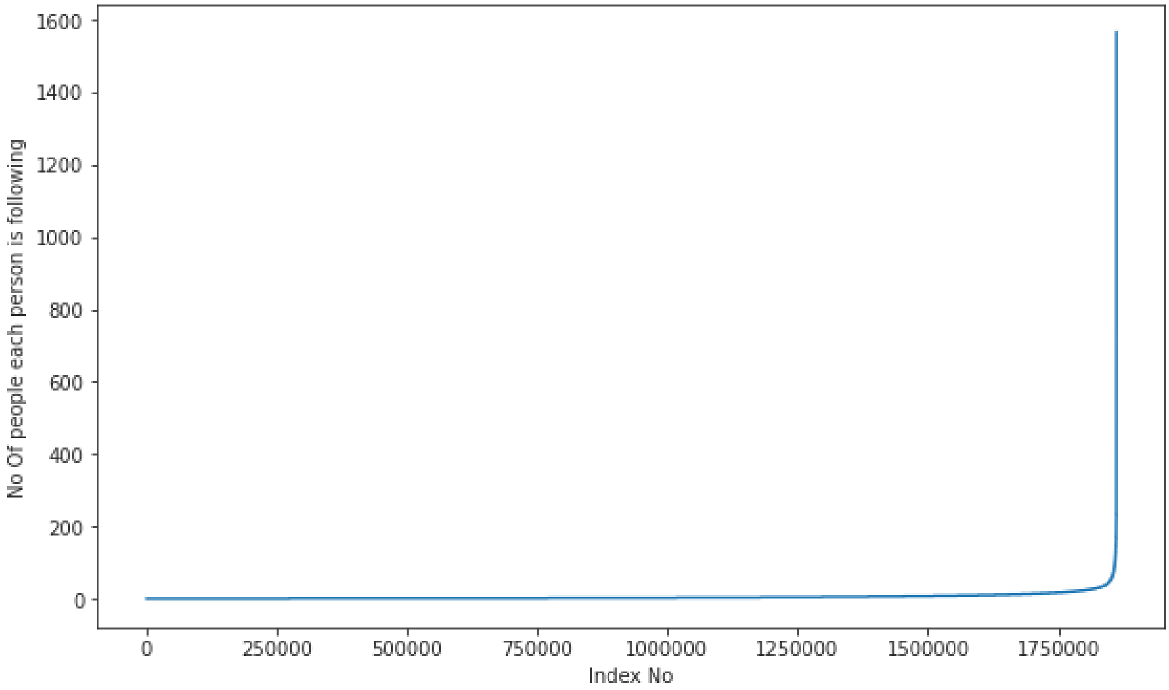
```
%matplotlib inline
sns.set_style('ticks')
fig, ax = plt.subplots()
fig.set_size_inches(11.7, 8.27)
sns.distplot(indegree_dist, color='#16A085')
plt.xlabel('PDF of Indegree')
sns.despine()
#plt.show()
```



No of people each person is following

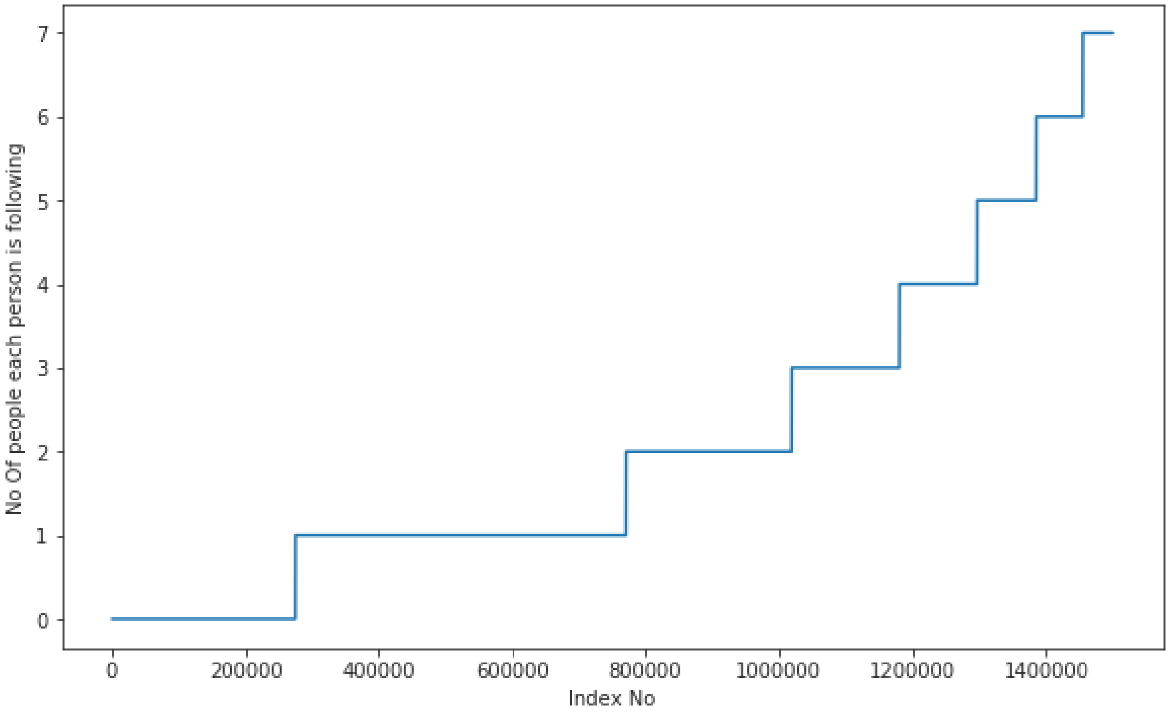
In [0]:

```
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.ylabel('No Of people each person is following')
plt.show()
```



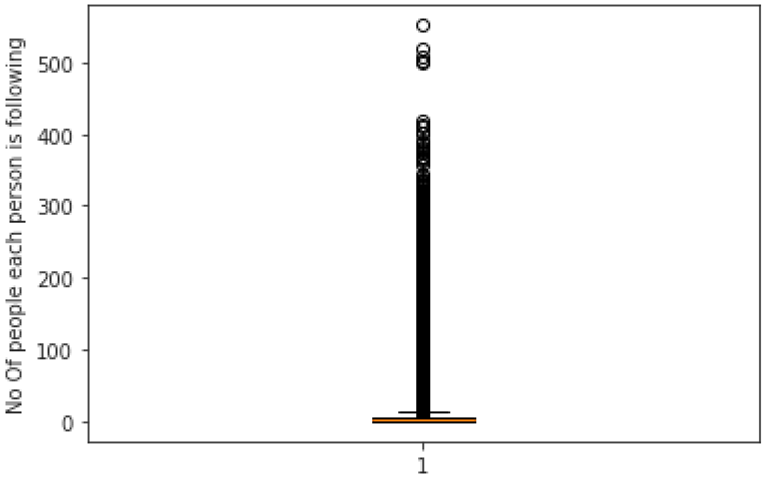
In [0]:

```
indegree_dist = list(dict(g.in_degree()).values())
indegree_dist.sort()
plt.figure(figsize=(10,6))
plt.plot(outdegree_dist[0:1500000])
plt.xlabel('Index No')
plt.ylabel('No Of people each person is following')
plt.show()
```



In [0]:

```
plt.boxplot(indegree_dist)
plt.ylabel('No Of people each person is following')
plt.show()
```



In [0]:

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

```
90 percentile value is 12.0
91 percentile value is 13.0
92 percentile value is 14.0
93 percentile value is 15.0
94 percentile value is 17.0
95 percentile value is 19.0
96 percentile value is 21.0
97 percentile value is 24.0
98 percentile value is 29.0
99 percentile value is 40.0
100 percentile value is 1566.0
```

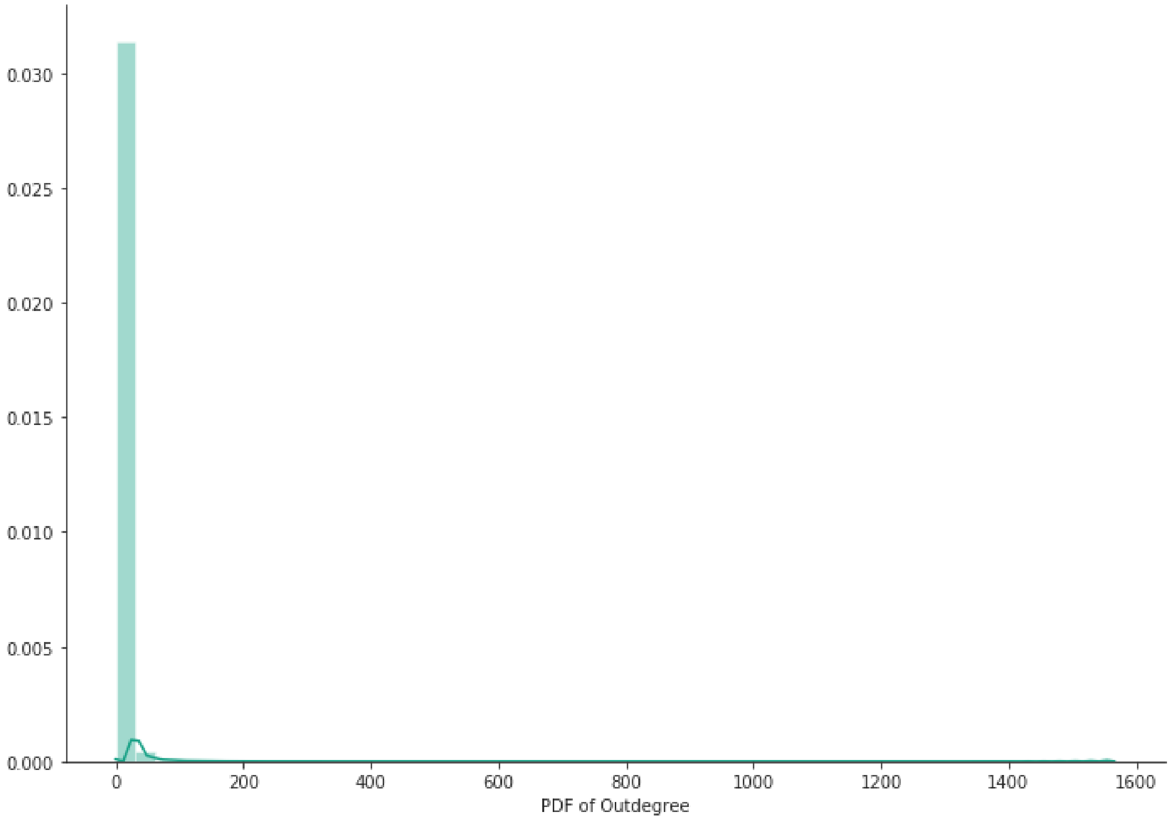
In [0]:

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

```
99.1 percentile value is 42.0
99.2 percentile value is 45.0
99.3 percentile value is 48.0
99.4 percentile value is 52.0
99.5 percentile value is 56.0
99.6 percentile value is 63.0
99.7 percentile value is 73.0
99.8 percentile value is 90.0
99.9 percentile value is 123.0
100.0 percentile value is 1566.0
```

In [0]:

```
sns.set_style('ticks')
fig, ax = plt.subplots()
fig.set_size_inches(11.7, 8.27)
sns.distplot(outdegree_dist, color='#16A085')
plt.xlabel('PDF of Outdegree')
sns.despine()
```



In [0]:

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

```
No of persons those are not following anyone are 274512 and % is 14.74
1115442858524
```

In [0]:

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

```
No of persons having zero followers are 188043 and % is 10.09778651287
1734
```

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 fol
```

```
No of persons those are not not following anyone and also not having a
ny followers are 0
```

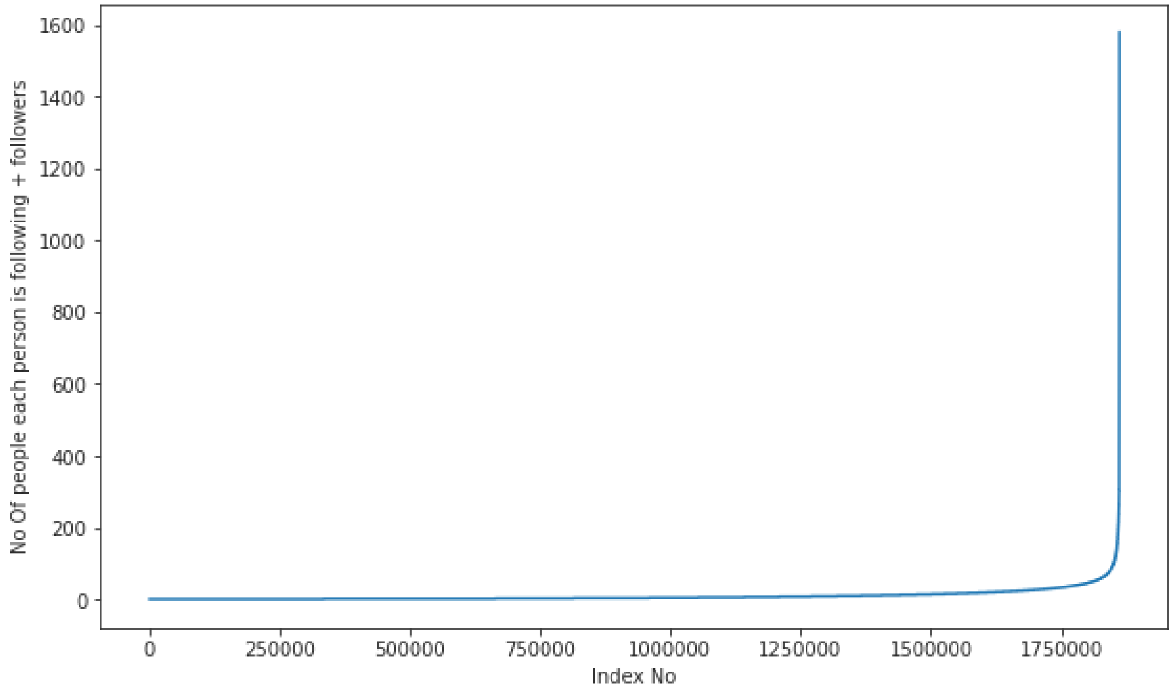
both followers + following

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()))
```

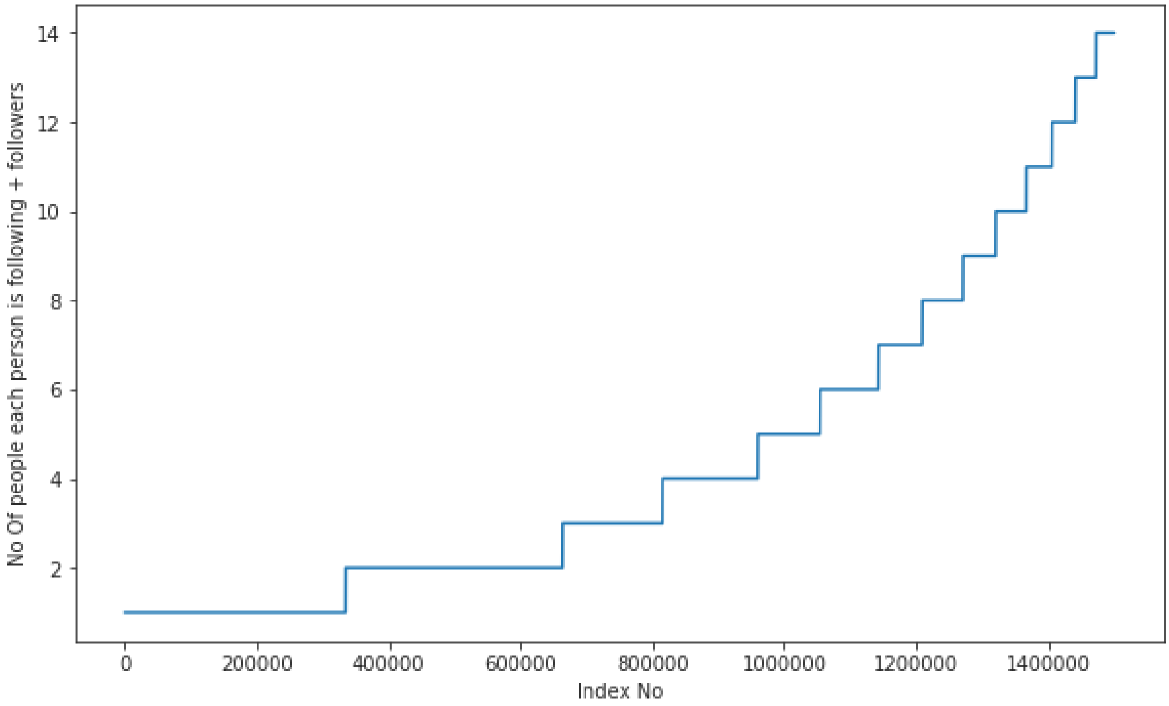
In [0]:

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



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.ylabel('No Of people each person is following + followers')
plt.show()
```



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
```

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/1
```

```
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
```

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 fol
```

```
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 fol
```

```
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_d
```

```
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
```

Posing a problem as classification problem

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/appliedai/facebook case study/data/a
    #getting all set of edges
    r = csv.reader(open('data/after_eda/train_woheader.csv','r'))
    edges = dict()
    for edge in r:
        edges[(edge[0], edge[1])] = 1


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

                    missing_edges.add((a,b))
                else:
                    continue
            except:
                    missing_edges.add((a,b))
        else:
            continue
    pickle.dump(missing_edges,open('data/after_eda/missing_edges_final.p','wb'))
else:
    missing_edges = pickle.load(open('/content/drive/My Drive/appliedai/facebook ca
```

```
CPU times: user 2.17 s, sys: 821 ms, total: 2.99 s
Wall time: 4.46 s
```

In [0]:

```
len(missing_edges)
```

Out[33]:

```
9437519
```

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/appliedai/facebook case study/data/
    #reading total data df
    df_pos = pd.read_csv('data/train.csv')
    df_neg = pd.DataFrame(list(missing_edges), columns=['source_node', 'destination

    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
    #and for feature generation
    X_train_pos, X_test_pos, y_train_pos, y_test_pos  = train_test_split(df_pos,np.
    X_train_neg, X_test_neg, y_train_neg, y_test_neg  = train_test_split(df_neg,np.

    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.shap
    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[

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

In [0]:

```
if (os.path.isfile('/content/drive/My Drive/appliedai/facebook case study/data/afte
    train_graph=nx.read_edgelist('/content/drive/My Drive/appliedai/facebook case s
    test_graph=nx.read_edgelist('/content/drive/My Drive/appliedai/facebook case st
    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
```

we have a cold start problem here

In [0]:

```
#final train and test data sets
if (not os.path.isfile('/content/drive/My Drive/appliedai/facebook case study/data/
(not os.path.isfile('/content/drive/My Drive/appliedai/facebook case study/data/aft
(not os.path.isfile('/content/drive/My Drive/appliedai/facebook case study/data/tra
(not os.path.isfile('/content/drive/My Drive/appliedai/facebook case study/data/tes
(os.path.isfile('/content/drive/My Drive/appliedai/facebook case study/data/after_e
(os.path.isfile('/content/drive/My Drive/appliedai/facebook case study/data/after_e
(os.path.isfile('/content/drive/My Drive/appliedai/facebook case study/data/after_e
(os.path.isfile('/content/drive/My Drive/appliedai/facebook case study/data/after_e

    X_train_pos = pd.read_csv('/content/drive/My Drive/appliedai/facebook case stud
    X_test_pos = pd.read_csv('/content/drive/My Drive/appliedai/facebook case study
    X_train_neg = pd.read_csv('/content/drive/My Drive/appliedai/facebook case stud
    X_test_neg = pd.read_csv('/content/drive/My Drive/appliedai/facebook case study

    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.shap
    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[

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

Featurization

Reading Preprocessed Data

In [4]:

```
if os.path.isfile('/content/drive/My Drive/appliedai/facebook case study/data/after
    train_graph=nx.read_edgelist('/content/drive/My Drive/appliedai/facebook case s
    print(nx.info(train_graph))
else:
    print("please run the FB_EDA.ipynb or download the files from drive")
```

```
Name: 
Type: DiGraph
Number of nodes: 1780722
Number of edges: 7550015
Average in degree:   4.2399
Average out degree:   4.2399
```

Similarity measures

Jaccard Distance:

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

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

In [0]:

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

In [6]:

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

```
0.0
```

In [7]:

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

```
0.0
```

In [0]:

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

In [9]:

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

```
0.0
```

In [10]:

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

```
0
```

Cosine distance

$$CosineDistance = \frac{|X \cap Y|}{SQRT|X| \cdot |Y|}$$

In [0]:

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

In [12]:

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

```
0.0
```

In [13]:

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

```
0
```

In [0]:

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

        if len(set(train_graph.predecessors(a))) == 0  | len(set(train_graph.predec
            return 0
        sim = (len(set(train_graph.predecessors(a)).intersection(set(train_graph.pr
                                     (math.sqrt(len(set(train_graph.predecessors(a)
        return sim
    except:
        return 0
```

In [15]:

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

```
0.02886751345948129
```

In [16]:

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

```
0
```

Ranking Measures

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

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.**

Page Ranking

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

In [0]:

```
if not os.path.isfile('/content/drive/My Drive/appliedai/facebook case study/data/f
    pr = nx.pagerank(train_graph, alpha=0.85)
    pickle.dump(pr,open('/content/drive/My Drive/appliedai/facebook case study/data
else:
    pr = pickle.load(open('/content/drive/My Drive/appliedai/facebook case study/da
```

In [18]:

```
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 [19]:

```
#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
```

Other Graph Features

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 [21]:

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

Out[21]:

```
10
```

In [22]:

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

Out[22]:

```
-1
```

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 [24]:

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

Out[24]:

```
0
```

In [25]:

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

Out[25]:

```
0
```

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.successo
        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 [27]:

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

Out[27]:

```
0
```

In [28]:

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

Out[28]:

```
0
```

If person was following back:

In [0]:

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

In [30]:

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

Out[30]:

```
1
```

In [31]:

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

Out[31]:

```
0
```

Katz Centrality:

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

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

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

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

$$\lambda$$

.

The parameter

$$\beta$$

controls the initial centrality and

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

In [0]:

```
if not os.path.isfile('/content/drive/My Drive/appliedai/facebook case study/data/f
    katz = nx.katz.katz_centrality(train_graph,alpha=0.005,beta=1)
    pickle.dump(katz,open('/content/drive/My Drive/appliedai/facebook case study/da
else:
    katz = pickle.load(open('/content/drive/My Drive/appliedai/facebook case study/
```

In [33]:

```
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 [34]:

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

```
0.0007483800935562018
```

Hits Score

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

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

In [0]:

```
if not os.path.isfile('/content/drive/My Drive/appliedai/facebook case study/data/f
    hits = nx.hits(train_graph, max_iter=100, tol=1e-08, nstart=None, normalized=Tr
    pickle.dump(hits,open('/content/drive/My Drive/appliedai/facebook case study/da
else:
    hits = pickle.load(open('/content/drive/My Drive/appliedai/facebook case study/
```

In [36]:

```


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
```

Reading a sample of Data from both train and test

In [0]:

```
import random
if os.path.isfile('/content/drive/My Drive/appliedai/facebook case study/data/after
    filename = "/content/drive/My Drive/appliedai/facebook case study/data/after_ed
    # 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 (exclu
    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]:

```
if os.path.isfile('/content/drive/My Drive/appliedai/facebook case study/data/after
    filename = "/content/drive/My Drive/appliedai/facebook case study/data/after_ed
    # 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 (excluc
    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 [39]:

```
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 [40]:

```
df_final_train = pd.read_csv('/content/drive/My Drive/appliedai/facebook case study
df_final_train['indicator_link'] = pd.read_csv('/content/drive/My Drive/appliedai/f
print("Our train matrix size ",df_final_train.shape)
df_final_train.head(2)
```

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

Out[40]:

	source_node	destination_node	indicator_link
0	273084	1505602	1
1	1011666	450550	1

In [41]:

```
df_final_test = pd.read_csv('/content/drive/My Drive/appliedai/facebook case study/
df_final_test['indicator_link'] = pd.read_csv('/content/drive/My Drive/appliedai/fa
print("Our test matrix size ",df_final_test.shape)
df_final_test.head(2)
```

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

Out[41]:

	source_node	destination_node	indicator_link
0	848424	784690	1
1	922285	491551	1

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 os.path.isfile('/content/drive/My Drive/appliedai/facebook case study/data/fea_s
    #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'
    df_final_test['jaccard_followers'] = df_final_test.apply(lambda row:
                                            jaccard_for_followers(row['source_node'

    #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'
    df_final_test['jaccard_followees'] = df_final_test.apply(lambda row:
                                            jaccard_for_followees(row['source_node'


        #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']
    df_final_test['cosine_followers'] = df_final_test.apply(lambda row:
                                            cosine_for_followers(row['source_node']

    #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']
    df_final_test['cosine_followees'] = df_final_test.apply(lambda row:
                                            cosine_for_followees(row['source_node']
```

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, inte
```

In [0]:

```
if os.path.isfile('/content/drive/My Drive/appliedai/facebook case study/data/fea_s
    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_f

    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_fea

    #hdf = HDFStore('/content/drive/My Drive/appliedai/facebook case study/data/fea
    #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/appliedai/facebook case stud
    df_final_test = read_hdf('/content/drive/My Drive/appliedai/facebook case study
```

In [48]:

```
df_final_train.columns
```

Out[48]:

```
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'],
      dtype='object')
```

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 os.path.isfile('/content/drive/My Drive/appliedai/facebook case study/data/fea_s
    #mapping adar index on train
    df_final_train['adar_index'] = df_final_train.apply(lambda row: calc_adar_in(ro
    #mapping adar index on test
    df_final_test['adar_index'] = df_final_test.apply(lambda row: calc_adar_in(row[

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

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

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

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

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

    #hdf = HDFStore('data/fea_sample/storage_sample_stage2.h5')
    #hdf.put('train_df',df_final_train, format='table', data_columns=True)
    #hdf.put('test_df',df_final_test, format='table', data_columns=True)
    #hdf.close()
else:
    df_final_train = read_hdf('/content/drive/My Drive/appliedai/facebook case stud
    df_final_test = read_hdf('/content/drive/My Drive/appliedai/facebook case study
```

Adding new set of features

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

1. Weight Features
    - weight of incoming edges
    - weight of outgoing edges
    - weight of incoming edges + weight of outgoing edges
    - weight of incoming edges * weight of outgoing edges
    - 2*weight of incoming edges + weight of outgoing edges
    - weight of incoming edges + 2*weight of outgoing edges
2. Page Ranking of source
3. Page Ranking of dest
4. katz of source
5. katz of dest
6. hubs of source
7. hubs of dest
8. authorities_s of source
9. authorities_s of dest

Weight Features

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

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

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

In [53]:

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

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

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

```
100%|██████████| 1780722/1780722 [00:17<00:00, 104196.79it/s]
```

In [0]:

```
if os.path.isfile('/content/drive/My Drive/appliedai/facebook case study/data/fea_s
    #mapping to pandas train
    df_final_train['weight_in'] = df_final_train.destination_node.apply(lambda x: W
    df_final_train['weight_out'] = df_final_train.source_node.apply(lambda x: Weigh

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


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

    #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.weigh
    df_final_test['weight_f4'] = (1*df_final_test.weight_in + 2*df_final_test.weigh
```

In [0]:

```
if os.path.isfile('/content/drive/My Drive/appliedai/facebook case study/data/fea_s

    #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.ge
    df_final_train['page_rank_d'] = df_final_train.destination_node.apply(lambda x:

    df_final_test['page_rank_s'] = df_final_test.source_node.apply(lambda x:pr.get(
    df_final_test['page_rank_d'] = df_final_test.destination_node.apply(lambda x: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(
    df_final_train['katz_d'] = df_final_train.destination_node.apply(lambda x: katz

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

    #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].g
    df_final_train['hubs_d'] = df_final_train.destination_node.apply(lambda x: hits

    df_final_test['hubs_s'] = df_final_test.source_node.apply(lambda x: hits[0].get
    df_final_test['hubs_d'] = df_final_test.destination_node.apply(lambda x: hits[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: hi
    df_final_train['authorities_d'] = df_final_train.destination_node.apply(lambda 

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

    #hdf = HDFStore('data/fea_sample/storage_sample_stage3.h5')
    #hdf.put('train_df',df_final_train, format='table', data_columns=True)
    #hdf.put('test_df',df_final_test, format='table', data_columns=True)
    #hdf.close()
else:
    df_final_train = read_hdf('/content/drive/My Drive/appliedai/facebook case stud
    df_final_test = read_hdf('/content/drive/My Drive/appliedai/facebook case study
```

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

In [0]:

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

In [0]:

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

In [59]:

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

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

In [0]:

```
if  os.path.isfile('/content/drive/My Drive/appliedai/facebook case study/data/fea
    #=================================================================================

    df_final_train[['svd_u_s_1', 'svd_u_s_2','svd_u_s_3', 'svd_u_s_4', 'svd_u_s_5',
    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'
    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',
    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'
    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',
    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',
    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',
    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',
    df_final_test.destination_node.apply(lambda x: svd(x, V.T)).apply(pd.Series)
    #=================================================================================

    #hdf = HDFStore('/content/drive/My Drive/appliedai/facebook case study/data/fe
    #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 [64]:

```
df_final_test.columns
```

Out[64]:

```
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', 'weigh
t_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')
```

Adding Preferential Attachment feature

Preferential Attachment feature:-

> 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; Refer: for more details (http://be.amazd.com/link-prediction/)

$$Score(x,y) = |\Gamma(x)| \cdot |\Gamma(y)|$$

In [0]:

```
#Preferential Attachment for followers
df_final_train["followers_Preferential_Attachment"]=df_final_train["num_followers_s
df_final_test["followers_Preferential_Attachment"]=df_final_test["num_followers_s"]
```

In [0]:

```
#Preferential Attachment for followees
df_final_train["followees_Preferential_Attachment"]=df_final_train["num_followees_s
df_final_test["followees_Preferential_Attachment"]=df_final_test["num_followees_s"]
```

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,inp
df_final_test.drop(['source_node', 'destination_node','indicator_link'],axis=1,inpl
```

In [69]:

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

Out[69]:

	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	num_followers_s	n
0	0.0	0.0	0.000000	0.000000	11	
1	0.2	0.2	0.176777	0.353553	2	

Adding svd_dot. feature

svd. feature:-

> 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 (https://storage.googleapis.com/kaggle-forum-message-attachments/2594/supervised_link_prediction.pdf)

In [0]:

```
#https://github.com/krpiyush5/Facebook-Friend-Recommendation-using-Graph-Mining/blo
if not os.path.isfile('/content/drive/My Drive/Facebook Friend Recommendation/data/
    #Svd_dot for  train
    df_final_train['svd_us_dot_ud'] = (df_final_train['svd_u_s_1']*df_final_train['
                                    + (df_final_train['svd_u_s_2']*df_final_train['
                                    + (df_final_train['svd_u_s_3']*df_final_train['
                                    + (df_final_train['svd_u_s_4']*df_final_train['
                                    + (df_final_train['svd_u_s_5']*df_final_train['
                                    + (df_final_train['svd_u_s_6']*df_final_train['
                                    + (df_final_train['svd_v_s_1']*df_final_train['
                                    + (df_final_train['svd_v_s_2']*df_final_train['
                                    + (df_final_train['svd_v_s_3']*df_final_train['
                                    + (df_final_train['svd_v_s_4']*df_final_train['
                                    + (df_final_train['svd_v_s_5']*df_final_train['
                                    + (df_final_train['svd_v_s_6']*df_final_train['
    #Svd_dot for  test
    df_final_test['svd_us_dot_ud'] = (df_final_test['svd_u_s_1']*df_final_test['svd
                                   + (df_final_test['svd_u_s_2']*df_final_test['svd
                                   + (df_final_test['svd_u_s_3']*df_final_test['svd
                                   + (df_final_test['svd_u_s_4']*df_final_test['svd
                                   + (df_final_test['svd_u_s_5']*df_final_test['svd
                                   + (df_final_test['svd_u_s_6']*df_final_test['svd
                                   + (df_final_test['svd_v_s_1']*df_final_test['svd
                                   + (df_final_test['svd_v_s_2']*df_final_test['svd
                                   + (df_final_test['svd_v_s_3']*df_final_test['svd
                                   + (df_final_test['svd_v_s_4']*df_final_test['svd
                                   + (df_final_test['svd_v_s_5']*df_final_test['svd
                                   + (df_final_test['svd_v_s_6']*df_final_test['svd
```

In [73]:

```
df_final_test.columns
```

Out[73]:

```
Index(['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', 'weigh
t_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',
       'followers_Preferential_Attachment',
       'followees_Preferential_Attachment', 'svd_us_dot_ud'],
      dtype='object')
```

Applying ML Models

Random Forest With Hyperparameter tuning

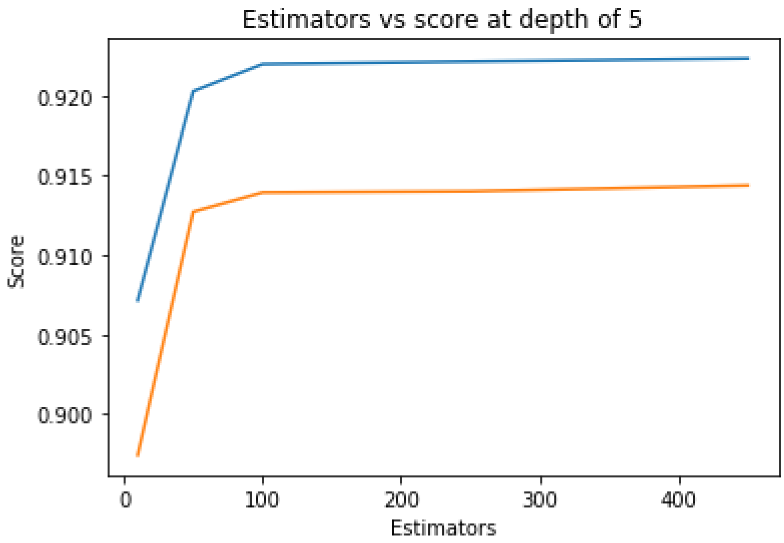
In [74]:

```
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
    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 0.9071442596860371 test Score 0.897395811
9060352
Estimators =  50 Train Score 0.9202947786546803 test Score 0.912736632
3480572
Estimators =  100 Train Score 0.9219963573178136 test Score 0.91394609
42208184
Estimators =  250 Train Score 0.9221604614577871 test Score 0.91402149
46263434
Estimators =  450 Train Score 0.9223525472675338 test Score 0.91438783
14354584
```

Out[74]:

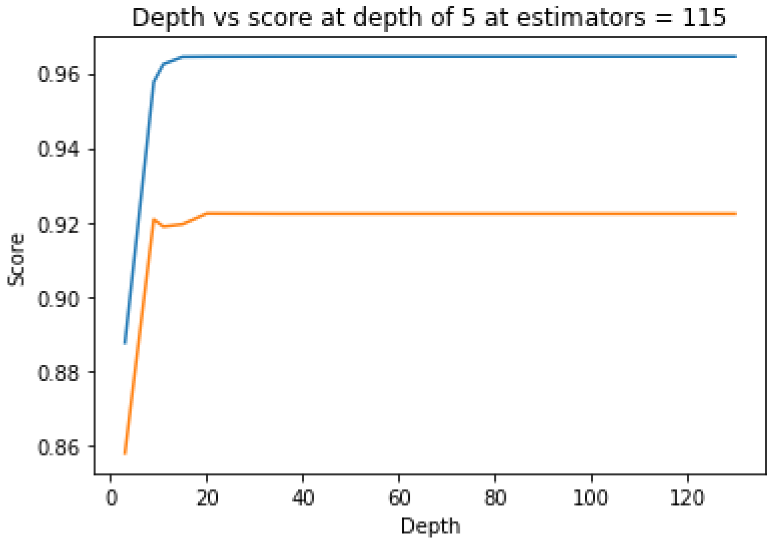
```
Text(0.5, 1.0, 'Estimators vs score at depth of 5')
```



In [75]:

```
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=
    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 0.8875170317576775 test Score 0.857798067735428
4
depth =  9 Train Score 0.9576138788763746 test Score 0.920795584713110
5
depth =  11 Train Score 0.9624283332655633 test Score 0.91883103347646
64
depth =  15 Train Score 0.9643985332805152 test Score 0.91947120397014
05
depth =  20 Train Score 0.9644537004114392 test Score 0.92236322030372
38
depth =  35 Train Score 0.9644924739482825 test Score 0.92228992077190
21
depth =  50 Train Score 0.9644924739482825 test Score 0.92228992077190
21
depth =  70 Train Score 0.9644924739482825 test Score 0.92228992077190
21
depth =  130 Train Score 0.9644924739482825 test Score 0.9222899207719
021
```



In [76]:

```
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(13,20),
              "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)
```

Out[76]:

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=RandomForestClassifier(bootstrap=True,
                                                    ccp_alpha=0.0,
                                                    class_weight=None,
                                                    criterion='gini',
                                                    max_depth=None,
                                                    max_features='aut
o',
                                                    max_leaf_nodes=Non
e,
                                                    max_samples=None,
                                                    min_impurity_decre
ase=0.0,
                                                    min_impurity_split
=None,
                                                    min_samples_leaf=
1,
                                                    min_samples_split=
2,
                                                    min_weight_fractio
n_leaf=0.0,
                                                    n_estimators=100,
n_job...
                                        'min_samples_leaf': <scipy.sta
ts._distn_infrastructure.rv_frozen object at 0x7f5dfe9f5ef0>,
                                        'min_samples_split': <scipy.st
ats._distn_infrastructure.rv_frozen object at 0x7f5dfe9f5278>,
                                        'n_estimators': <scipy.stats._
distn_infrastructure.rv_frozen object at 0x7f5dfe9f5588>},
                   pre_dispatch='2*n_jobs', random_state=25, refit=Tru
e,
                   return_train_score=False, scoring='f1', verbose=0)
```

In [82]:

```
print('mean test scores',rf_random.cv_results_['mean_test_score'])
```

```
mean test scores [0.96342975 0.96371897 0.96234551 0.96300359 0.964445
79]
```

In [83]:

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

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=Non
e,
                       criterion='gini', max_depth=17, max_features='a
uto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=N
one,
                       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, ve
rbose=0,
                       warm_start=False)
```

In [0]:

```
clf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=17, 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 [86]:

```
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 0.9663126687920076
Test f1 score 0.9232209348282819
```

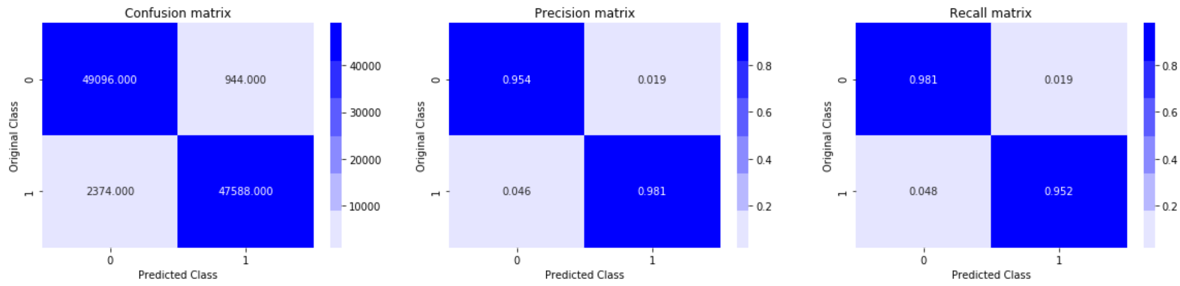
In [0]:

```
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 format26687920076
Test f1 score 0.9232209348282819
    cmap=sns.light_palette("blue")
    plt.subplot(1, 3, 1)
    sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabel
    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, yticklabel
    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, yticklabel
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()
```

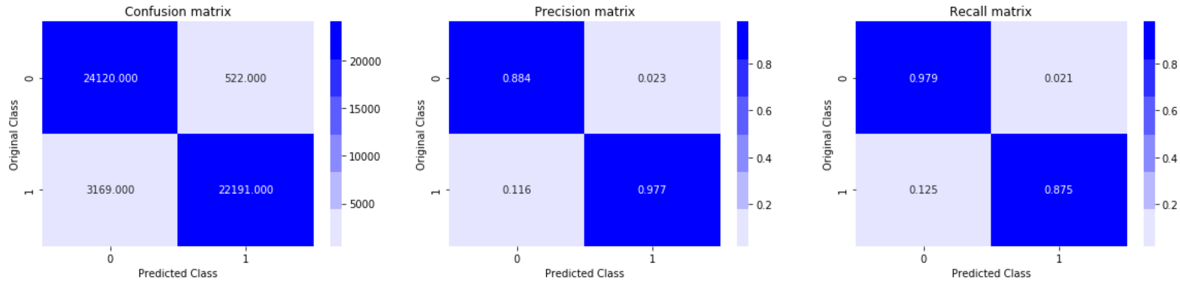
In [88]:

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

```
Train confusion_matrix
```

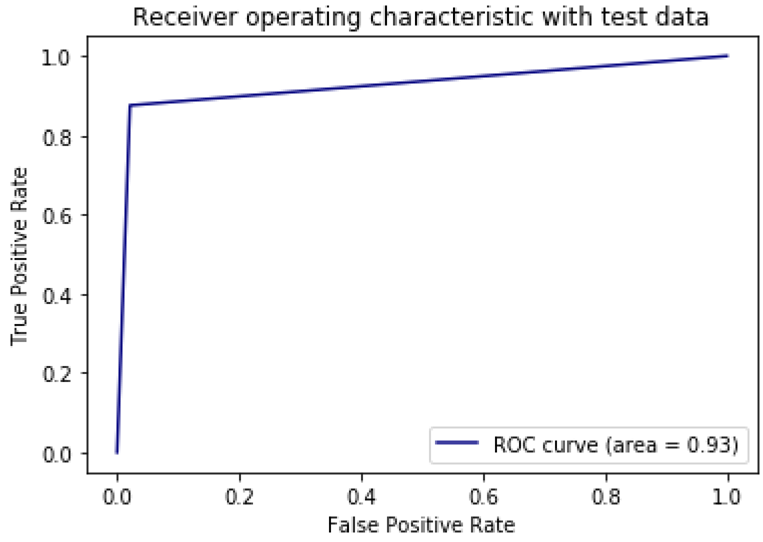


```
Test confusion_matrix
```



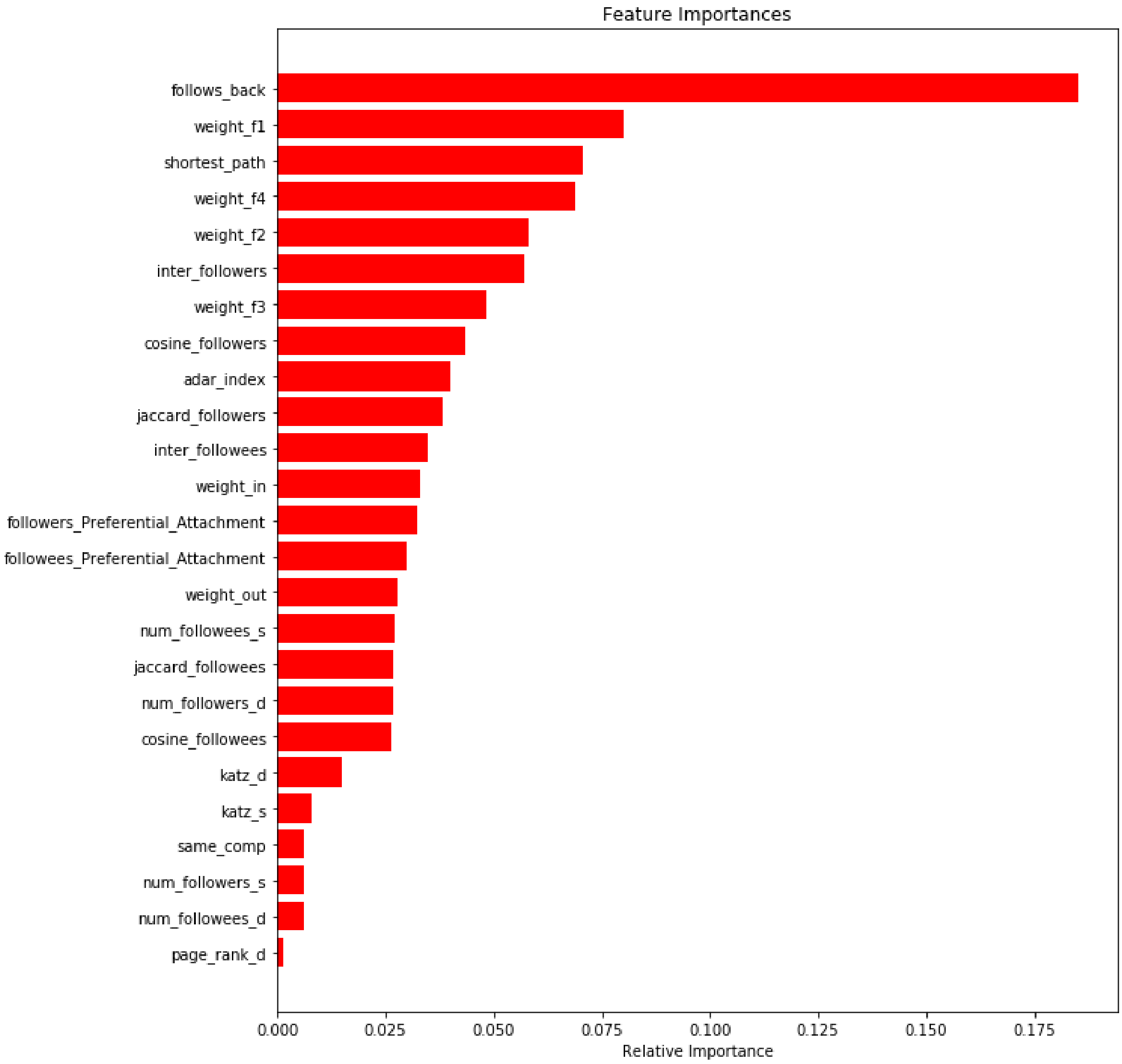
In [89]:

```
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 [90]:

```
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()
```



hyperparameter tuning XG boost

In [91]:

```
import xgboost as xgb
clf = xgb.XGBClassifier()
param_dist = {"n_estimators":sp_randint(105,125),
              "max_depth": sp_randint(14,20)
              }
model = RandomizedSearchCV(clf, param_distributions=param_dist,
                                   n_iter=5,cv=3,scoring='f1',random_state=25)


model.fit(df_final_train,y_train)
```

Out[91]:

```
RandomizedSearchCV(cv=3, error_score=nan,
                   estimator=XGBClassifier(base_score=0.5, booster='gb
tree',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=
0,
                                           learning_rate=0.1, max_delt
a_step=0,
                                           max_depth=3, min_child_weig
ht=1,
                                           missing=None, n_estimators=
100,
                                           n_jobs=1, nthread=None,
                                           objective='binary:logisti
c',
                                           random_state=0, reg_alpha=
0,
                                           reg_lambda=1, sc...
                                           seed=None, silent=None, sub
sample=1,
                                           verbosity=1),
                   iid='deprecated', n_iter=5, n_jobs=None,
                   param_distributions={'max_depth': <scipy.stats._dis
tn_infrastructure.rv_frozen object at 0x7f605d0d4908>,
                                        'n_estimators': <scipy.stats._
distn_infrastructure.rv_frozen object at 0x7f5e0744f3c8>},
                   pre_dispatch='2*n_jobs', random_state=25, refit=Tru
e,
                   return_train_score=False, scoring='f1', verbose=0)
```

In [94]:

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

```
mean test scores [0.97966314 0.97928221 0.97950384 0.97947822 0.979844
43]
```

In [95]:

```
print(model.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=15,
              min_child_weight=1, missing=None, n_estimators=110, n_jo
bs=1,
              nthread=None, objective='binary:logistic', random_state=
0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=Non
e,
              silent=None, subsample=1, verbosity=1)
```

In [0]:

```
clf=xgb.XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=15, min_child_weight=1, missing=None, n_estimators=110,
       n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1)
```

In [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 [99]:

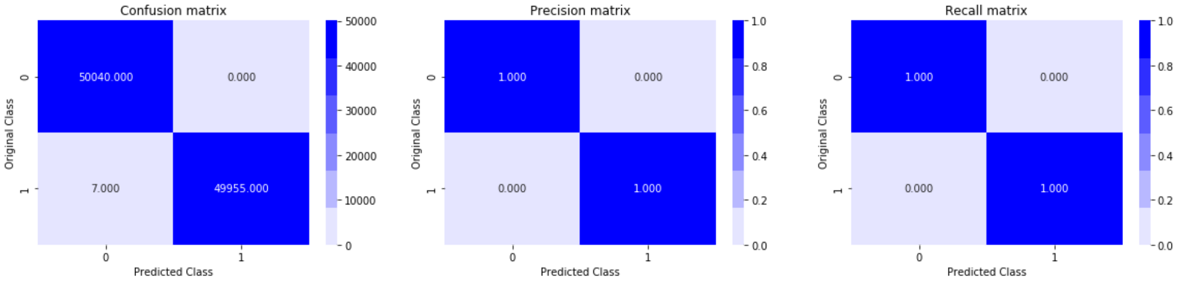
```
print('Train f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))
```

```
Train f1 score 0.9999299418517369
Test f1 score 0.9273415761836021
```

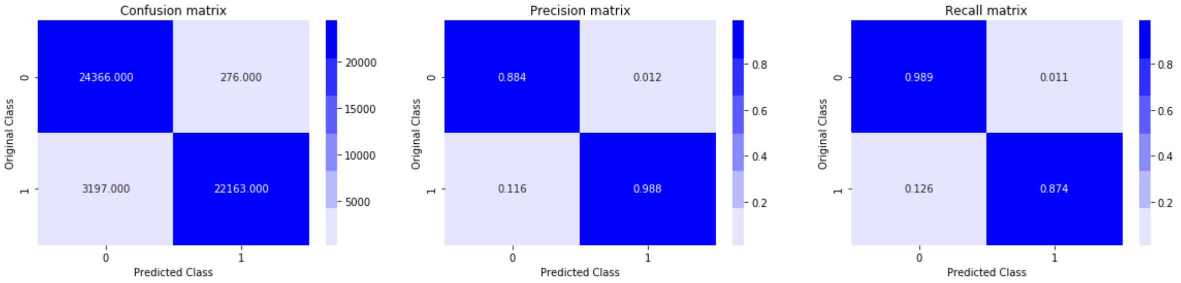
In [100]:

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

```
Train confusion_matrix
```

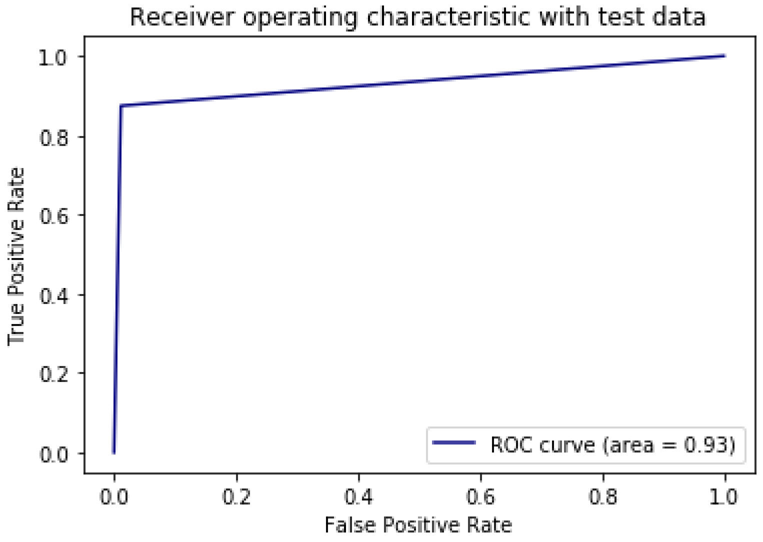


```
Test confusion_matrix
```



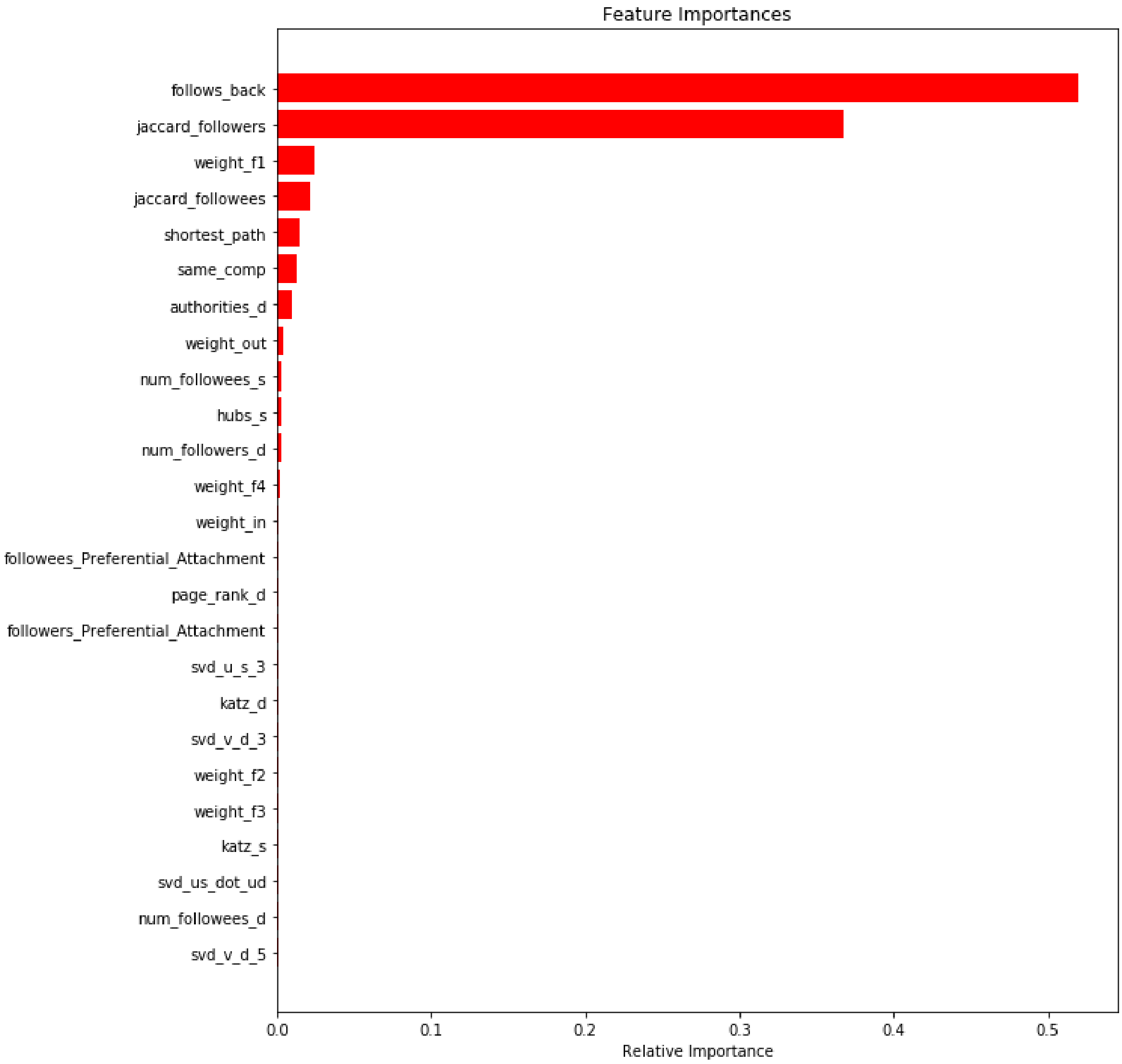
In [101]:

```
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 [102]:

```
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()
```



Procedure:

1. Imported data
2. Performed EDA using Number of followers and followees features generated from graph using networkx library
3. generated edges for making data to supported format for supervised learning models
4. Splitting the data into train and CV(test)
5. Did Feature engineering and created more useful features for prediction(Jaccard Distance ,shortest path ,cosine distance , Ranking ,community,adamic/adar index hits score etc)
6. Added Preferential attachment and svd. features (tasks)

7. Did Hyperamater tuning and then applied Random Forest and XG boost
8. Summarized the details

Summary

In [104]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model", "n_estimators", "max_depth", "Train f1-Score","Test f1-Sc
x.add_row(['Random Forest','121','17','0.966','0.923'])
x.add_row(['XGBOOST','110','15','0.999','0.927'])
print(x)
```

```
+---------------+--------------+-----------+----------------+---------
------+
|     Model     | n_estimators | max_depth | Train f1-Score | Test f1-
Score |
+---------------+--------------+-----------+----------------+---------
------+
| Random Forest |     121      |     17    |     0.966      |     0.92
3     |
|    XGBOOST    |     110      |     15    |     0.999      |     0.92
7     |
+---------------+--------------+-----------+----------------+---------
------+
```

In [0]: