

Social network Graph Link Prediction - Facebook Challenge

In [2]:

```
#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.pyplot as plt
import seaborn as sns #Plots
from matplotlib import rcParams #Size of plots
from sklearn.cluster import MiniBatchKMeans, KMeans #Clustering
import math
import pickle
import os
# to install xgboost: pip3 install xgboost
import xgboost as xgb

import warnings
import networkx as nx
\
import pdb
import pickle
from pandas import HDFStore, DataFrame
from pandas import read_hdf
from scipy.sparse.linalg import svds, eigs
import gc
from tqdm import tqdm
```

1. Reading Data

In [3]:

```
if os.path.isfile('E:/applied ai/module 6/case study 3 facebook/data/after_eda/train_pos_af
    train_graph=nx.read_edgelist('E:/applied ai/module 6/case study 3 facebook/data/after_e
    print(nx.info(train_graph))
else:
    print("please run the FB_EDA.ipynb or download the files from drive")
```

Name:

Type: DiGraph

Number of nodes: 1780722

Number of edges: 7550015

Average in degree: 4.2399

Average out degree: 4.2399

2. Similarity measures

2.1 Jaccard Distance:

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

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

In [4]:

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

In [5]:

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

0.0

In [6]:

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

0.0

In [7]:

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

In [8]:

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

0

In [9]:

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

0

2.2 Cosine distance

$$\text{Cosine Distance} = \frac{|X \cap Y|}{|X| \cdot |Y|}$$

In [10]:

```
#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)))))
    except:
        return 0
```

In [11]:

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

0.0

In [12]:

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

0

In [13]:

```
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_graph.predecessors(b)))))
    except:
        return 0
```

In [14]:

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

0.02886751345948129

In [15]:

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

0

2.3 Preferential Attachment

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

In [16]:

```
#for followees
def preferential_attachment_for_followees(a,b):
    try:
        if len(set(train_graph.successors(a))) == 0 | len(set(train_graph.successors(b))) == 0:
            return 0
        score = (len(set(train_graph.successors(a)))*len(set(train_graph.successors(b))))
        return score
    except:
        return 0
```

In [17]:

```
print(preferential_attachment_for_followees(273084,1505602))
```

120

In [18]:

```
print(preferential_attachment_for_followees(273084,1635354))
```

0

In [19]:

```
#for followers
def preferential_attachment_for_followers(a,b):
    try:
        if len(set(train_graph.predecessors(a))) == 0 | len(set(train_graph.predecessors(b))) == 0:
            return 0
        score = (len(set(train_graph.predecessors(a)))*len(set(train_graph.predecessors(b))))
        return score
    except:
        return 0
```

In [20]:

```
print(preferential_attachment_for_followers(2,470294))
```

60

In [21]:

```
print(preferential_attachment_for_followers(669354,1635354))
```

0

3. Ranking Measures

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

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



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

3.1 Page Ranking

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

In [22]:

```
if not os.path.isfile('E:/applied ai/module 6/case study 3 facebook/data/fea_sample/page_ra
    pr = nx.pagerank(train_graph, alpha=0.85)
    pickle.dump(pr,open('E:/applied ai/module 6/case study 3 facebook/data/fea_sample/page_
else:
    pr = pickle.load(open('E:/applied ai/module 6/case study 3 facebook/data/fea_sample/pag
```

In [23]:

```
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 [24]:

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

5.615699699389075e-07

4. Other Graph Features

4.1 Shortest path:

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

In [25]:

```
#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 [26]:

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

Out[26]:

10

In [27]:

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

Out[27]:

-1

4.2 Checking for same community

In [28]:

```
#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 [29]:

belongs_to_same_wcc(861, 1659750)

Out[29]:

0

In [30]:

belongs_to_same_wcc(669354,1635354)

Out[30]:

0

4.3 Adamic/Adar Index:

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

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

In [31]:

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

In [32]:

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

Out[32]:

0

In [33]:

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

Out[33]:

0

4.4 Is person was following back:

In [34]:

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

In [35]:

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

Out[35]:

1

In [36]:

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

Out[36]:

0

4.5 Katz Centrality:

[https://en.wikipedia.org/wiki/Katz_centrality_\(https://en.wikipedia.org/wiki/Katz_centrality\)](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 λ

The parameter β

controls the initial centrality and

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

In [37]:

```
if not os.path.isfile('E:/applied ai/module 6/case study 3 facebook/data/fea_sample/katz.p'
    katz = nx.katz.katz_centrality(train_graph,alpha=0.005,beta=1)
    pickle.dump(katz,open('E:/applied ai/module 6/case study 3 facebook/data/fea_sample/kat
else:
    katz = pickle.load(open('E:/applied ai/module 6/case study 3 facebook/data/fea_sample/k
```

In [38]:

```
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 [39]:

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

```
0.0007483800935562018
```

4.6 Hits Score

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

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

In [40]:

```
if not os.path.isfile('E:/applied ai/module 6/case study 3 facebook/data/fea_sample/hits.p'):
    hits = nx.hits(train_graph, max_iter=100, tol=1e-08, nstart=None, normalized=True)
    pickle.dump(hits, open('E:/applied ai/module 6/case study 3 facebook/data/fea_sample/hits.p', 'wb'))
else:
    hits = pickle.load(open('E:/applied ai/module 6/case study 3 facebook/data/fea_sample/hits.p', 'rb'))
```

In [41]:

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

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

5. Featurization

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

In [42]:

```
! gdown --id 1lcxzVZ0-MkPmoH31S35Q8rRfrecKSXb1
! gdown --id 1_KN7S8zfHdkRjRYOEtBxBVq8JrGxPXD
```

```
'gdown' is not recognized as an internal or external command,
operable program or batch file.
'gdown' is not recognized as an internal or external command,
operable program or batch file.
```

In []:

In [43]:

```
import random
if os.path.isfile('E:/applied ai/module 6/case study 3 facebook/data/after_eda/train_after_eda.csv'):
    filename = "E:/applied ai/module 6/case study 3 facebook/data/after_eda/train_after_eda.csv"
    # you uncomment this line, if you dont know the length of the file name
    # here we have hardcoded the number of lines as 15100030
    # n_train = sum(1 for line in open(filename)) #number of records in file (excludes header)
    n_train = 15100028
    s = 100000 #desired sample size
    skip_train = sorted(random.sample(range(1,n_train+1),n_train-s))
    #https://stackoverflow.com/a/22259008/4084039
```

In [44]:

```
if os.path.isfile('E:/applied ai/module 6/case study 3 facebook/data/after_eda/train_after_
filename = "E:/applied ai/module 6/case study 3 facebook/data/after_eda/test_after_eda.
# you uncomment this line, if you dont know the length of the file name
# here we have hardcoded the number of lines as 3775008
# n_test = sum(1 for line in open(filename)) #number of records in file (excludes header)
n_test = 3775008
s = 50000 #desired sample size
skip_train = sorted(random.sample(range(1,n_test+1),n_test-s))
#https://stackoverflow.com/a/22259008/4084039
```

In [45]:

```
print("Number of rows in the train data file:", n_train)
print("Number of rows we are going to eliminate 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 eliminate in test data are", len(skip_test))
```

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

In [46]:

```
#https://drive.google.com/file/d/19mviN_yeJIfakb4kU5NfKdQl0QtQ-kH/view?usp=sharing
!gdown --id 19mviN_yeJIfakb4kU5NfKdQl0QtQ-kH
```

'gdown' is not recognized as an internal or external command,
operable program or batch file.

In [47]:

```
#https://drive.google.com/file/d/1H6qybuXr8i_USWu3k3uLXEOurc-SELUh/view?usp=sharing
!gdown --id 1H6qybuXr8i_USWu3k3uLXEOurc-SELUh
```

'gdown' is not recognized as an internal or external command,
operable program or batch file.

In [48]:

```
df_final_train = pd.read_csv('E:/applied ai/module 6/case study 3 facebook/data/after_eda/t
skiprows=skip_train, names=['source_node', 'destination_node']
df_final_train['indicator_link'] = pd.read_csv('E:/applied ai/module 6/case study 3 facebook
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[48]:

	source_node	destination_node	indicator_link
0	273084	1505602	1
1	1076063	58393	1

In [49]:

```
df_final_test = pd.read_csv('E:/applied ai/module 6/case study 3 facebook/data/after_eda/test.csv', skiprows=skip_train, names=['source_node', 'destination_node'])
df_final_test['indicator_link'] = pd.read_csv('E:/applied ai/module 6/case study 3 facebook/data/after_eda/test.csv', skiprows=skip_train, names=['indicator_link'])
print("Our train matrix size ", df_final_test.shape)
df_final_test.head(2)
```

Our train matrix size (25192, 3)

Out[49]:

	source_node	destination_node	indicator_link
0	848424	784690	1
1	1110670	731849	1

5.2 Adding a set of features

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

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

In [50]:

```
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_followees_s,num_followers_d,num_followees_d,inter_followers
```

In []:

In [51]:

```

if not os.path.isfile('E:/applied ai/module 6/case study 3 facebook/data/fea_sample/storage
#mapping Jaccard Distance for followees on train
df_final_train['prefer_score_followees'] =
df_final_train.apply(lambda row: jaccard_for_followees(row['source_node'],row['destination_node']),axis=1)
#mapping Jaccard Distance for followers on train
df_final_train['prefer_score_followers'] =
df_final_train.apply(lambda row: jaccard_for_followers(row['source_node'],row['destination_node']),axis=1)
#mapping Jaccard Distance for followees on train
df_final_test['prefer_score_followees'] =
df_final_test.apply(lambda row: jaccard_for_followees(row['source_node'],row['destination_node']),axis=1)
#mapping Jaccard Distance for followers on train
df_final_test['prefer_score_followers'] =
df_final_test.apply(lambda row: jaccard_for_followers(row['source_node'],row['destination_node']),axis=1)
#=====

#mapping Cosine distance for followees on train
df_final_train['prefer_score_followees'] =
df_final_train.apply(lambda row: cosine_for_followees(row['source_node'],row['destination_node']),axis=1)
#mapping Cosine distance for followers on train
df_final_train['prefer_score_followers'] =
df_final_train.apply(lambda row: cosine_for_followers(row['source_node'],row['destination_node']),axis=1)
#mapping Cosine distance for followees on train
df_final_test['prefer_score_followees'] =
df_final_test.apply(lambda row: cosine_for_followees(row['source_node'],row['destination_node']),axis=1)
#mapping Cosine distance for followers on train
df_final_test['prefer_score_followers'] =
df_final_test.apply(lambda row: cosine_for_followers(row['source_node'],row['destination_node']),axis=1)
#=====

#mapping preferential attachment score for followees on train
df_final_train['prefer_score_followees'] =
df_final_train.apply(lambda row: preferential_attachment_for_followees(row['source_node'],row['destination_node']),axis=1)
#mapping preferential attachment score for followers on train
df_final_train['prefer_score_followers'] =
df_final_train.apply(lambda row: preferential_attachment_for_followers(row['source_node'],row['destination_node']),axis=1)
#mapping preferential attachment score for followees on train
df_final_test['prefer_score_followees'] =
df_final_test.apply(lambda row: preferential_attachment_for_followees(row['source_node'],row['destination_node']),axis=1)
#mapping preferential attachment score for followers on train
df_final_test['prefer_score_followers'] =
df_final_test.apply(lambda row: preferential_attachment_for_followers(row['source_node'],row['destination_node']),axis=1)
#=====

df_final_train['num_followers_s'], df_final_train['num_followers_d'], \
df_final_train['num_followees_s'], df_final_train['num_followees_d'], \
df_final_train['inter_followers'], df_final_train['inter_followees']= compute_features_st
df_final_test['num_followers_s'], df_final_test['num_followers_d'], \
df_final_test['num_followees_s'], df_final_test['num_followees_d'], \
df_final_test['inter_followers'], df_final_test['inter_followees']= compute_features_st
#=====

hdf = HDFStore('E:/applied ai/module 6/case study 3 facebook/data/fea_sample/storage_sample.hdf')
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('E:/applied ai/module 6/case study 3 facebook/data/fea_sample/storage_sample.hdf','train_df',mode='r')
    df_final_test = read_hdf('E:/applied ai/module 6/case study 3 facebook/data/fea_sample/storage_sample.hdf','test_df',mode='r')

```

```
'test_df', mode='r')
```

In [52]:

```
df_final_train.head()
```

Out[52]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_fc
0	273084	1505602	1	0	0.000000	C
1	832016	1543415	1	0	0.187135	C
2	1325247	760242	1	0	0.369565	C
3	1368400	1006992	1	0	0.000000	C
4	140165	1708748	1	0	0.000000	C

5 rows × 33 columns

5.3 Adding new set of features

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

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

In [53]:

```
if not os.path.isfile('E:/applied ai/module 6/case study 3 facebook/data/fea_sample/storage_sa
#mapping adar index on train
df_final_train['adar_index'] = df_final_train.apply(lambda row: calc_adar_in(row['source_n
                                         row['destin
#mapping adar index on test
df_final_test['adar_index'] = df_final_test.apply(lambda row: calc_adar_in(row['source_n
                                         row['destin
#-----
#mapping followback or not on train
df_final_train['follows_back'] = df_final_train.apply(lambda row: follows_back(row['source_n
                                         row['destin
#mapping followback or not on test
df_final_test['follows_back'] = df_final_test.apply(lambda row: follows_back(row['source_n
                                         row['destin
#-----
#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['s
                                         row['destin
##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['s
                                         row['destin
#-----
#mapping shortest path on train
df_final_train['shortest_path'] = df_final_train.apply(lambda row: compute_shortest_pat
    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_
    row['source_node'],row['destination_node']),axis=1)

hdf = HDFStore('E:/applied ai/module 6/case study 3 facebook/data/fea_sample/storage_sa
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('E:/applied ai/module 6/case study 3 facebook/data/fea_sample/
                                'train_df',mode='r')
    df_final_test = read_hdf('E:/applied ai/module 6/case study 3 facebook/data/fea_sample/
                                'test_df',mode='r')
```

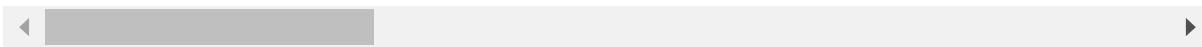
In [54]:

```
df_final_train.head()
```

Out[54]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_fk
0	273084	1505602	1	0	0.000000	C
1	832016	1543415	1	0	0.187135	C
2	1325247	760242	1	0	0.369565	C
3	1368400	1006992	1	0	0.000000	C
4	140165	1708748	1	0	0.000000	C

5 rows × 33 columns



5.4 Adding new set of features

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

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

Weight Features

In order to determine the similarity of nodes, an edge weight value was calculated between nodes. Edge weight decreases as the neighbor count goes up. Intuitively, consider one million people following a celebrity on a social network then chances are most of them never met each other or the celebrity. On the other hand, if a user has 30 contacts in his/her social network, the chances are higher that many of them know each other.

credit - Graph-based Features for Supervised Link Prediction William Cukierski, Benjamin Hamner, Bo Yang

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

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

In [55]:

```
#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 [00:21<00:00, 83700.53it/s]

In [56]:

```
if not os.path.isfile('E:/applied ai/module 6/case study 3 facebook/data/fea_sample/storage
#mapping to pandas train
df_final_train['weight_in'] = df_final_train.destination_node.apply(lambda x: Weight_in.get(x))
df_final_train['weight_out'] = df_final_train.source_node.apply(lambda x: Weight_out.get(x))

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

#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 [57]:

```

if not os.path.isfile('E:/applied ai/module 6/case study 3 facebook/data/fea_sample/storage_sa
#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_p
df_final_train['page_rank_d'] = df_final_train.destination_node.apply(lambda x:pr.get(x,mea
df_final_test['page_rank_s'] = df_final_test.source_node.apply(lambda x:pr.get(x,mean_p
df_final_test['page_rank_d'] = df_final_test.destination_node.apply(lambda x:pr.get(x,mea
#=====

#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_k
df_final_train['katz_d'] = df_final_train.destination_node.apply(lambda x: katz.get(x,mea
df_final_test['katz_s'] = df_final_test.source_node.apply(lambda x: katz.get(x,mean_kat
df_final_test['katz_d'] = df_final_test.destination_node.apply(lambda x: katz.get(x,mea
#=====

#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].ge
df_final_train['authorities_d'] = df_final_train.destination_node.apply(lambda x: hits[1]
df_final_test['authorities_s'] = df_final_test.source_node.apply(lambda x: hits[1].get(x
df_final_test['authorities_d'] = df_final_test.destination_node.apply(lambda x: hits[1]
#=====

hdf = HDFStore('E:/applied ai/module 6/case study 3 facebook/data/fea_sample/storage_sa
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('E:/applied ai/module 6/case study 3 facebook/data/fea_sample/
                                'train_df',mode='r')
    df_final_test = read_hdf('E:/applied ai/module 6/case study 3 facebook/data/fea_sample/
                                'test_df',mode='r')

```

In [58]:

```
df_final_train.head()
```

Out[58]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_fk
0	273084	1505602	1	0	0.000000	C
1	832016	1543415	1	0	0.187135	C
2	1325247	760242	1	0	0.369565	C
3	1368400	1006992	1	0	0.000000	C
4	140165	1708748	1	0	0.000000	C

5 rows × 33 columns



5.5 Adding new set of features

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

1. SVD features for both source and destination
2. Dot product between source node SVD and destination node SVD features

In [59]:

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

In [60]:

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

In [61]:

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

In [62]:

```
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 [65]:

```
if not os.path.isfile('E:/applied ai/module 6/case study 3 facebook/data/fea_sample/storage
#=====

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_
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_
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_
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_
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_
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_
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_
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_
df_final_test.destination_node.apply(lambda x: svd(x, V.T)).apply(pd.Series)
```

In [66]:

```

if not os.path.isfile('E:/applied ai/module 6/case study 3 facebook/data/fea_sample/storage_sa
#Dot product between source node SVD and destination node SVD features on train
df_final_train['svd_dot_u'] = (df_final_train['svd_u_s_1']*df_final_train['svd_u_d_1'])
+(df_final_train['svd_u_s_2']*df_final_train['svd_u_d_2'])
+(df_final_train['svd_u_s_3']*df_final_train['svd_u_d_3'])
+(df_final_train['svd_u_s_4']*df_final_train['svd_u_d_4'])
+(df_final_train['svd_u_s_5']*df_final_train['svd_u_d_5'])
+(df_final_train['svd_u_s_6']*df_final_train['svd_u_d_6'])

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

#=====
#Dot product between source node SVD and destination node SVD features on test
df_final_test['svd_dot_u'] = (df_final_test['svd_u_s_1']*df_final_test['svd_u_d_1'])
+(df_final_test['svd_u_s_2']*df_final_test['svd_u_d_2'])+(df_final_test['svd_u_s_3']*df
+(df_final_test['svd_u_s_4']*df_final_test['svd_u_d_4'])+(df_final_test['svd_u_s_5']*df
+(df_final_test['svd_u_s_6']*df_final_test['svd_u_d_6'])

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

#=====

hdf = HDFStore('E:/applied ai/module 6/case study 3 facebook/data/fea_sample/storage_sa
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 []:

```

# prepared and stored the data from machine Learning models
# please check the FB_Models.ipynb

```

In [67]:

```
df_final_train.head()
```

Out[67]:

...	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
...	8.108439e-13	1.719704e-14	-1.355369e-12	4.675319e-13	1.128586e-06	6.616719e-14	9.771080e-13	4.160
...	3.703512e-12	2.251738e-10	1.244943e-12	-1.636952e-10	-3.112911e-10	6.738902e-02	2.607857e-11	2.372
...	-3.064925e-18	-5.508668e-19	7.375743e-18	6.865057e-18	1.336961e-18	-5.047914e-18	-2.497235e-18	-4.634
...	1.513498e-12	4.498074e-13	-9.818087e-10	3.454680e-11	5.214019e-08	9.595842e-13	3.047045e-10	1.246
...	3.360426e-13	1.407670e-14	-3.453532e-15	1.121299e-15	3.097681e-09	8.173247e-16	1.122939e-15	5.096

In []:

