Facebook Recommendation Friend - Using Graph Minning

Business Problem:

Given a directed social graph, we have to predict missing links to recommend friends/connnections/followers (Link Prediction in graph)

Data:

1. train.csv contains the directed social graph, represented in a 2-column csv (source_node, destination_node)
2. test.csv contains a list of nodes to recommend other nodes to in a 1-column csv (source_node)

Total number of Nodes/Vertices - 186220
Total number of Edges/links - 9437519

Machine Learning Problem:

It is a Binary Classification Problem.
0 - No relation between them(No edge/Link) 1 - Relationship exists(Existence of edge/Link)

Business Objectives and Constraints:

```
1. No low latency rate required.
2. Predciting the probability of a link  is useful so as to recommend the highest probability li
nks to a user.
```

Performance Metrics:

F1-Score
Confusion Matrix

Exploratory data Analysis:

In [32]:

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

Visualizing a sub-graph:

In [4]:

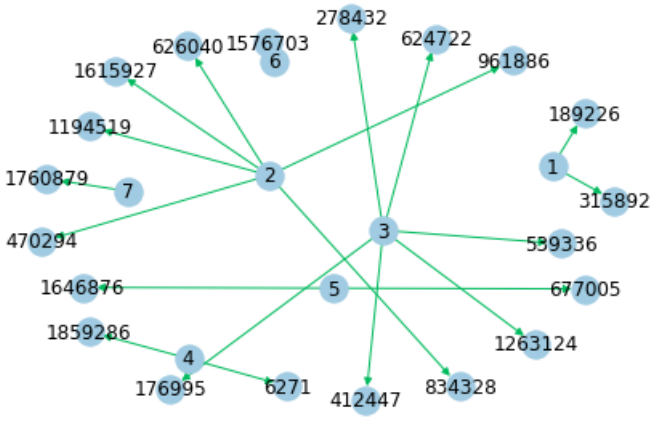
```
import networkx as nx
import os

path = r'C:\Users\Friend\AI\AI_datasets\FacebookFriend\FB\data\after_eda'
src = os.path.join(path,'train_woheader.csv')

g = nx.read_edgelist(src,delimiter=',',create_using=nx.DiGraph(),nodetype=int)
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
```

In [5]:

```
import pandas as pd
from matplotlib import pyplot as plt

if not os.path.isfile(os.path.join(path,'train_woheader_sample.csv')):
    pd.read_csv(src, nrows=20).to_csv(os.path.join(path,'train_woheader_sample.csv'),header=False,index
=False)

subgraph=nx.read_edgelist(os.path.join(path,'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_labe
ls=True)
plt.savefig(os.path.join(path,"graph_sample.pdf"))
print(nx.info(subgraph))
```

```
Name:
Type: DiGraph
Number of nodes: 27
Number of edges: 20
Average in degree:   0.7407
Average out degree:   0.7407
```



Follower Stats:

- 99% of people having 40 or fewer followers.

In [11]:

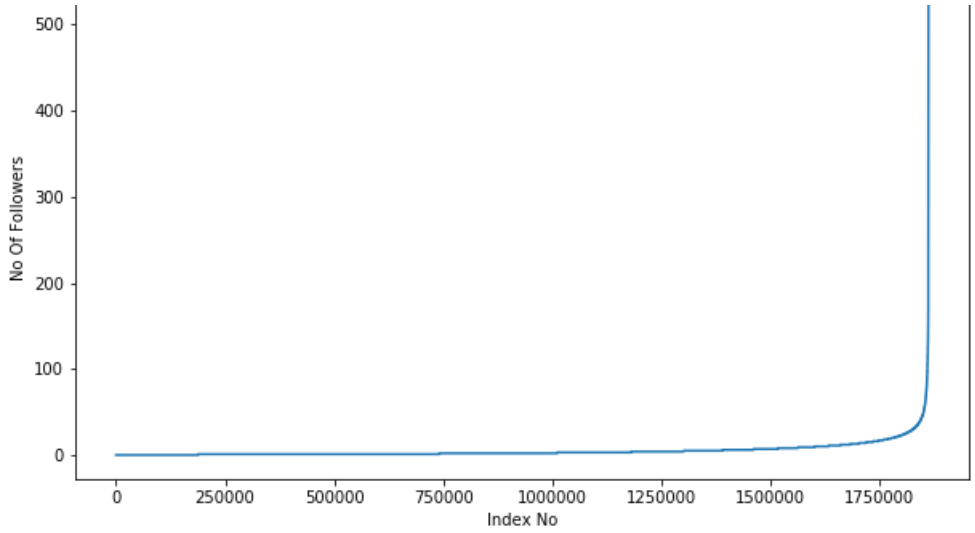
```
# Number of unique persons

len(g.nodes())
```

Out[11]:

```
1862220
```

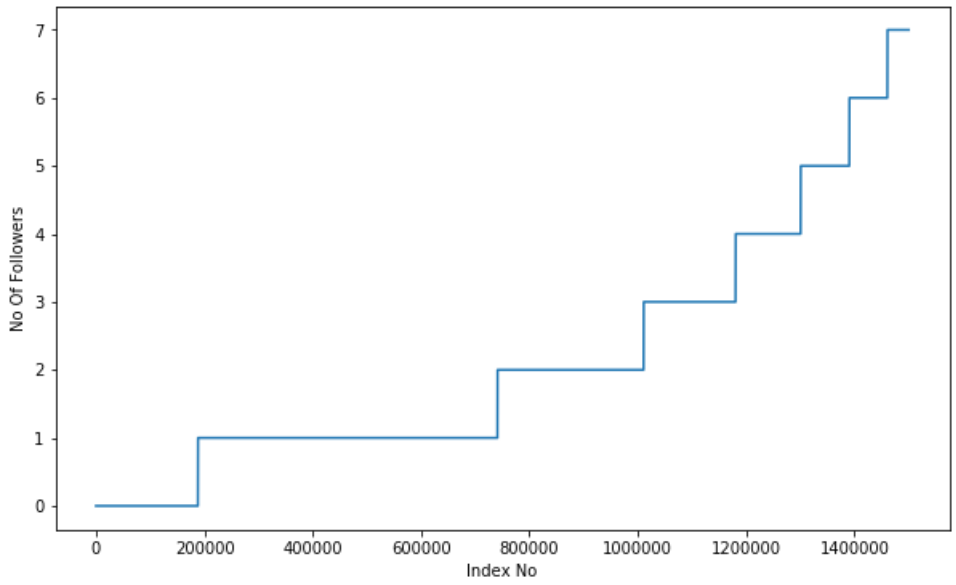
In [15]:

```
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()
```



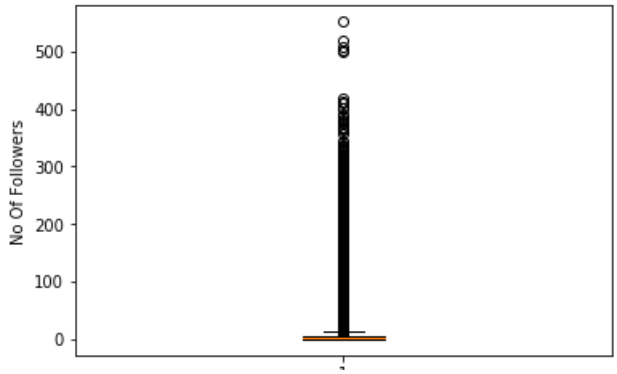
In [16]:

```
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 [17]:

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



In [18]:

```
import numpy as np

### 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
```

In [19]:

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

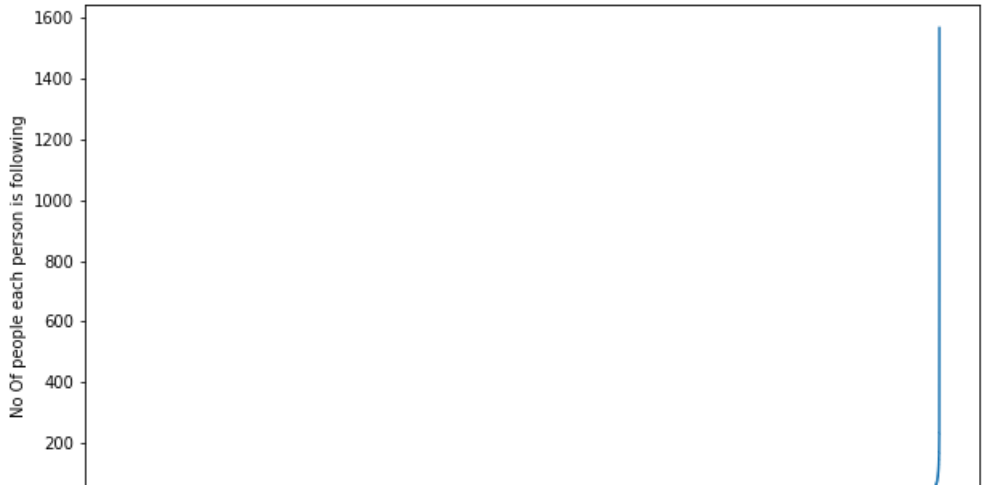
```
99.1 percentile value is 42.0
99.2 percentile value is 44.0
99.3 percentile value is 47.0
99.4 percentile value is 50.0
99.5 percentile value is 55.0
99.6 percentile value is 61.0
99.7 percentile value is 70.0
99.8 percentile value is 84.0
99.9 percentile value is 112.0
100.0 percentile value is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

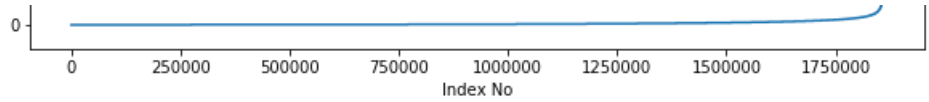
Follower Stats:

- 99% of people follows 40 or fewer.

In [20]:

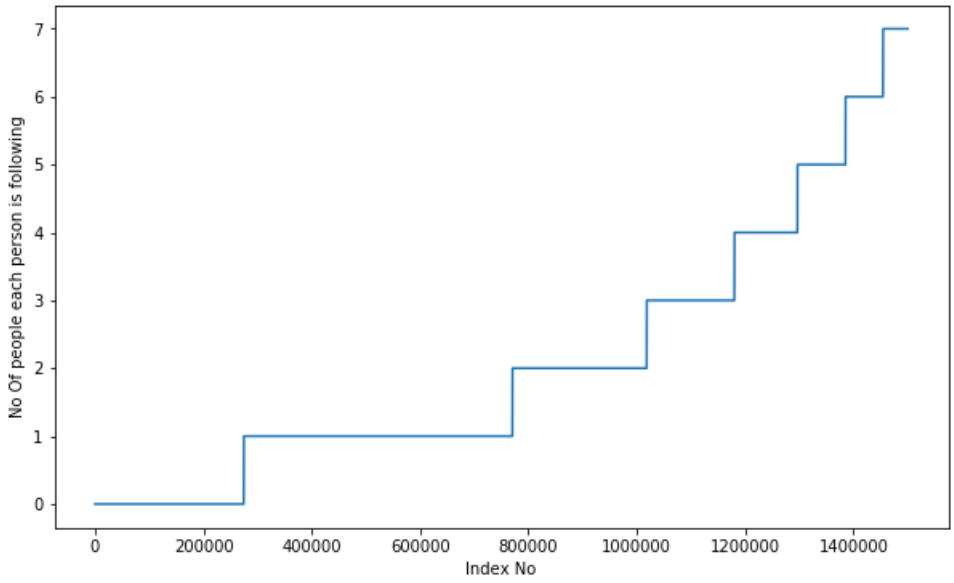
```
outdegree_dist = list(dict(g.out_degree()).values())
outdegree_dist.sort()
plt.figure(figsize=(10,6))
plt.plot(outdegree_dist)
plt.xlabel('Index No')
plt.ylabel('No Of people each person is following')
plt.show()
```





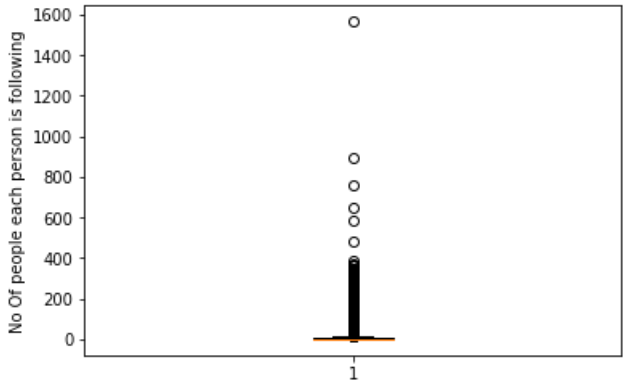
In [21]:

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



In [22]:

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



In [23]:

```
### 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 [24]:

```
### 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
```

Follower + Following Stats:

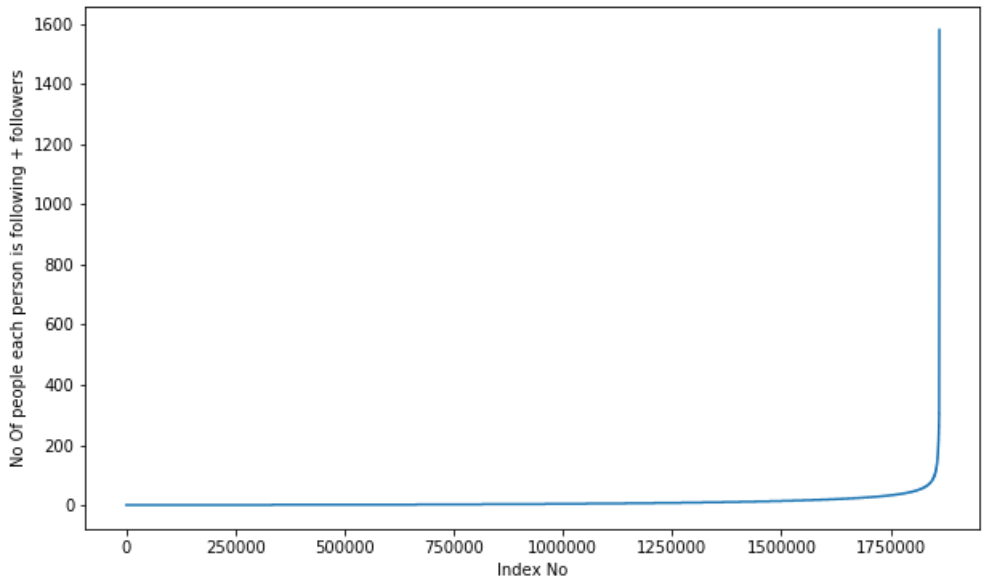
- 99% of people have 79 or fewer in_out counts.

In [26]:

```
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()))
```

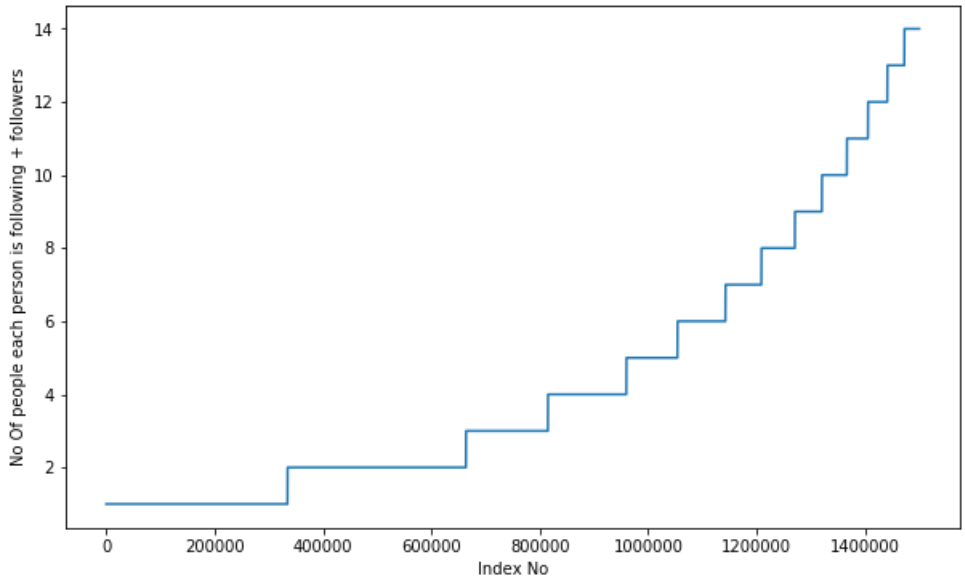
In [27]:

```
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 [28]:

```
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 [29]:

```
### 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 [30]:

```
### 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
```

Data Preparation:

In [ ]:

```
import random
```

```
import pickle

if not os.path.isfile(os.path.join(path,'missing_edges_final.p')):
    #getting all set of edges
    r = csv.reader(open(os.path.join(path,'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(os.path.join(path,'missing_edges_final.p'),'wb'))
else:
    missing_edges = pickle.load(open(os.path.join(path,'missing_edges_final.p'),'rb'))
```

Data Split:

In [ ]:

```
from sklearn.model_selection import train_test_split

if (not os.path.isfile(os.path.join(path,'train_pos_after_eda.csv')) and (not os.path.isfile(os.path.jo
in(path,'test_pos_after_eda.csv'))):
    df_pos = pd.read_csv(r'C:\Users\Friend\AI\AI_datasets\FacebookFriend\FB\data\train.csv')
    df_neg = pd.DataFrame(list(missing_edges), columns=['source_node', 'destination_node'])

    #Trian test split
    #Spiltted data into 80-20
    #positive links and negative links seperatly because we need positive training data only for creati
ng 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)),te
st_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)),t
est_size=0.2, random_state=9)

    X_train_pos.to_csv(os.path.join(path,'train_pos_after_eda.csv'),header=False, index=False)
    X_test_pos.to_csv(os.path.join(path,'test_pos_after_eda.csv'),header=False, index=False)
    X_train_neg.to_csv(os.path.join(path,'train_neg_after_eda.csv'),header=False, index=False)
    X_test_neg.to_csv(os.path.join(path,'test_neg_after_eda.csv'),header=False, index=False)
else:
    #Graph from Traing data only
    del missing_edges
```

In [31]:

```
names = ['source_node', 'destination_node']

X_train_pos = pd.read_csv(os.path.join(path,'train_pos_after_eda.csv'), names=names)
X_train_neg = pd.read_csv(os.path.join(path,'train_neg_after_eda.csv'), names=names)
X_test_pos = pd.read_csv(os.path.join(path,'test_pos_after_eda.csv'), names=names)
X_test_neg = pd.read_csv(os.path.join(path,'test_neg_after_eda.csv'), names=names)
```

In [10]:

```
y_train = pd.read_csv(r'C:\Users\Friend\AI\AI_datasets\FacebookFriend\FB\data\train_y.csv',names=['Labe
ls'])
y_test = pd.read_csv(r'C:\Users\Friend\AI\AI_datasets\FacebookFriend\FB\data\test_y.csv',names=['Labels
'])
```

In [11]:

```
X_train = X_train_pos.append(X_train_neg,ignore_index=True)
X_test = X_test_pos.append(X_test_neg,ignore_index=True)
print(X_train.shape,y_train.shape,X_test.shape,y_test.shape)
```

```
(15100030, 2) (15100030, 1) (3775008, 2) (3775008, 1)
```

Featurizations:

In [ ]:

```
train_graph=nx.read_edgelist(os.path.join(path,'train_pos_after_eda.csv'),delimiter=',',create_using=nx
.DiGraph(),nodetype=int)
nx.info(train_graph)
```

Jaccard Distance:

In [1]:

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


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

Cosine Similarity:

In [2]:

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

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

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

Page Rank:

In [8]:

```
if not os.path.isfile(r'C:\Users\Friend\AI\AI_datasets\FacebookFriend\FB\data\fea_sample\page_rank.p'):
    pr = nx.pagerank(train_graph, alpha=0.85)
    pickle.dump(pr,open(r'C:\Users\Friend\AI\AI_datasets\FacebookFriend\FB\data\fea_sample\page_rank.p'
,'wb'))
else:
    pr = pickle.load(open(r'C:\Users\Friend\AI\AI_datasets\FacebookFriend\FB\data\fea_sample\page_rank.
p','rb'))

mean_pr = float(sum(pr.values())) / len(pr)
```

Shortest Path:

In [ ]:

```
#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
```

Connected Components:

In [ ]:

```
#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
```

Adamic/Adar Index:

In [10]:

```
#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
```

Follows back:

In [11]:

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

Kartz centrality:

In [13]:

```
if not os.path.isfile(r'C:\Users\Friend\AI\AI_datasets\FacebookFriend\FB\data\fea_sample\katz.p'):
    katz = nx.katz.katz_centrality(train_graph,alpha=0.005,beta=1)
    pickle.dump(katz,open(r'C:\Users\Friend\AI\AI_datasets\FacebookFriend\FB\data\fea_sample\katz.p','w
b'))
else:
    katz = pickle.load(open(r'C:\Users\Friend\AI\AI_datasets\FacebookFriend\FB\data\fea_sample\katz.p',
'rb'))
```

Hits Algorithm:

In [14]:

```
if not os.path.isfile(r'C:\Users\Friend\AI\AI_datasets\FacebookFriend\FB\data\fea_sample\hits.p'):
    hits = nx.hits(train_graph, max_iter=100, tol=1e-08, nstart=None, normalized=True)
    pickle.dump(hits,open(r'C:\Users\Friend\AI\AI_datasets\FacebookFriend\FB\data\fea_sample\hits.p','w
b'))
else:
    hits = pickle.load(open(r'C:\Users\Friend\AI\AI_datasets\FacebookFriend\FB\data\fea_sample\hits.p',
'rb'))
```

SVD features:

In [21]:

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

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

In [ ]:

```
#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)}

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

U, s, V = svds(Adj, k = 6)
```

weight features:

In [ ]:

```
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()))
```

Data Preparation:

In [17]:

```
import random

n_train =  15100028
s = 100000
skip_train = sorted(random.sample(range(1,n_train+1),n_train-s))

df_final_train = pd.read_csv(os.path.join(path,'train_after_eda.csv'), skiprows=skip_train, names=['sou
rce_node', 'destination_node'])
df_final_train['indicator_link'] = pd.read_csv(r'C:\Users\Friend\AI\AI_datasets\FacebookFriend\FB\data\
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[17]:

	source_node	destination_node	indicator_link
0	273084	1505602	1
1	196674	421096	1

In [19]:

```
n_test = 3775006
s = 50000
skip_test = sorted(random.sample(range(1,n_test+1),n_test-s))

df_final_test = pd.read_csv(os.path.join(path,'test_after_eda.csv'), skiprows=skip_test, names=['source
_node', 'destination_node'])
df_final_test['indicator_link'] = pd.read_csv(r'C:\Users\Friend\AI\AI_datasets\FacebookFriend\FB\data\t
est_y.csv', skiprows=skip_test, names=['indicator_link'])
```

```
est_y.csv", skiprows=skip_test, names=['indicator_link'])
print("Our test matrix size ",df_final_test.shape)
df_final_test.head(2)
```

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

Out[19]:

	source_node	destination_node	indicator_link
0	848424	784690	1
1	838669	1682749	1

In [ ]:

```
from pandas import read_hdf

if not os.path.isfile(r'C:\Users\Friend\AI\AI_datasets\FacebookFriend\FB\data\fea_sample\storage_sample
_stage1.h5'):
    #mapping jaccrd followers to train and test data
    df_final_train['jaccard_followers'] = df_final_train.apply(lambda row:
                                            jaccard_for_followers(row['source_node'],row['destination_n
ode']),axis=1)
    df_final_test['jaccard_followers'] = df_final_test.apply(lambda row:
                                            jaccard_for_followers(row['source_node'],row['destination_n
ode']),axis=1)

    #mapping jaccrd followees to train and test data
    df_final_train['jaccard_followees'] = df_final_train.apply(lambda row:
                                            jaccard_for_followees(row['source_node'],row['destination_n
ode']),axis=1)
    df_final_test['jaccard_followees'] = df_final_test.apply(lambda row:
                                            jaccard_for_followees(row['source_node'],row['destination_n
ode']),axis=1)


        #mapping cosine followers to train and test data
    df_final_train['cosine_followers'] = df_final_train.apply(lambda row:
                                            cosine_for_followers(row['source_node'],row['destination_no
de']),axis=1)
    df_final_test['cosine_followers'] = df_final_test.apply(lambda row:
                                            cosine_for_followers(row['source_node'],row['destination_no
de']),axis=1)

    #mapping cosine followees to train and test data
    df_final_train['cosine_followees'] = df_final_train.apply(lambda row:
                                            cosine_for_followees(row['source_node'],row['destination_no
de']),axis=1)
    df_final_test['cosine_followees'] = df_final_test.apply(lambda row:
                                            cosine_for_followees(row['source_node'],row['destination_no
de']),axis=1)
```

In [20]:

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

```
            d2=set(train_graph.successors(row['destination_node']))
        except:
            d1 = set()
            d2 = set()
        num_followers_s.append(len(s1))
        num_followees_s.append(len(s2))

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

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

    return num_followers_s, num_followers_d, num_followees_s, num_followees_d, inter_followers, inter_f
ollowees
```

In [ ]:

```
if not os.path.isfile(r'C:\Users\Friend\AI\AI_datasets\FacebookFriend\FB\data\fea_sample\storage_sample
_stage1.h5'):
    df_final_train['num_followers_s'], df_final_train['num_followers_s'], \
    df_final_train['num_followees_s'], df_final_train['num_followees_d'], \
    df_final_train['inter_followers'], df_final_train['inter_followees']= compute_features_stage1(df_fi
nal_train)

    df_final_test['num_followers_s'], df_final_test['num_followers_s'], \
    df_final_test['num_followees_s'], df_final_test['num_followees_d'], \
    df_final_test['inter_followers'], df_final_test['inter_followees']= compute_features_stage1(df_fina
l_test)

    hdf = HDFStore(r'C:\Users\Friend\AI\AI_datasets\FacebookFriend\FB\data\fea_sample\storage_sample_st
age1.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(r'C:\Users\Friend\AI\AI_datasets\FacebookFriend\FB\data\fea_sample\storag
e_sample_stage1.h5', 'train_df',mode='r')
    df_final_test = read_hdf(r'C:\Users\Friend\AI\AI_datasets\FacebookFriend\FB\data\fea_sample\storage
_sample_stage1.h5', 'test_df',mode='r')
```

In [ ]:

```
if not os.path.isfile(r'C:\Users\Friend\AI\AI_datasets\FacebookFriend\FB\data\fea_sample\storage_sample
_stage2.h5'):
    #mapping adar index on train
    df_final_train['adar_index'] = df_final_train.apply(lambda row: calc_adar_in(row['source_node'],row
['destination_node']),axis=1)
    #mapping adar index on test
    df_final_test['adar_index'] = df_final_test.apply(lambda row: calc_adar_in(row['source_node'],row['
destination_node']),axis=1)

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

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

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

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

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

```
    hdf = HDFStore(r'C:\Users\Friend\AI\AI_datasets\FacebookFriend\FB\data\fea_sample\storage_sample_st
age2.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(r'C:\Users\Friend\AI\AI_datasets\FacebookFriend\FB\data\fea_sample\storag
e_sample_stage2.h5', 'train_df',mode='r')
    df_final_test = read_hdf(r'C:\Users\Friend\AI\AI_datasets\FacebookFriend\FB\data\fea_sample\storage
_sample_stage2.h5', 'test_df',mode='r')
```

In [ ]:

```
if not os.path.isfile(r'C:\Users\Friend\AI\AI_datasets\FacebookFriend\FB\data\fea_sample\storage_sample
_stage3.h5'):

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

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

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

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

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

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

    #Hits algorithm score for source and destination in Train and Test
    #if anything not there in train graph then adding 0
    df_final_train['authorities_s'] = df_final_train.source_node.apply(lambda x: hits[1].get(x,0))
    df_final_train['authorities_d'] = df_final_train.destination_node.apply(lambda x: hits[1].get(x,0))

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

    hdf = HDFStore(r'C:\Users\Friend\AI\AI_datasets\FacebookFriend\FB\data\fea_sample\storage_sample_st
age3.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(r'C:\Users\Friend\AI\AI_datasets\FacebookFriend\FB\data\fea_sample\storag
e_sample_stage3.h5', 'train_df',mode='r')
    df_final_test = read_hdf(r'C:\Users\Friend\AI\AI_datasets\FacebookFriend\FB\data\fea_sample\storage
_sample_stage3.h5', 'test_df',mode='r')
```

In [ ]:

```
if not os.path.isfile(r'C:\Users\Friend\AI\AI_datasets\FacebookFriend\FB\data\fea_sample\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_wei
ght_out))
```

```
    #mapping to pandas test
    df_final_test['weight_in'] = df_final_test.destination_node.apply(lambda x: Weight_in.get(x,mean_we
ight_in))
    df_final_test['weight_out'] = df_final_test.source_node.apply(lambda x: Weight_out.get(x,mean_weigh
t_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 [ ]:

```
if not os.path.isfile(r'C:\Users\Friend\AI\AI_datasets\FacebookFriend\FB\data\fea_sample\storage_sample
_stage4.h5'):
    #================================================================================================
=

    df_final_train[['svd_u_s_1', 'svd_u_s_2','svd_u_s_3', 'svd_u_s_4', 'svd_u_s_5', 'svd_u_s_6']] = \
    df_final_train.source_node.apply(lambda x: svd(x, U)).apply(pd.Series)

    df_final_train[['svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3', 'svd_u_d_4', 'svd_u_d_5','svd_u_d_6']] = \
    df_final_train.destination_node.apply(lambda x: svd(x, U)).apply(pd.Series)
    #================================================================================================
=

    df_final_train[['svd_v_s_1','svd_v_s_2', 'svd_v_s_3', 'svd_v_s_4', 'svd_v_s_5', 'svd_v_s_6',]] = \
    df_final_train.source_node.apply(lambda x: svd(x, V.T)).apply(pd.Series)

    df_final_train[['svd_v_d_1', 'svd_v_d_2', 'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5','svd_v_d_6']] = \
    df_final_train.destination_node.apply(lambda x: svd(x, V.T)).apply(pd.Series)
    #================================================================================================
=

    df_final_test[['svd_u_s_1', 'svd_u_s_2','svd_u_s_3', 'svd_u_s_4', 'svd_u_s_5', 'svd_u_s_6']] = \
    df_final_test.source_node.apply(lambda x: svd(x, U)).apply(pd.Series)

    df_final_test[['svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3', 'svd_u_d_4', 'svd_u_d_5','svd_u_d_6']] = \
    df_final_test.destination_node.apply(lambda x: svd(x, U)).apply(pd.Series)

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

    df_final_test[['svd_v_s_1','svd_v_s_2', 'svd_v_s_3', 'svd_v_s_4', 'svd_v_s_5', 'svd_v_s_6',]] = \
    df_final_test.source_node.apply(lambda x: svd(x, V.T)).apply(pd.Series)

    df_final_test[['svd_v_d_1', 'svd_v_d_2', 'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5','svd_v_d_6']] = \
    df_final_test.destination_node.apply(lambda x: svd(x, V.T)).apply(pd.Series)
    #================================================================================================
=

    hdf = HDFStore(r'C:\Users\Friend\AI\AI_datasets\FacebookFriend\FB\data\fea_sample\storage_sample_st
age4.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()
```

Load Data:

In [2]:

```
df_final_train = read_hdf(r'C:\Users\Friend\AI\AI_datasets\FacebookFriend\FB\data\fea_sample\storage_sa
mple_stage4.h5', 'train_df',mode='r')
```

```
df_final_test = read_hdf(r'C:\Users\Friend\AI\AI_datasets\FacebookFriend\FB\data\fea_sample\storage_sample_stage4.h5', 'test_df',mode='r')
```

In [3]:

```
df_final_train.columns
```

Out[3]:

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

In [4]:

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

In [5]:

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

In [6]:

```
df_final_train.columns
```

Out[6]:

```
Index(['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'],
      dtype='object')
```

Machine Learning Models:

In [13]:

```
from scipy.stats import randint as sp_randint
from scipy.stats import uniform
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score,roc_curve, auc
```

In [12]:

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

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

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

    labels = [0,1]
    # representing A in heatmap format
    cmap=sns.light_palette("blue")
    plt.subplot(1, 3, 1)
    sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Confusion matrix")

    plt.subplot(1, 3, 2)
    sns.heatmap(B, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Precision matrix")

    plt.subplot(1, 3, 3)
    # representing B in heatmap format
    sns.heatmap(A, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()
```

In [11]:

```
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 [14]:

```
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 [15]:

```
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)

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

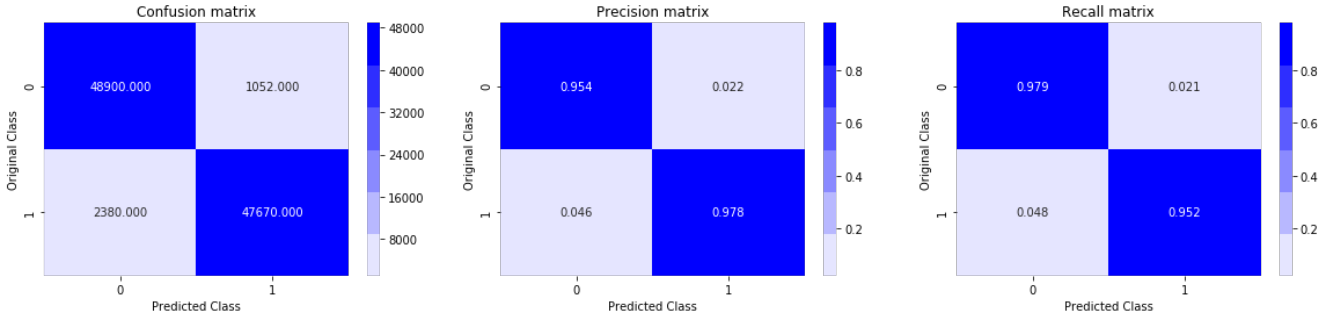
```
print('Train f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))
```

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

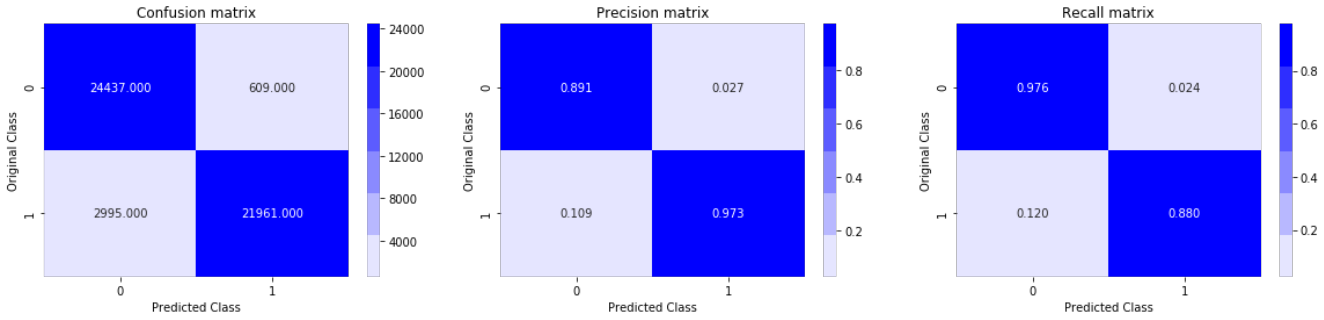
In [16]:

```
#plot confusion matrix

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
```

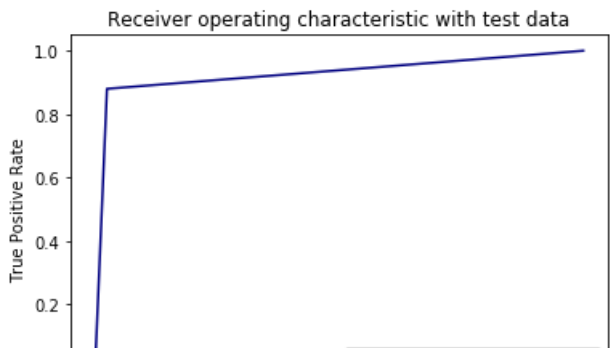


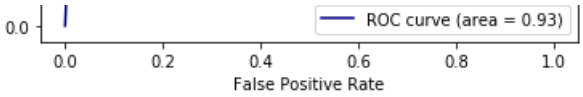

```
Test confusion_matrix
```



In [17]:

```
#plot roc_auc curve

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 [18]:

```
#Feature importance

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()
```