Social network Graph Link Prediction - Facebook Challenge

In [3]:

```
#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 [93]:

```
if os.path.isfile('train_pos_after_eda.csv'):
    train_graph=nx.read_edgelist('train_pos_after_eda.csv',delimiter=',',create_using=nx
.DiGraph(),nodetype=int)
    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/

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

In [94]:

```
#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 [95]:

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

```
0.0
```

In [96]:

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

```
0.0
```

In [97]:

```
#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 [98]:

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

```
0
```

In [99]:

```
#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 [100]:

```
#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.successor
s(b)))))/\
                                    (math.sqrt(len(set(train_graph.successors(a)))*len((
set(train_graph.successors(b))))))
        return sim
    except:
        return 0
```

In [101]:

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

```
0.0
```

In [102]:

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

```
0
```

In [103]:

```
def cosine_for_followers(a,b):
    try:
        
        if len(set(train_graph.predecessors(a))) == 0  | len(set(train_graph.predecessor
s(b))) == 0:
            return 0
        sim = (len(set(train_graph.predecessors(a)).intersection(set(train_graph.predece
ssors(b)))))/\
                                     (math.sqrt(len(set(train_graph.predecessors(a))))*(
len(set(train_graph.predecessors(b)))))
        return sim
    except:
        return 0
```

In [104]:

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

```
0.02886751345948129
```

In [105]:

```
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

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 [106]:

```
#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 [107]:

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

Out[107]:

```
10
```

In [108]:

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

Out[108]:

```
-1
```

4.2 Checking for same community

In [109]:

```
#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 [110]:

```
len(wcc)
```

Out[110]:

```
48602
```

In [111]:

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

Out[111]:

```
0
```

In [112]:

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

Out[112]:

```
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 [113]:

```
#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 [114]:

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

Out[114]:

```
0
```

In [115]:

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

Out[115]:

```
0
```

4.4 Is persion was following back:

In [116]:

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

In [117]:

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

Out[117]:

```
1
```

In [118]:

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

Out[118]:

```
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 [119]:

```
if not os.path.isfile('katz.p'):
    katz = nx.katz.katz_centrality(train_graph,alpha=0.005,beta=1)
    pickle.dump(katz,open('katz.p','wb'))
```

```
else:
    katz = pickle.load(open('katz.p','rb'))
```

In [120]:

```
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 [121]:

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

```
0.0007483800935562018
```

4.6 Hits Score

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

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

In [122]:

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

In [123]:

```
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 [2]:

```
! gdown --id 1lcxzVZ0-MkPmoH31S35Q8rRfrecKSXb1
! gdown --id 1_KN7S8zfHdrkRjRYOEtBxBVq8JrGxPXD
```

```
Downloading...
From: https://drive.google.com/uc?id=1lcxzVZ0-MkPmoH31S35Q8rRfrecKSXb1
To: /content/train_after_eda.csv
239MB [00:02, 102MB/s]
Downloading...
From: https://drive.google.com/uc?id=1_KN7S8zfHdrkRjRYOEtBxBVq8JrGxPXD
To: /content/test_after_eda.csv
59.7MB [00:00, 184MB/s]
```

In [124]:

```
import random
```

```
if os.path.isfile('train_after_eda.csv'):
    filename = "train_after_eda.csv"
    n_train = sum(1 for line in open(filename))
    print (n_train)
    s = 100000
    print (n_train-s)
    skip_train = sorted(random.sample(range(1,n_train+1),n_train-s))
    print (len(skip_train))
```

```
15100030
15000030
15000030
```

In [125]:

```
import random
if os.path.isfile('test_after_eda.csv'):
    filename = "test_after_eda.csv"
    n_test = sum(1 for line in open(filename))
    print (n_test)
    s = 50000
    print (n_test-s)
    skip_test = sorted(random.sample(range(1,n_test+1),n_test-s))
    print (len(skip_test))
```

```
3775008
3725008
3725008
```

In [126]:

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

```
Number of rows in the train data file: 15100030
Number of rows we are going to elimiate in train data are 15000030
Number of rows in the test data file: 3775008
Number of rows we are going to elimiate in test data are 3725008
```

In [36]:

```
#https://drive.google.com/file/d/19mviN_yeJIfakb4kU5NfKdQlOQtaQ-kH/view?usp=sharing
!gdown --id 19mviN_yeJIfakb4kU5NfKdQlOQtaQ-kH
```

```
/bin/bash: gdown: command not found
```

In [7]:

```
#https://drive.google.com/file/d/1H6qybuXr8i_USWu3k3ulXEOurc-SElUh/view?usp=sharing
!gdown --id 1H6qybuXr8i_USWu3k3ulXEOurc-SElUh
```

```
Downloading...
From: https://drive.google.com/uc?id=1H6qybuXr8i_USWu3k3ulXEOurc-SElUh
To: /content/test_y.csv
11.3MB [00:00, 98.1MB/s]
```

In [175]:

```
df_final_train = pd.read_csv('train_after_eda.csv',skiprows = skip_train ,names=['source_
node', 'destination_node'])
df_final_train['indicator_link'] = pd.read_csv('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  (100001, 3)
```

Out[175]:

source_node	destination_node	indicator_link

	source_node	destination_node	indicator_link
0	273084	1505602	1
1	1677755	298631	1

In [176]:

```
df_final_test = pd.read_csv('test_after_eda.csv', skiprows = skip_test, names=['source_no
de', 'destination_node'])
df_final_test['indicator_link'] = pd.read_csv('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  (50001, 3)
```

Out[176]:

	source_node	destination_node	indicator_link
0	848424	784690	1
1	1475479	1059570	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 [134]:

```
for i,row in df_final_train.iterrows():
    jaccard_followers = []
    jaccard_followees = []
    print (i)
    print (row)
    print("***************************")
    try:
        j1 = jaccard_for_followers(row['source_node'],row['destination_node'])
        j2 = jaccard_for_followees(row['source_node'],row['destination_node'])
    except:
        j1 = set()
        j2 = set()

    print (j1)
    print (j2)
    print("***************************")
    jaccard_followers.append(j1)
    jaccard_followees.append(j2)

    print (jaccard_followers)
    print (jaccard_followees)
    print("***************************")
    break
```

```
0
source_node          273084
destination_node    1505602
indicator_link            1
indicator_link               1
Name: 0, dtype: int64
***************************
0
0.0
***************************
***************************
[0]
[0.0]
```

In [177]:

```python
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
    jaccard_followers = []
    jaccard_followees = []
    cosine_followers = []
    cosine_followees = []
    num_followers_s=[]
    num_followees_s=[]
    num_followers_d=[]
    num_followees_d=[]
    inter_followers=[]
    inter_followees=[]
    for i,row in df_final.iterrows():
        try:
            j1 = jaccard_for_followers(row['source_node'],row['destination_node'])
            j2 = jaccard_for_followees(row['source_node'],row['destination_node'])
        except:
            j1 = set()
            j2 = set()

        try:
            c1 = cosine_for_followers(row['source_node'],row['destination_node'])
            c2 = cosine_for_followees(row['source_node'],row['destination_node'])
        except:
            c1 = set()
            c2 = set()
        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()



        jaccard_followers.append(j1)
        jaccard_followees.append(j2)

        cosine_followers.append(c1)
        cosine_followees.append(c2)

        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 jaccard_followers,jaccard_followees,cosine_followers,cosine_followees,num_followers_s,num_followees_s,num_followers_d,num_followees_d,inter_followers,inter_followees
```

In [178]:

```
# !pip install tables
import tables
```

In [180]:

```
if not os.path.isfile('storage_sample_stage1.h5'):
    df_final_train['jaccard_followers'], df_final_train['jaccard_followees'],\
    df_final_train['cosine_followers'], df_final_train['cosine_followees'], \
    df_final_train['num_followers_s'], df_final_train['num_followees_s'], \
    df_final_train['num_followers_d'], df_final_train['num_followees_d'], \
    df_final_train['inter_followers'], df_final_train['inter_followees']= compute_featur
es_stage1(df_final_train)

    df_final_test['jaccard_followers'], df_final_test['jaccard_followees'], \
    df_final_test['cosine_followers'], df_final_test['cosine_followees'], \
    df_final_test['num_followers_s'], df_final_test['num_followees_s'], \
    df_final_test['num_followers_d'], df_final_test['num_followees_d'], \
    df_final_test['inter_followers'], df_final_test['inter_followees']= compute_features
_stage1(df_final_test)

    hdf = HDFStore('storage_sample_stage1.h5')
    hdf.put('train_df',df_final_train, format='table', data_columns=True)
    hdf.put('test_df',df_final_test, format='table', data_columns=True)
    hdf.close()
else:
    df_final_train = read_hdf('storage_sample_stage1.h5', 'train_df',mode='r')
    df_final_test = read_hdf('storage_sample_stage1.h5', 'test_df',mode='r')
```

In [181]:

```
df_final_train.head()
```

Out[181]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	nun
0	273084	1505602	1	0	0.000000	0.000000	0.000000	
1	1677755	298631	1	0	0.266667	0.192847	0.478091	
2	1014884	740353	1	0	0.000000	0.000000	0.000000	
3	1462461	1600294	1	0	0.153846	0.063888	0.272166	
4	1613640	1313162	1	0	0.000000	0.000000	0.000000	

In [182]:

```
a=df_final_train['num_followers_s'].values
b=df_final_train['num_followers_d'].values
temp = []
for x,y in (zip(a,b)):
    i = 0
    if x==0:
        if y!=0:
            i +=1
            temp.append(i)
print (len(a))
print (len(b))
print (len(temp))
```

```
100001
100001
9176
```

In [183]:

```
# df_final_train[(df_final_train['num_followers_s'] == 0) & (df_final_train['num_follower
s_d'] != 0)]
```

In [184]:

```
np.count_nonzero(a)
```

Out[184]:

```
89437
```

In [185]:

```
np.count_nonzero(b)
```

Out[185]:

```
91515
```

In [186]:

```
# ! gdown --id 1fDJptlCFEWNV5UNGPc4geTykgFI3PDCV
```

In [188]:

```
df_final_train.columns
```

Out[188]:

```
Index(['source_node', 'destination_node', 'indicator_link',
       'jaccard_followers', 'jaccard_followees', 'cosine_followers',
       'cosine_followees', 'num_followers_s', 'num_followees_s',
       'num_followers_d', 'num_followees_d', 'inter_followers',
       'inter_followees'],
      dtype='object')
```

In [163]:

```
# df_final_train = read_hdf('storage_sample_stage4.h5', 'train_df',mode='r')
# df_final_test = read_hdf('storage_sample_stage4.h5', 'test_df',mode='r')
```

In [164]:

```
# df_final_train_new=df_final_train.drop(['num_followers_s','num_followees_s','num_follow
ees_d','inter_followers','inter_followees'],axis=1)
```

In [165]:

```
# df_final_train['num_followers_d']= compute_features_stage1(df_final_train)
```

In [166]:

```
# df_final_train.tail()
```

In [167]:

```
# for val in df_final_train_new['num_followers_s'].values:
#   if(val>0):
#     print(val)
```

In [168]:

```
# https://drive.google.com/file/d/10qJ04GRcaDxc16gmJXb8rpGPmlyys7E2/view?usp=sharing
# ! gdown --id 10qJ04GRcaDxc16gmJXb8rpGPmlyys7E2
```

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 [189]:

```
if not os.path.isfile('storage_sample_stage2.h5'):
    #mapping adar index on train
    df_final_train['adar_index'] = df_final_train.apply(lambda row: calc_adar_in(row['so
urce_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['sour
ce_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['sou
rce_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(r
ow['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_p
ath_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_pat
h_length(row['source_node'],row['destination_node']),axis=1)

    hdf = HDFStore('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('storage_sample_stage2.h5', 'train_df',mode='r')
    df_final_test = read_hdf('storage_sample_stage2.h5', 'test_df',mode='r')
```

In [190]:

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

Out[190]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	nun
0	273084	1505602	1	0	0.000000	0.000000	0.000000	
1	1677755	298631	1	0	0.266667	0.192847	0.478091	

In [191]:

```
df_final_train.columns
```

Out[191]:

```
Index(['source_node', 'destination_node', 'indicator_link',
       'jaccard_followers', 'jaccard_followees', 'cosine_followers',
       'cosine_followees', 'num_followers_s', 'num_followees_s',
       'num_followers_d', 'num_followees_d', 'inter_followers',
       'num_followers_d', 'num_followees_d', 'inter_followers',
       'inter_followees', 'adar_index', 'follows_back', 'same_comp',
       'shortest_path'],
      dtype='object')
```

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 [192]:

```
#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 [00:13<00:00, 134998.65it/s]
```

In [193]:

```
if not os.path.isfile('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_i
n.get(x,mean_weight_in))
    df_final_test['weight_out'] = df_final_test.source_node.apply(lambda x: Weight_out.g
et(x,mean_weight_out))


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

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

In [201]:

```
if not os.path.isfile('page_rank.p'):
    pr = nx.pagerank.pagerank(train_graph,alpha=0.85,max_iter=100, tol=1e-06)
    pickle.dump(pr,open('page_rank.p','wb'))
else:
    pr = pickle.load(open('page_rank.p','rb'))
```

In [202]:

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

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

```
min 1.6556497245737814e-07
max 2.7098251341935827e-05
mean 5.615699699389075e-07
5.615699699389075e-07
```

In [203]:

```
if not os.path.isfile('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,m
ean_pr))
    df_final_train['page_rank_d'] = df_final_train.destination_node.apply(lambda x:pr.ge
t(x,mean_pr))

    df_final_test['page_rank_s'] = df_final_test.source_node.apply(lambda x:pr.get(x,mea
n_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,mea
n_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].g
et(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: hi
ts[1].get(x,0))

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

    hdf = HDFStore('storage_sample_stage3.h5')
    hdf.put('train_df',df_final_train, format='table', data_columns=True)
    hdf.put('test_df',df_final_test, format='table', data_columns=True)
    hdf.close()
else:
    df_final_train = read_hdf('storage_sample_stage3.h5', 'train_df',mode='r')
    df_final_test = read_hdf('storage_sample_stage3.h5', 'test_df',mode='r')
```

In [204]:

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

Out[204]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	nun
0	273084	1505602	1	0	0.000000	0.000000	0.000000	
1	1677755	298631	1	0	0.266667	0.192847	0.478091	

2 rows × 31 columns

In [205]:

```
df_final_train.columns
```

Out[205]:

```
Index(['source_node', 'destination_node', 'indicator_link',
       'jaccard_followers', 'jaccard_followees', 'cosine_followers',
       'cosine_followees', 'num_followers_s', 'num_followees_s',
       'num_followers_d', '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'],
      dtype='object')
```

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 [206]:

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

In [207]:

```
#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 [208]:

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

In [209]:

```
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 [4]:

```
if not os.path.isfile('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('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()

else:
    df_final_train = read_hdf('storage_sample_stage4.h5', 'train_df',mode='r')
    df_final_test = read_hdf('storage_sample_stage4.h5', 'test_df',mode='r')
```

In [5]:

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

Out[5]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	nun
0	273084	1505602	1	0	0.000000	0.000000	0.000000	
1	1677755	298631	1	0	0.266667	0.192847	0.478091	

2 rows × 55 columns

In [6]:

```
df_final_train.columns
```

Out[6]:

```
Index(['source_node', 'destination_node', 'indicator_link',
       'jaccard_followers', 'jaccard_followees', 'cosine_followers',
       'cosine_followees', 'num_followers_s', 'num_followees_s',
       'num_followers_d', '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')
```

Preferential Attachment

Train dataset

1. Preferential Attachment for followers

In [10]:

```
train_followers_source = np.array(df_final_train['num_followers_s'])
train_followers_dest   = np.array(df_final_train['num_followers_d'])
preferential_followers = []
for i in range(len(train_followers_source)):
    preferential_followers.append(train_followers_source[i]*train_followers_dest[i])
df_final_train['preferential_attach_followers']= preferential_followers
df_final_train.head()
```

Out[10]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	nun
0	273084	1505602	1	0	0.000000	0.000000	0.000000	
1	1677755	298631	1	0	0.266667	0.192847	0.478091	
2	1014884	740353	1	0	0.000000	0.000000	0.000000	
3	1462461	1600294	1	0	0.153846	0.063888	0.272166	
4	1613640	1313162	1	0	0.000000	0.000000	0.000000	

5 rows × 57 columns

2. Preferential Attachment for followees

In [9]:

```
train_followees_source = np.array(df_final_train['num_followees_s'])
train_followees_dest   = np.array(df_final_train['num_followees_d'])
preferential_followees=[]
for i in range(len(train_followees_source)):
    preferential_followees.append(train_followees_source[i]*train_followees_dest[i])
df_final_train['preferential_attach_followees']= preferential_followees
df_final_train.head()
```

Out[9]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	nun
0	273084	1505602	1	0	0.000000	0.000000	0.000000	
1	1677755	298631	1	0	0.266667	0.192847	0.478091	
2	1014884	740353	1	0	0.000000	0.000000	0.000000	
3	1462461	1600294	1	0	0.153846	0.063888	0.272166	

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	nun
4	1613640	1313162	1	0	0.000000	0.000000	0.000000	

5 rows × 57 columns

Test dataset

1. Preferential Attachment for followers

In [11]:

```
test_followers_source = np.array(df_final_test['num_followers_s'])
test_followers_dest   = np.array(df_final_test['num_followers_d'])
preferential_followers = []
for i in range(len(test_followers_source)):
    preferential_followers.append(test_followers_source[i]*test_followers_dest[i])
df_final_test['preferential_attach_followers']= preferential_followers
df_final_test.head()
```

Out[11]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	nun
0	848424	784690	1	0	0.000000	0.029161	0.000000	
1	1475479	1059570	1	0	0.136364	0.090722	0.300000	
2	1033369	1457439	1	0	0.000000	0.000000	0.000000	
3	278731	399191	1	0	0.000000	0.000000	0.000000	
4	1425933	230726	1	0	0.631579	0.144338	0.774597	

5 rows × 56 columns

2. Preferential Attachment for followees

In [12]:

```
test_followees_source = np.array(df_final_test['num_followees_s'])
test_followees_dest   = np.array(df_final_test['num_followees_d'])
preferential_followees=[]
for i in range(len(test_followees_source)):
    preferential_followees.append(test_followees_source[i]*test_followees_dest[i])
df_final_test['preferential_attach_followees']= preferential_followees
df_final_test.head()
```

Out[12]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	nun
0	848424	784690	1	0	0.000000	0.029161	0.000000	
1	1475479	1059570	1	0	0.136364	0.090722	0.300000	

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	nun
2	1033369	1457439	1	0	0.000000	0.000000	0.000000	
3	278731	399191	1	0	0.000000	0.000000	0.000000	
4	1425933	230726	1	0	0.631579	0.144338	0.774597	

5 rows × 57 columns

Adding feature svd_dot

Train Dataset

In [17]:

```
source_u_1=df_final_train['svd_u_s_1']
source_u_2=df_final_train['svd_u_s_2']
source_u_3=df_final_train['svd_u_s_3']
source_u_4=df_final_train['svd_u_s_4']
source_u_5=df_final_train['svd_u_s_5']
source_u_6=df_final_train['svd_u_s_6']
source_v_1=df_final_train['svd_v_s_1']
source_v_2=df_final_train['svd_v_s_2']
source_v_3=df_final_train['svd_v_s_3']
source_v_4=df_final_train['svd_v_s_4']
source_v_5=df_final_train['svd_v_s_5']
source_v_6=df_final_train['svd_v_s_6']

dest_u_1=df_final_train['svd_u_d_1']
dest_u_2=df_final_train['svd_u_d_2']
dest_u_3=df_final_train['svd_u_d_3']
dest_u_4=df_final_train['svd_u_d_4']
dest_u_5=df_final_train['svd_u_d_5']
dest_u_6=df_final_train['svd_u_d_6']
dest_v_1=df_final_train['svd_v_d_1']
dest_v_2=df_final_train['svd_v_d_2']
dest_v_3=df_final_train['svd_v_d_3']
dest_v_4=df_final_train['svd_v_d_4']
dest_v_5=df_final_train['svd_v_d_5']
dest_v_6=df_final_train['svd_v_d_6']
```

In [18]:

```
svd_dot_train = []
for i in range(len(np.array(source_u_1))):
    source = []
    dest   = []
    source.append(np.array(source_u_1[i]))
    source.append(np.array(source_u_2[i]))
    source.append(np.array(source_u_3[i]))
    source.append(np.array(source_u_4[i]))
    source.append(np.array(source_u_5[i]))
    source.append(np.array(source_u_6[i]))
    source.append(np.array(source_v_1[i]))
    source.append(np.array(source_v_2[i]))
    source.append(np.array(source_v_3[i]))
    source.append(np.array(source_v_4[i]))
    source.append(np.array(source_v_5[i]))
    source.append(np.array(source_v_6[i]))

    dest.append(np.array(dest_u_1[i]))
    dest.append(np.array(dest_u_2[i]))
    dest.append(np.array(dest_u_3[i]))
    dest.append(np.array(dest_u_4[i]))
    dest.append(np.array(dest_u_5[i]))
    dest.append(np.array(dest_u_6[i]))
    dest.append(np.array(dest_v_1[i]))
    dest.append(np.array(dest_v_2[i]))
    dest.append(np.array(dest_v_3[i]))
    dest.append(np.array(dest_v_4[i]))
    dest.append(np.array(dest_v_5[i]))
    dest.append(np.array(dest_v_6[i]))

    svd_dot_train.append(np.dot(source,dest))
df_final_train['svd_dot_train']=svd_dot_train
```

In [19]:

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

Out[19]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	nun
0	273084	1505602	1	0	0.000000	0.000000	0.000000	
1	1677755	298631	1	0	0.266667	0.192847	0.478091	

2 rows × 58 columns

Test Dataset

In [20]:

```
source_u_1=df_final_test['svd_u_s_1']
source_u_2=df_final_test['svd_u_s_2']
source_u_3=df_final_test['svd_u_s_3']
source_u_4=df_final_test['svd_u_s_4']
source_u_5=df_final_test['svd_u_s_5']
source_u_6=df_final_test['svd_u_s_6']
source_v_1=df_final_test['svd_v_s_1']
source_v_2=df_final_test['svd_v_s_2']
source_v_3=df_final_test['svd_v_s_3']
source_v_4=df_final_test['svd_v_s_4']
source_v_5=df_final_test['svd_v_s_5']
source_v_6=df_final_test['svd_v_s_6']

dest_u_1=df_final_test['svd_u_d_1']
dest_u_2=df_final_test['svd_u_d_2']
dest_u_3=df_final_test['svd_u_d_3']
dest_u_4=df_final_test['svd_u_d_4']
dest_u_5=df_final_test['svd_u_d_5']
dest_u_6=df_final_test['svd_u_d_6']
dest_v_1=df_final_test['svd_v_d_1']
dest_v_2=df_final_test['svd_v_d_2']
dest_v_3=df_final_test['svd_v_d_3']
dest_v_4=df_final_test['svd_v_d_4']
dest_v_5=df_final_test['svd_v_d_5']
dest_v_6=df_final_test['svd_v_d_6']
```

In [21]:

```
svd_dot_test = []
for i in range(len(np.array(source_u_1))):
    source = []
    dest   = []
    source.append(np.array(source_u_1[i]))
    source.append(np.array(source_u_2[i]))
    source.append(np.array(source_u_3[i]))
    source.append(np.array(source_u_4[i]))
    source.append(np.array(source_u_5[i]))
    source.append(np.array(source_u_6[i]))
    source.append(np.array(source_v_1[i]))
    source.append(np.array(source_v_2[i]))
    source.append(np.array(source_v_3[i]))
    source.append(np.array(source_v_4[i]))
    source.append(np.array(source_v_5[i]))
    source.append(np.array(source_v_6[i]))

    dest.append(np.array(dest_u_1[i]))
    dest.append(np.array(dest_u_2[i]))
    dest.append(np.array(dest_u_3[i]))
    dest.append(np.array(dest_u_4[i]))
    dest.append(np.array(dest_u_5[i]))
    dest.append(np.array(dest_u_6[i]))
    dest.append(np.array(dest_v_1[i]))
    dest.append(np.array(dest_v_2[i]))
    dest.append(np.array(dest_v_3[i]))
    dest.append(np.array(dest_v_4[i]))
    dest.append(np.array(dest_v_5[i]))
    dest.append(np.array(dest_v_6[i]))

    svd_dot_test.append(np.dot(source,dest))
df_final_test['svd_dot_test']=svd_dot_test
```

In [22]:

```
df_final_test.head(2)
```

Out[22]:

	source_node	destination_node	indicator_link	jaccard_followers	jaccard_followees	cosine_followers	cosine_followees	nun
0	848424	784690	1	0	0.000000	0.029161	0.0	
1	1475479	1059570	1	0	0.136364	0.090722	0.3	

2 rows × 58 columns

In [23]:

```
hdf = HDFStore('storage_sample_stage5.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 [24]:

```
# df_final_train = read_hdf('storage_sample_stage4.h5', 'train_df',mode='r')
# df_final_test = read_hdf('storage_sample_stage4.h5', 'test_df',mode='r')
```

In [ ]: