Social network Graph Link Prediction - Facebook Challenge

In [1]:

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

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

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

1. Reading Data

In [2]:
```
if os.path.isfile('G:\\machine_learning\\case_study\\Case Study 3Facebo
ok Friend Recommendation using Graph Mining\\assignment\\FacebookRecrui
ting\\train_pos_after_eda.csv'):
    train_graph=nx.read_edgelist('G:\\machine_learning\\case_study\\Cas
e Study 3Facebook Friend Recommendation using Graph Mining\\assignment
\\FacebookRecruiting\\train_pos_after_eda.csv',delimiter=',',create_usi
ng=nx.DiGraph(),nodetype=int)
    print(nx.info(train_graph))
else:
    print("please run the FB_EDA.ipynb or download the files from driv
e")
```

```
Name: 
Type: DiGraph
Number of nodes: 88074
Number of edges: 80000
Average in degree:   0.9083
Average out degree:   0.9083
```

2. Similarity measures

2.1 Jaccard Distance:

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

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

In [3]:
```
#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 [4]:
```
#one test case
print(jaccard_for_followees(273084,1505602))
```

0

In [5]:
```
#node 1635354 not in graph
print(jaccard_for_followees(273084,1505602))
```

0

In [6]:
```
#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)))))
        return sim
    except:
        return 0
```

In [7]:
```
print(jaccard_for_followers(273084,470294))
```

0

In [8]:
```
#node 1635354 not in graph
print(jaccard_for_followees(669354,1635354))
```

0

2.2 Cosine distance

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

In [9]:
```
#for followees
def cosine_for_followees(a,b):
    try:
        if len(set(train_graph.successors(a))) == 0  | len(set(train_gr
aph.successors(b))) == 0:
            return 0
        sim = (len(set(train_graph.successors(a)).intersection(set(trai
n_graph.successors(b)))))/\
                                    (math.sqrt(len(set(train_graph.succ
essors(a)))*len((set(train_graph.successors(b))))))
        return sim
    except:
        return 0
```

In [10]:
```
print(cosine_for_followees(273084,1505602))
```

0

In [11]:
```
print(cosine_for_followees(273084,1635354))
```

0

In [12]:
```
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)))))
        return sim
    except:
        return 0
```

In [13]:
```
print(cosine_for_followers(2,470294))
```

```
0
```

In [14]:
```
print(cosine_for_followers(669354,1635354))
```

```
0
```

3. Ranking Measures

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

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

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

3.1 Page Ranking

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

In [15]:

```
if not os.path.isfile('G:\\machine_learning\\case_study\\Case Study 3Facebook Friend Recommendation using Graph Mining\\assignment\\FacebookRecruiting\\page_rank.p'):
    pr = nx.pagerank(train_graph, alpha=0.85)
    pickle.dump(pr,open('G:\\machine_learning\\case_study\\Case Study 3 Facebook Friend Recommendation using Graph Mining\\assignment\\FacebookRecruiting\\page_rank.p','wb'))
else:
    pr = pickle.load(open('G:\\machine_learning\\case_study\\Case Study 3Facebook Friend Recommendation using Graph Mining\\assignment\\FacebookRecruiting\\page_rank.p','rb'))
```

In [16]:

```
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 9.912933221869401e-06
max 3.4595422581020974e-05
mean 1.1354088607303828e-05
```

In [17]:

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

```
1.1354088607303828e-05
```

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 [18]:
```
#if has direct edge then deleting that edge and calculating shortest pa
th
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 [19]:
```
#testing
compute_shortest_path_length(77697, 826021)
```

Out[19]: -1

In [20]:
```
#testing
compute_shortest_path_length(669354,1635354)
```

Out[20]: -1

4.2 Checking for same community

In [21]:
```
#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 [22]:
```
belongs_to_same_wcc(861, 1659750)
```

Out[22]: 0

In [23]:
```
belongs_to_same_wcc(669354,1635354)
```

Out[23]: 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 [24]:
```
#adar index
def calc_adar_in(a,b):
    sum=0
    try:
        n=list(set(train_graph.successors(a)).intersection(set(train_gr
aph.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 [25]:
```
calc_adar_in(1,189226)
```

Out[25]: 0

In [26]:
```
calc_adar_in(669354,1635354)
```

Out[26]: 0

4.4 Is persion was following back:

In [27]:
```
def follows_back(a,b):
```

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

In [28]:
```
follows_back(1,189226)
```

Out[28]: 0

In [29]:
```
follows_back(669354,1635354)
```

Out[29]: 0

4.5 Katz Centrality:

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

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

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

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

$$\lambda$$

.

The parameter

$$\beta$$

controls the initial centrality and

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

In [30]:
```
if not os.path.isfile('G:\\machine_learning\\case_study\\Case Study 3Facebook Friend Recommendation using Graph Mining\\assignment\\FacebookRe
```

```
cruiting\\katz.p'):
    katz = nx.katz.katz_centrality(train_graph,alpha=0.005,beta=1)
    pickle.dump(katz,open('G:\\machine_learning\\case_study\\Case Study
 3Facebook Friend Recommendation using Graph Mining\\assignment\\Facebo
okRecruiting\\katz.p','wb'))
else:
    katz = pickle.load(open('G:\\machine_learning\\case_study\\Case Stu
dy 3Facebook Friend Recommendation using Graph Mining\\assignment\\Face
bookRecruiting\\katz.p','rb'))
```

In [31]:
```
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 0.0033543230149292707
max 0.003507475953016042
mean 0.0033695712468092666
```

In [32]:
```
mean_katz = float(sum(katz.values())) / len(katz)
print(mean_katz)
```

```
0.0033695712468092666
```

4.6 Hits Score

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

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

In [33]:
```
if not os.path.isfile('G:\\machine_learning\\case_study\\Case Study 3Fa
cebook Friend Recommendation using Graph Mining\\assignment\\FacebookRe
cruiting\\hits.p'):
    hits = nx.hits(train_graph, max_iter=100, tol=1e-08, nstart=None, n
ormalized=True)
    pickle.dump(hits,open('G:\\machine_learning\\case_study\\Case Study
 3Facebook Friend Recommendation using Graph Mining\\assignment\\Facebo
```

```
okRecruiting\\hits.p','wb'))
else:
    hits = pickle.load(open('G:\\machine_learning\\case_study\\Case Study 3Facebook Friend Recommendation using Graph Mining\\assignment\\FacebookRecruiting\\hits.p','rb'))
```

In [34]:
```
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 0.7953934352957154
mean 1.1354088607307501e-05
```

5. Featurization

In [35]:
```
import random
if os.path.isfile('G:\\machine_learning\\case_study\\Case Study 3Facebook Friend Recommendation using Graph Mining\\assignment\\FacebookRecruiting\\train_after_eda.csv'):
    filename = "G:\\machine_learning\\case_study\\Case Study 3Facebook Friend Recommendation using Graph Mining\\assignment\\FacebookRecruiting\\train_after_eda.csv"
    # you uncomment this line, if you dont know the lentgh of the file name
    # here we have hardcoded the number of lines as 15100030
    # n_train = sum(1 for line in open(filename)) #number of records in file (excludes header)
    n_train =  15100028
    s = 100000 #desired sample size
    skip_train = sorted(random.sample(range(1,n_train+1),n_train-s))
    #https://stackoverflow.com/a/22259008/4084039
```

In [36]:
```
if os.path.isfile('G:\\machine_learning\\case_study\\Case Study 3Facebook Friend Recommendation using Graph Mining\\assignment\\FacebookRecruiting\\train_after_eda.csv'):
```

```
    filename = "G:\\machine_learning\\case_study\\Case Study 3Facebook
 Friend Recommendation using Graph Mining\\assignment\\FacebookRecruiti
ng\\test_after_eda.csv"
    # you uncomment this line, if you dont know the lentgh of the file
 name
    # here we have hardcoded the number of lines as 3775008
    # n_test = sum(1 for line in open(filename)) #number of records in
 file (excludes header)
    n_test = 3775006
    s = 50000 #desired sample size
    skip_test = sorted(random.sample(range(1,n_test+1),n_test-s))
    #https://stackoverflow.com/a/22259008/4084039
```

In [37]:

```
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(s
kip_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(sk
ip_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 [38]:

```
df_final_train = pd.read_csv('G:\\machine_learning\\case_study\\Case St
udy 3Facebook Friend Recommendation using Graph Mining\\assignment\\Fac
ebookRecruiting\\train_after_eda.csv', skiprows=skip_train, names=['sou
rce_node', 'destination_node'])
df_final_train['indicator_link'] = pd.read_csv('G:\\machine_learning\\c
ase_study\\Case Study 3Facebook Friend Recommendation using Graph Minin
g\\assignment\\FacebookRecruiting\\train_y.csv', skiprows=skip_train, n
ames=['indicator_link'])
print("Our train matrix size ",df_final_train.shape)
df_final_train.head(2)
```

```
Our train matrix size  (1127, 3)
```

Out[38]:

	source_node	destination_node	indicator_link
0	5538	106083	1
1	12757	328730	1

In [39]:

```
df_final_test = pd.read_csv('G:\\machine_learning\\case_study\\Case Study 3Facebook Friend Recommendation using Graph Mining\\assignment\\FacebookRecruiting\\test_after_eda.csv', skiprows=skip_test, names=['source_node', 'destination_node'])
df_final_test['indicator_link'] = pd.read_csv('G:\\machine_learning\\case_study\\Case Study 3Facebook Friend Recommendation using Graph Mining\\assignment\\FacebookRecruiting\\test_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  (529, 3)
```

Out[39]:

	source_node	destination_node	indicator_link
0	11593	1630594	1
1	13501	1159956	1

5.2 Adding a set of features

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

1. jaccard_followers
2. jaccard_followees
3. cosine_followers
4. cosine_followees
5. num_followers_s
6. num_followees_s

7. num_followers_d
8. num_followees_d
9. inter_followers
10. inter_followees

In [40]:
```
if not os.path.isfile('G:\\machine_learning\\case_study\\Case Study 3Fa
cebook Friend Recommendation using Graph Mining\\assignment\\FacebookRe
cruiting\\storage_sample_stage1.h5'):
    #mapping jaccrd followers to train and test data
    df_final_train['jaccard_followers'] = df_final_train.apply(lambda r
ow:
                                            jaccard_for_followers(row[
'source_node'],row['destination_node']),axis=1)
    df_final_test['jaccard_followers'] = df_final_test.apply(lambda row
:
                                            jaccard_for_followers(row[
'source_node'],row['destination_node']),axis=1)

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


        #mapping jaccrd followers to train and test data
    df_final_train['cosine_followers'] = df_final_train.apply(lambda ro
w:
                                            cosine_for_followers(row['s
ource_node'],row['destination_node']),axis=1)
    df_final_test['cosine_followers'] = df_final_test.apply(lambda row:
                                            cosine_for_followers(row['s
ource_node'],row['destination_node']),axis=1)
```

```
    #mapping jaccrd followees to train and test data
    df_final_train['cosine_followees'] = df_final_train.apply(lambda ro
w:
                                            cosine_for_followees(row['s
ource_node'],row['destination_node']),axis=1)
    df_final_test['cosine_followees'] = df_final_test.apply(lambda row:
                                            cosine_for_followees(row['s
ource_node'],row['destination_node']),axis=1)
```

In [41]:
```
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_follo
wees_d, inter_followers, inter_followees
```

In [42]:

```
if not os.path.isfile('G:\\machine_learning\\case_study\\Case Study 3Fa
cebook Friend Recommendation using Graph Mining\\assignment\\FacebookRe
cruiting\\storage_sample_stage1.h5'):
    df_final_train['num_followers_s'], df_final_train['num_followers_d'
], \
    df_final_train['num_followees_s'], df_final_train['num_followees_d'
], \
    df_final_train['inter_followers'], df_final_train['inter_followees'
]= compute_features_stage1(df_final_train)

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

    hdf = HDFStore('G:\\machine_learning\\case_study\\Case Study 3Faceb
ook Friend Recommendation using Graph Mining\\assignment\\FacebookRecru
iting\\storage_sample_stage1.h5')
    hdf.put('train_df',df_final_train, format='table', data_columns=Tru
e)
    hdf.put('test_df',df_final_test, format='table', data_columns=True)
    hdf.close()
else:
    df_final_train = read_hdf('G:\\machine_learning\\case_study\\Case S
tudy 3Facebook Friend Recommendation using Graph Mining\\assignment\\Fa
cebookRecruiting\\storage_sample_stage1.h5', 'train_df',mode='r')
    df_final_test = read_hdf('G:\\machine_learning\\case_study\\Case St
udy 3Facebook Friend Recommendation using Graph Mining\\assignment\\Fac
ebookRecruiting\\storage_sample_stage1.h5', 'test_df',mode='r')
```

5.3 Adding new set of features

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

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

In [43]:

```
if not os.path.isfile('G:\\machine_learning\\case_study\\Case Study 3Facebook Friend Recommendation using Graph Mining\\assignment\\FacebookRecruiting\\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'],row['destination_node']),axis=1)

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

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

    ##mapping same component of wcc or not on train
    df_final_test['same_comp'] = df_final_test.apply(lambda row: belongs_to_same_wcc(row['source_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('G:\\machine_learning\\case_study\\Case Study 3Facebook Friend Recommendation using Graph Mining\\assignment\\FacebookRecruiting\\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('G:\\machine_learning\\case_study\\Case Study 3Facebook Friend Recommendation using Graph Mining\\assignment\\FacebookRecruiting\\storage_sample_stage2.h5', 'train_df',mode='r')
    df_final_test = read_hdf('G:\\machine_learning\\case_study\\Case Study 3Facebook Friend Recommendation using Graph Mining\\assignment\\FacebookRecruiting\\storage_sample_stage2.h5', 'test_df',mode='r')
```

5.4 Adding new set of features

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

1. Weight Features
    - weight of incoming edges
    - weight of outgoing edges
    - weight of incoming edges + weight of outgoing edges
    - weight of incoming edges * weight of outgoing edges
    - 2*weight of incoming edges + weight of outgoing edges

- weight of incoming edges + 2*weight of outgoing edges

2. Page Ranking of source
3. Page Ranking of dest
4. katz of source
5. katz of dest
6. hubs of source
7. hubs of dest
8. authorities_s of source
9. authorities_s of dest

Weight Features

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

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

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

In [44]:
```
#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
```

```python
    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%|██████████████████████████████| 88074/88074 [00:01<00:00, 74373.73it/s]
```

In [45]:

```python
if not os.path.isfile('G:\\machine_learning\\case_study\\Case Study 3Facebook Friend Recommendation using Graph Mining\\assignment\\FacebookRecruiting\\storage_sample_stage3.h5'):
    #mapping to pandas train
    df_final_train['weight_in'] = df_final_train.destination_node.apply(lambda x: Weight_in.get(x,mean_weight_in))
    df_final_train['weight_out'] = df_final_train.source_node.apply(lambda x: Weight_out.get(x,mean_weight_out))

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


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

    #some features engineerings on the in and out weights
    df_final_test['weight_f1'] = df_final_test.weight_in + df_final_tes
```

```
t.weight_out
    df_final_test['weight_f2'] = df_final_test.weight_in * df_final_tes
t.weight_out
    df_final_test['weight_f3'] = (2*df_final_test.weight_in + 1*df_fina
l_test.weight_out)
    df_final_test['weight_f4'] = (1*df_final_test.weight_in + 2*df_fina
l_test.weight_out)
```

In [46]:

```
if not os.path.isfile('G:\\machine_learning\\case_study\\Case Study 3Fa
cebook Friend Recommendation using Graph Mining\\assignment\\FacebookRe
cruiting\\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(la
mbda x:pr.get(x,mean_pr))
    df_final_train['page_rank_d'] = df_final_train.destination_node.app
ly(lambda x:pr.get(x,mean_pr))

    df_final_test['page_rank_s'] = df_final_test.source_node.apply(lamb
da 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(la
mbda 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(lamb
da 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(la
mbda 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(lamb
da 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.a
pply(lambda x: hits[1].get(x,0))

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

    hdf = HDFStore('G:\\machine_learning\\case_study\\Case Study 3Faceb
ook Friend Recommendation using Graph Mining\\assignment\\FacebookRecru
iting\\storage_sample_stage3.h5')
    hdf.put('train_df',df_final_train, format='table', data_columns=Tru
e)
    hdf.put('test_df',df_final_test, format='table', data_columns=True)
    hdf.close()
else:
    df_final_train = read_hdf('G:\\machine_learning\\case_study\\Case S
tudy 3Facebook Friend Recommendation using Graph Mining\\assignment\\Fa
```

```
cebookRecruiting\\storage_sample_stage3.h5', 'train_df',mode='r')
    df_final_test = read_hdf('G:\\machine_learning\\case_study\\Case St
udy 3Facebook Friend Recommendation using Graph Mining\\assignment\\Fac
ebookRecruiting\\storage_sample_stage3.h5', 'test_df',mode='r')
```

In [47]:
```
df_final_train.head()
```

Out[47]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_foll
0	5538	106083	1	0	0.0	
1	4554	1631938	1	0	0.0	
2	10361	464320	1	0	0.0	
3	18646	700984	1	0	0.0	
4	1074	86388	1	0	0.0	

5 rows × 31 columns

Adding new feature Preferential Attachement

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.

Preferential Attachement for followers

In [48]:
```
#Got solution  from internet
```

```
#for train dataset
nfs=np.array(df_final_train['num_followers_s'])
nfd=np.array(df_final_train['num_followers_d'])
preferential_followers=[]
for i in range(len(nfs)):
    preferential_followers.append(nfd[i]*nfs[i])
df_final_train['prefer_Attach_followers']= preferential_followers
df_final_train.head()
```

Out[48]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_foll
0	5538	106083	1	0	0.0	
1	4554	1631938	1	0	0.0	
2	10361	464320	1	0	0.0	
3	18646	700984	1	0	0.0	
4	1074	86388	1	0	0.0	

5 rows × 32 columns

In [49]:

```
#Got solution  from internet
#for test dataset
nfs=np.array(df_final_test['num_followers_s'])
nfd=np.array(df_final_test['num_followers_d'])
preferential_followers=[]
for i in range(len(nfs)):
    preferential_followers.append(nfd[i]*nfs[i])
df_final_test['prefer_Attach_followers']= preferential_followers
df_final_test.head()
```

Out[49]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_foll

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_foll
0	11593	1630594	1	0	0.0	
1	6600	1026592	1	0	0.0	
2	16397	26108	1	0	0.0	
3	16478	249963	1	0	0.0	
4	18578	1528291	1	0	0.0	

5 rows × 32 columns

Preferential Attachement for followees

In [50]:

```
#Got solution  from internet
#for train dataset
nfs=np.array(df_final_train['num_followees_s'])
nfd=np.array(df_final_train['num_followees_d'])
preferential_followees=[]
for i in range(len(nfs)):
    preferential_followees.append(nfd[i]*nfs[i])
df_final_train['prefer_Attach_followees']= preferential_followees
df_final_train.head()
```

Out[50]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_foll
0	5538	106083	1	0	0.0	
1	4554	1631938	1	0	0.0	
2	10361	464320	1	0	0.0	

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_foll
3	18646	700984	1	0	0.0	
4	1074	86388	1	0	0.0	

5 rows × 33 columns

In [51]:

```
#Got solution  from internet
#for test dataset
nfs=np.array(df_final_test['num_followees_s'])
nfd=np.array(df_final_test['num_followees_d'])
preferential_followees=[]
for i in range(len(nfs)):
    preferential_followees.append(nfd[i]*nfs[i])
df_final_test['prefer_Attach_followees']= preferential_followees
df_final_test.head()
```

Out[51]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_foll
0	11593	1630594	1	0	0.0	
1	6600	1026592	1	0	0.0	
2	16397	26108	1	0	0.0	
3	16478	249963	1	0	0.0	
4	18578	1528291	1	0	0.0	

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

In [52]:
```
#Got solution  from internet
def svd(x, S):
    try:
        z = sadj_dict[x]
        return S[z]
    except:
        return [0,0,0,0,0,0]
```

In [53]:
```
#for svd features to get feature vector creating a dict node val and in
edx in svd vector
sadj_col = sorted(train_graph.nodes())
sadj_dict = { val:idx for idx,val in enumerate(sadj_col)}
```

In [54]:
```
Adj = nx.adjacency_matrix(train_graph,nodelist=sorted(train_graph.nodes
())).asfptype()
```

In [55]:
```
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 (88074, 88074)
U Shape (88074, 6)
V Shape (6, 88074)
s Shape (6,)
```

In [56]:
```
#Got solution  from internet
if not os.path.isfile('G:\\machine_learning\\case_study\\Case Study 3Fa
cebook Friend Recommendation using Graph Mining\\assignment\\FacebookRe
```

```
cruiting\\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.Seri
es)

    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.Se
ries)

    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.Serie
s)

    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.Ser
ies)

    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(p
d.Series)
    #===================================================================
==================================

#     hdf = HDFStore('data/fea_sample/storage_sample_stage4.h5')
#     hdf.put('train_df',df_final_train, format='table', data_columns=T
rue)
#     hdf.put('test_df',df_final_test, format='table', data_columns=Tru
e)
#     hdf.close()
```

In [57]:

```
df_final_train.head()
```

Out[57]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_foll
0	5538	106083	1	0	0.0	
1	4554	1631938	1	0	0.0	
2	10361	464320	1	0	0.0	
3	18646	700984	1	0	0.0	
4	1074	86388	1	0	0.0	

5 rows × 57 columns

In [58]:
```
df_final_train.columns
```

Out[58]:
```
Index(['source_node', 'destination_node', 'indicator_link',
       'jaccard_followers', 'jaccard_followees', 'cosine_followers',
       'cosine_followees', 'num_followers_s', 'num_followers_d',
       'num_followees_s', 'num_followees_d', 'inter_followers',
       'inter_followees', 'adar_index', 'follows_back', 'same_comp',
       'shortest_path', 'weight_in', 'weight_out', 'weight_f1', 'weight
_f2',
       'weight_f3', 'weight_f4', 'page_rank_s', 'page_rank_d', 'katz_
s',
       'katz_d', 'hubs_s', 'hubs_d', 'authorities_s', 'authorities_d',
       'prefer_Attach_followers', 'prefer_Attach_followees', '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 feature svd_dot

svd_dot is Dot product between sourse node svd and destination node svd features

In [59]:
```
#for train datasets
s1,s2,s3,s4,s5,s6=df_final_train['svd_u_s_1'],df_final_train['svd_u_s_
2'],df_final_train['svd_u_s_3'],df_final_train['svd_u_s_4'],df_final_tr
ain['svd_u_s_5'],df_final_train['svd_u_s_6']
s7,s8,s9,s10,s11,s12=df_final_train['svd_v_s_1'],df_final_train['svd_v_
s_2'],df_final_train['svd_v_s_3'],df_final_train['svd_v_s_4'],df_final_
train['svd_v_s_5'],df_final_train['svd_v_s_6']

d1,d2,d3,d4,d5,d6=df_final_train['svd_u_d_1'],df_final_train['svd_u_d_
2'],df_final_train['svd_u_d_3'],df_final_train['svd_u_d_4'],df_final_tr
```

```
ain['svd_u_d_5'],df_final_train['svd_u_d_6']
d7,d8,d9,d10,d11,d12=df_final_train['svd_v_d_1'],df_final_train['svd_v_
d_2'],df_final_train['svd_v_d_3'],df_final_train['svd_v_d_4'],df_final_
train['svd_v_d_5'],df_final_train['svd_v_d_6']
```

In [60]:

```
svd_dot=[]
for i in range(len(np.array(s1))):
    a=[]
    b=[]
    a.append(np.array(s1[i]))
    a.append(np.array(s2[i]))
    a.append(np.array(s3[i]))
    a.append(np.array(s4[i]))
    a.append(np.array(s5[i]))
    a.append(np.array(s6[i]))
    a.append(np.array(s7[i]))
    a.append(np.array(s8[i]))
    a.append(np.array(s9[i]))
    a.append(np.array(s10[i]))
    a.append(np.array(s11[i]))
    a.append(np.array(s12[i]))
    b.append(np.array(d1[i]))
    b.append(np.array(d2[i]))
    b.append(np.array(d3[i]))
    b.append(np.array(d4[i]))
    b.append(np.array(d5[i]))
    b.append(np.array(d6[i]))
    b.append(np.array(d7[i]))
    b.append(np.array(d8[i]))
    b.append(np.array(d9[i]))
    b.append(np.array(d10[i]))
    b.append(np.array(d11[i]))
    b.append(np.array(d12[i]))
    svd_dot.append(np.dot(a,b))
df_final_train['svd_dot']=svd_dot
```

In [61]:

```
df_final_train.head()
```

Out[61]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_foll
0	5538	106083	1	0	0.0	
1	4554	1631938	1	0	0.0	
2	10361	464320	1	0	0.0	
3	18646	700984	1	0	0.0	
4	1074	86388	1	0	0.0	

5 rows × 58 columns

In [62]:

```
#for test dataset
s1,s2,s3,s4,s5,s6=df_final_test['svd_u_s_1'],df_final_test['svd_u_s_2'
],df_final_test['svd_u_s_3'],df_final_test['svd_u_s_4'],df_final_test[
'svd_u_s_5'],df_final_test['svd_u_s_6']
s7,s8,s9,s10,s11,s12=df_final_test['svd_v_s_1'],df_final_test['svd_v_s_
2'],df_final_test['svd_v_s_3'],df_final_test['svd_v_s_4'],df_final_test
['svd_v_s_5'],df_final_test['svd_v_s_6']

d1,d2,d3,d4,d5,d6=df_final_test['svd_u_d_1'],df_final_test['svd_u_d_2'
],df_final_test['svd_u_d_3'],df_final_test['svd_u_d_4'],df_final_test[
'svd_u_d_5'],df_final_test['svd_u_d_6']
d7,d8,d9,d10,d11,d12=df_final_test['svd_v_d_1'],df_final_test['svd_v_d_
2'],df_final_test['svd_v_d_3'],df_final_test['svd_v_d_4'],df_final_test
['svd_v_d_5'],df_final_test['svd_v_d_6']
```

In [63]:

```
svd_dot=[]
for i in range(len(np.array(s1))):
    a=[]
    b=[]
    a.append(np.array(s1[i]))
```

```
    a.append(np.array(s2[i]))
    a.append(np.array(s3[i]))
    a.append(np.array(s4[i]))
    a.append(np.array(s5[i]))
    a.append(np.array(s6[i]))
    a.append(np.array(s7[i]))
    a.append(np.array(s8[i]))
    a.append(np.array(s9[i]))
    a.append(np.array(s10[i]))
    a.append(np.array(s11[i]))
    a.append(np.array(s12[i]))
    b.append(np.array(d1[i]))
    b.append(np.array(d2[i]))
    b.append(np.array(d3[i]))
    b.append(np.array(d4[i]))
    b.append(np.array(d5[i]))
    b.append(np.array(d6[i]))
    b.append(np.array(d7[i]))
    b.append(np.array(d8[i]))
    b.append(np.array(d9[i]))
    b.append(np.array(d10[i]))
    b.append(np.array(d11[i]))
    b.append(np.array(d12[i]))
    svd_dot.append(np.dot(a,b))
df_final_test['svd_dot']=svd_dot
```

In [64]:
```
df_final_test.head()
```

Out[64]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_foll
0	11593	1630594	1	0	0.0	
1	6600	1026592	1	0	0.0	
2	16397	26108	1	0	0.0	
3	16478	249963	1	0	0.0	
4	18578	1528291	1	0	0.0	

5 rows × 58 columns

In [65]:
```
hdf = HDFStore('G:\\machine_learning\\case_study\\Case Study 3Facebook
 Friend Recommendation using Graph Mining\\assignment\\FacebookRecruiti
ng\\storage_sample_stage4.h5')
hdf.put('train_df',df_final_train, format='table', data_columns=True)
hdf.put('test_df',df_final_test, format='table', data_columns=True)
hdf.close()
```

In [ ]:

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

In [66]:
```
# prepared and stored the data from machine learning models
# pelase check the FB_Models.ipynb
```

In [ ]: