Social network Graph Link Prediction - Facebook Challenge

In [1]:

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

```
Mounted at /content/drive
```

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 [4]:

```
if os.path.isfile('/content/drive/MyDrive/Facebook/data/after_eda/train_pos_after_eda.csv'):
    train_graph=nx.read_edgelist('/content/drive/MyDrive/Facebook/data/after_eda/train_pos_after_eda.csv',delimit
    print(nx.info(train_graph))
else:
    print("please run the FB_EDA.ipynb or download the files from drive")
```

```
DiGraph with 1780722 nodes and 7550015 edges
```

2. Similarity measures

2.1 Jaccard Distance:

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

j = |X ∩ Y| |X ∪ Y|

In [5]:

```
#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 [6]:
```
#one test case
print(jaccard_for_followees(273084,1505602))
```

```
0.0
```

In [7]:
```
#node 1635354 not in graph
print(jaccard_for_followees(273084,1505602))
```

```
0.0
```

In [8]:
```
#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 [9]:
```
print(jaccard_for_followers(273084,470294))
```

```
0
```

In [10]:
```
#node 1635354 not in graph
print(jaccard_for_followees(669354,1635354))
```

```
0
```

2.2 Cosine distance

CosineDistance = $|X \cap Y||X| \cdot |Y|$

In [11]:
```
#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.successor
        return sim
    except:
        return 0
```

In [12]:
```
print(cosine_for_followees(273084,1505602))
```

```
0.0
```

In [13]:
```
print(cosine_for_followees(273084,1635354))
```

```
0
```

In [14]:
```
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.prede
        return sim
    except:
        return 0
```

In [15]:
```
print(cosine_for_followers(2,470294))
```

```
0.02886751345948129
```

In [16]:

```
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 [17]:

```
if not os.path.isfile('/content/drive/MyDrive/Facebook/data/fea_sample/page_rank.p'):
    pr = nx.pagerank(train_graph, alpha=0.85)
    pickle.dump(pr,open('page_rank.p','wb'))
else:
    pr = pickle.load(open('/content/drive/MyDrive/Facebook/data/fea_sample/page_rank.p','rb'))
```

In [18]:

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

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

In [19]:

```
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 [20]:

```
#if has direct edge then deleting that edge and calculating shortest path
def compute_shortest_path_length(a,b):
    p=-1
    try:
        if train_graph.has_edge(a,b):
            train_graph.remove_edge(a,b)
            p= nx.shortest_path_length(train_graph,source=a,target=b)
            train_graph.add_edge(a,b)
        else:
            p= nx.shortest_path_length(train_graph,source=a,target=b)
        return p
    except:
        return -1
```

In [21]:

```
#testing
```

```
compute_shortest_path_length(77697, 826021)
```

Out[21]:

```
10
```

In [22]:

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

Out[22]:

```
-1
```

4.2 Checking for same community

In [23]:

```python
#getting weekly connected edges from graph
wcc=list(nx.weakly_connected_components(train_graph))
def belongs_to_same_wcc(a,b):
    index = []
    if train_graph.has_edge(b,a):
        return 1
    if train_graph.has_edge(a,b):
            for i in wcc:
                if a in i:
                    index= i
                    break
            if (b in index):
                train_graph.remove_edge(a,b)
                if compute_shortest_path_length(a,b)==-1:
                    train_graph.add_edge(a,b)
                    return 0
                else:
                    train_graph.add_edge(a,b)
                    return 1
            else:
                return 0
    else:
            for i in wcc:
                if a in i:
                    index= i
                    break
            if(b in index):
                return 1
            else:
                return 0
```

In [24]:

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

Out[24]:

```
0
```

In [25]:

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

Out[25]:

```
0
```

4.3 Adamic/Adar Index:

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

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

In [26]:

```python
#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 [27]:

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

Out[27]:

```
0
```

In [28]:

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

Out[28]:

```
0
```

4.4 Is persion was following back:

In [29]:

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

In [30]:

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

Out[30]:

```
1
```

In [31]:

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

Out[31]:

```
0
```

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 λ.

The parameter β controls the initial centrality and

$\alpha < {}^{1\lambda}{}_{max}$.

In [32]:

```
if not os.path.isfile('/content/drive/MyDrive/Facebook/data/fea_sample/katz.p'):
    katz = nx.katz.katz_centrality(train_graph,alpha=0.005,beta=1)
    pickle.dump(katz,open('/content/drive/MyDrive/Facebook/data/fea_sample/katz.p','wb'))
else:
    katz = pickle.load(open('/content/drive/MyDrive/Facebook/data/fea_sample/katz.p','rb'))
```

In [33]:

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

```
min 0.0007313532484065916
max 0.003394554981699122
mean 0.0007483800935562018
```

In [34]:

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

```
0.0007483800935562018
```

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 [35]:

```
if not os.path.isfile('/content/drive/MyDrive/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('/content/drive/MyDrive/Facebook/data/fea_sample/hits.p','wb'))
else:
    hits = pickle.load(open('/content/drive/MyDrive/Facebook/data/fea_sample/hits.p','rb'))
```

In [36]:

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

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

5. Featurization

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

In [ ]:

```
! gdown --id 1lcxzVZ0-MkPmoH3lS35Q8rRfrecKSXb1
! gdown --id 1_KN7S8zfHdrkRjRYOEtBxBVq8JrGxPXD
```

```
Downloading...
From: https://drive.google.com/uc?id=1lcxzVZ0-MkPmoH3lS35Q8rRfrecKSXb1
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 [ ]:

In [37]:

```
import random
if os.path.isfile('/content/drive/MyDrive/Facebook/data/after_eda/train_after_eda.csv'):
    filename = "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 [39]:

```
if os.path.isfile('/content/drive/MyDrive/Facebook/data/after_eda/test_after_eda.csv'):
    filename = "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 [40]:

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

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

In [41]:

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

```
Downloading...
From: https://drive.google.com/uc?id=19mviN_yeJIfakb4kU5NfKdQlOQtaQ-kH
To: /content/train_y.csv
100% 45.3M/45.3M [00:00<00:00, 87.9MB/s]
```

In [ ]:

```
#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 [42]:

```
df_final_train = pd.read_csv('/content/drive/MyDrive/Facebook/data/after_eda/train_after_eda.csv', skiprows=skip_
df_final_train['indicator_link'] = pd.read_csv('/content/drive/MyDrive/Facebook/data/train_y.csv', skiprows=skip_
print("Our train matrix size ",df_final_train.shape)
df_final_train.head(2)
```

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

Out[42]:

	source_node	destination_node	indicator_link
0	273084	1505602	1
1	1538840	1141034	1

In [43]:

```
df_final_test = pd.read_csv('/content/drive/MyDrive/Facebook/data/after_eda/test_after_eda.csv', skiprows=skip_tr
df_final_test['indicator_link'] = pd.read_csv('/content/drive/MyDrive/Facebook/data/test_y.csv', skiprows=skip_tr
print("Our train matrix size ",df_final_test.shape)
df_final_test.head(2)
```

```
Our train matrix size  (24802, 3)
```

Out[43]:

	source_node	destination_node	indicator_link
0	848424	784690	1
1	1556382	708946	1

In [44]:

```
df_final_train.head()
```

Out[44]:

	source_node	destination_node	indicator_link
0	273084	1505602	1
1	1538840	1141034	1
2	529989	99844	1
3	1658696	1426508	1
4	1281563	640964	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 [45]:

```
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))
        # print(size(num_followers_s), size(num_followees_s, size(num_followers_d, size(num_followees_d)))


        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,inter_followees
```

In [46]:

```
num_followers_s,num_followees_s,num_followers_d,num_followees_d,inter_followers,inter_followees = compute_featur
```

In [47]:

```
num_followers_s_,num_followees_s_,num_followers_d_,num_followees_d_,inter_followers_,inter_followees_ = compute_
```

In [48]:

```
if not os.path.isfile('/content/drive/MyDrive/Facebook/data/storage_sample_stage1.h5'):
    df_final_train['num_followers_s'], df_final_train['num_followees_s'], df_final_train['num_followers_d'], df_f

    df_final_test['num_followers_s'], df_final_test['num_followees_s'], df_final_test['num_followers_d'], df_fina

    hdf = HDFStore('/content/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('/content/drive/MyDrive/Facebook/data/fea_sample/storage_sample_stage1.h5', 'trai
    # df_final_test = read_hdf('/content/drive/MyDrive/Facebook/data/fea_sample/storage_sample_stage1.h5', 'test_
```

In [49]:

```
# df_final_train = df_final_train.drop('num_follower_d', axis = 1)
df_final_train.head()
```

Out[49]:

	source_node	destination_node	indicator_link	num_followers_s	num_followees_s	num_followers_d	num_followees_d	inter_followers	inter_f
0	273084	1505602	1	11	15	6	8	0	
1	1538840	1141034	1	34	41	5	4	4	
2	529989	99844	1	28	50	16	1	9	
3	1658696	1426508	1	6	8	2	3	1	
4	1281563	640964	1	35	23	15	26	1	

In [50]:

```
df_final_test['num_followers_d'] = num_followers_d_
df_final_test.head()
```

Out[50]:

	source_node	destination_node	indicator_link	num_followers_s	num_followees_s	num_followers_d	num_followees_d	inter_followers	inter_f
0	848424	784690	1	6	6	14	9	1	
1	1556382	708946	1	0	0	4	0	0	
2	1204860	134642	1	9	13	7	7	1	
3	1294891	1447581	1	6	5	11	13	0	
4	1246795	639914	1	1	2	3	3	0	

In [51]:

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

In [ ]:

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

Out[ ]:

```
89571
```

In [ ]:

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

Out[ ]:

```
91634
```

In [ ]:

```

```

In [ ]:

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

```
Downloading...
From: https://drive.google.com/uc?id=1fDJptlCFEWNV5UNGPc4geTykgFI3PDCV
To: /content/storage_sample_stage4.h5
103MB [00:00, 155MB/s]
```

In [52]:

```
df_final_train_new = df_final_train.drop(['num_followers_s',    'num_followees_s',      'num_followers_d', 'num_f
```

In [53]:

```
df_final_test.shape
```

Out[53]:

```
(24802, 9)
```

In [ ]:

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

In [ ]:

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

```
Downloading...
From: https://drive.google.com/uc?id=10qJ04GRcaDxc16gmJXb8rpGPmlyys7E2
To: /content/storage_sample_stage2.h5
22.9MB [00:00, 105MB/s]
```

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 [54]:

```
df_final_train.head()
```

Out[54]:

	source_node	destination_node	indicator_link	num_followers_s	num_followees_s	num_followers_d	num_followees_d	inter_followers	inter_f
0	273084	1505602	1	11	15	6	8	0	
1	1538840	1141034	1	34	41	5	4	4	
2	529989	99844	1	28	50	16	1	9	
3	1658696	1426508	1	6	8	2	3	1	
4	1281563	640964	1	35	23	15	26	1	

In [55]:

```
df_final_test.head()
```

Out[55]:

	source_node	destination_node	indicator_link	num_followers_s	num_followees_s	num_followers_d	num_followees_d	inter_followers	inter_f
0	848424	784690	1	6	6	14	9	1	
1	1556382	708946	1	0	0	4	0	0	
2	1204860	134642	1	9	13	7	7	1	
3	1294891	1447581	1	6	5	11	13	0	
4	1246795	639914	1	1	2	3	3	0	

In [56]:

```
if os.path.isfile('/content/drive/MyDrive/Facebook/data/fea_sample/storage_sample_stage2.h5'):
    #mapping adar index on train
    df_final_train['adar_index'] = df_final_train.apply(lambda row: calc_adar_in(row['source_node'],row['destinat
    #mapping adar index on test
    df_final_test['adar_index'] = df_final_test.apply(lambda row: calc_adar_in(row['source_node'],row['destinatio
    #-------------------------------------------------------------------------------------------------------
    #mapping followback or not on train
    df_final_train['follows_back'] = df_final_train.apply(lambda row: follows_back(row['source_node'],row['destin

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

    #-------------------------------------------------------------------------------------------------------
    #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['de

    ##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['dest

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

    hdf = HDFStore('/content/storage_sample_stage2.h5')
    hdf.put('train_df',df_final_train, format='table', data_columns=True)
    hdf.put('test_df',df_final_test, format='table', data_columns=True)
    hdf.close()
# else:
    # df_final_train = read_hdf('/content/drive/MyDrive/Facebook/data/fea_sample/storage_sample_stage2.h5', 'trai
    # df_final_test = read_hdf('/content/drive/MyDrive/Facebook/data/fea_sample/storage_sample_stage2.h5', 'test_
```

In [57]:

```
df_final_train.head()
```

Out[57]:

	source_node	destination_node	indicator_link	num_followers_s	num_followees_s	num_followers_d	num_followees_d	inter_followers	inter_f
0	273084	1505602	1	11	15	6	8	0	
1	1538840	1141034	1	34	41	5	4	4	
2	529989	99844	1	28	50	16	1	9	
3	1658696	1426508	1	6	8	2	3	1	
4	1281563	640964	1	35	23	15	26	1	

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 = 1√1 + |X|

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

In [58]:

```
#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 [00:19<00:00, 90341.99it/s]
```

In [59]:

```
if os.path.isfile('/content/drive/MyDrive/Facebook/data/fea_sample/storage_sample_stage3.h5'):
    #mapping to pandas train
    df_final_train['weight_in'] = df_final_train.destination_node.apply(lambda x: Weight_in.get(x,mean_weight_in)
    df_final_train['weight_out'] = df_final_train.source_node.apply(lambda x: Weight_out.get(x,mean_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_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 [60]:

```
if os.path.isfile('/content/drive/MyDrive/Facebook/data/fea_sample/storage_sample_stage3.h5'):

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

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

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

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

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

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

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

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

    hdf = HDFStore('/content/storage_sample_stage3.h5')
    hdf.put('train_df',df_final_train, format='table', data_columns=True)
    hdf.put('test_df',df_final_test, format='table', data_columns=True)
    hdf.close()
else:
    df_final_train = read_hdf('/content/drive/MyDrive/Facebook/data/fea_sample/storage_sample_stage3.h5', 'train_
    df_final_test = read_hdf('/content/drive/MyDrive/Facebook/data/fea_sample/storage_sample_stage3.h5', 'test_di
```

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 [61]:

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

In [62]:

```
#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 [63]:

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

In [64]:

```
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 os.path.isfile('/content/drive/MyDrive/Facebook/data/fea_sample/storage_sample_stage4.h5'):
    #===================================================================================================

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

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

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

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

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

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

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

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

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

    hdf = HDFStore('/content/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 [66]:

```
# df_final_test['num_followers_d'] = num_followers_d
df_final_train.head()
```

Out[66]:

	source_node	destination_node	indicator_link	num_followers_s	num_followees_s	num_followers_d	num_followees_d	inter_followers	inter_f
0	273084	1505602	1	11	15	6	8	0	
1	1538840	1141034	1	34	41	5	4	4	
2	529989	99844	1	28	50	16	1	9	
3	1658696	1426508	1	6	8	2	3	1	
4	1281563	640964	1	35	23	15	26	1	

Preferential Attachment

Preferential Attachment for followers

In [67]:

```
#train set prefencial attachment followers
s = np.array(df_final_train['num_followers_s'])
d = np.array(df_final_train['num_followers_d'])
pref_attach = []
for i in range(len(df_final_train)):
  pref_attach.append(s[i]*d[i])
df_final_train['Preferential_followers'] = pref_attach
df_final_train.head()
```

Out[67]:

	source_node	destination_node	indicator_link	num_followers_s	num_followees_s	num_followers_d	num_followees_d	inter_followers	inter_f
0	273084	1505602	1	11	15	6	8	0	
1	1538840	1141034	1	34	41	5	4	4	
2	529989	99844	1	28	50	16	1	9	
3	1658696	1426508	1	6	8	2	3	1	
4	1281563	640964	1	35	23	15	26	1	

In [68]:

```
#test set prefrential attachment followers
s = np.array(df_final_test['num_followers_s'])
d = np.array(df_final_test['num_followers_d'])
pref_attach = []
for i in range(len(df_final_test)):
  pref_attach.append(s[i]*d[i])
df_final_test['Preferential_followers'] = pref_attach
df_final_test.head()
```

Out[68]:

	source_node	destination_node	indicator_link	num_followers_s	num_followees_s	num_followers_d	num_followees_d	inter_followers	inter_f
0	848424	784690	1	6	6	14	9	1	
1	1556382	708946	1	0	0	4	0	0	
2	1204860	134642	1	9	13	7	7	1	
3	1294891	1447581	1	6	5	11	13	0	
4	1246795	639914	1	1	2	3	3	0	

Preferential Attachment for followees

In [69]:

```
#train set prefencial attachment followees
s = np.array(df_final_train['num_followees_s'])
d = np.array(df_final_train['num_followees_d'])
pref_attach = []
for i in range(len(df_final_train)):
  pref_attach.append(s[i]*d[i])
df_final_train['Preferential_followees'] = pref_attach
df_final_train.head()
```

Out[69]:

	source_node	destination_node	indicator_link	num_followers_s	num_followees_s	num_followers_d	num_followees_d	inter_followers	inter_f
0	273084	1505602	1	11	15	6	8	0	
1	1538840	1141034	1	34	41	5	4	4	
2	529989	99844	1	28	50	16	1	9	
3	1658696	1426508	1	6	8	2	3	1	
4	1281563	640964	1	35	23	15	26	1	

In [70]:

```
#test set prefencial attachment followees
s = np.array(df_final_test['num_followees_s'])
d = np.array(df_final_test['num_followees_d'])
pref_attach = []
for i in range(len(df_final_test)):
  pref_attach.append(s[i]*d[i])
df_final_test['Preferential_followees'] = pref_attach
df_final_test.head()
```

Out[70]:

	source_node	destination_node	indicator_link	num_followers_s	num_followees_s	num_followers_d	num_followees_d	inter_followers	inter_f
0	848424	784690	1	6	6	14	9	1	
1	1556382	708946	1	0	0	4	0	0	
2	1204860	134642	1	9	13	7	7	1	
3	1294891	1447581	1	6	5	11	13	0	
4	1246795	639914	1	1	2	3	3	0	

In [71]:

```
df_final_train.head()
```

Out[71]:

	source_node	destination_node	indicator_link	num_followers_s	num_followees_s	num_followers_d	num_followees_d	inter_followers	inter_f
0	273084	1505602	1	11	15	6	8	0	
1	1538840	1141034	1	34	41	5	4	4	
2	529989	99844	1	28	50	16	1	9	
3	1658696	1426508	1	6	8	2	3	1	
4	1281563	640964	1	35	23	15	26	1	

In [ ]:

```
# df_final_train = df_final_train.drop('Preferential_Attachment', axis = 1)
# df_final_test = df_final_test.drop('Preferential_Attachment', axis = 1)
```

Adding feature SVD dot

In [ ]:

```
# source = np.vstack(df_final_train['svd_u_s_1'],df_final_train['svd_u_s_2'],df_final_train['svd_u_s_3'],df_final
                  #  df_final_train['svd_v_s_1'],df_final_train['svd_v_s_2'],df_final_train['svd_v_s_3'],df_final
# )
# destination = np.vstack(df_final_train['svd_u_d_1'],df_final_train['svd_u_d_2'],df_final_train['svd_u_d_3'],df_
                        # df_final_train['svd_v_d_1'],df_final_train['svd_v_d_2'],df_final_train['svd_v_d_3'],df_
# )
```

In [172…

```
# svd_dot = []
# dest = destination.reshape(-1, 12)
# print(dest.shape)
# svd_dot.append(np.dot(source, dest))
```

```
(100002, 12)
```

In [171…

```
# for i in range(len(svd_dot)):
  # print(svd_dot[i][1])
```

```
[ 6.83091889e-06  8.43547080e-05  8.76503885e-05  8.81604695e-05
  5.77177553e-05  5.04438253e-07  3.03458796e-04  1.15509459e-05
  4.77532709e-03  4.82379184e-07  1.16727713e-05 -1.37528676e-03]
```

In [156…

```
# df_final_train['svd_dot'] = svd_dot
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-156-d076700a791c> in <module>()
----> 1 df_final_train['svd_dot'] = svd_dot

/usr/local/lib/python3.7/dist-packages/pandas/core/frame.py in __setitem__(self, key, value)
```

```
   3042         else:
   3043             # set column
-> 3044             self._set_item(key, value)
   3045 
   3046     def _setitem_slice(self, key: slice, value):

/usr/local/lib/python3.7/dist-packages/pandas/core/frame.py in _set_item(self, key, value)
   3118         """
   3119         self._ensure_valid_index(value)
-> 3120         value = self._sanitize_column(key, value)
   3121         NDFrame._set_item(self, key, value)
   3122 

/usr/local/lib/python3.7/dist-packages/pandas/core/frame.py in _sanitize_column(self, key, value, broadcast)
   3766 
   3767             # turn me into an ndarray
-> 3768             value = sanitize_index(value, self.index)
   3769             if not isinstance(value, (np.ndarray, Index)):
   3770                 if isinstance(value, list) and len(value) > 0:

/usr/local/lib/python3.7/dist-packages/pandas/core/internals/construction.py in sanitize_index(data, index)
    746     if len(data) != len(index):
    747         raise ValueError(
--> 748             "Length of values "
    749             f"({len(data)}) "
    750             "does not match length of index "

ValueError: Length of values (1) does not match length of index (100002)
```

In [173…

```
#https://github.com/somjit101/Facebook-Friend-Recommendation/blob/main/FB_Graph_Edge_Prediction.ipynb
#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_tr
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

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_tr
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
```

In [174…

```
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 [175…

```
df_final_train.head()
```

Out[175…

	source_node	destination_node	indicator_link	num_followers_s	num_followees_s	num_followers_d	num_followees_d	inter_followers	inter_f
0	273084	1505602	1	11	15	6	8	0	
1	1538840	1141034	1	34	41	5	4	4	
2	529989	99844	1	28	50	16	1	9	
3	1658696	1426508	1	6	8	2	3	1	

4	1281563	640964	1	35	23	15	26	1

In [176…

```
#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[
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_te

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[
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_te
```

In [177…

```
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 [178…

```
hdf = HDFStore('/content/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 [ ]:

```
# prepared and stored the data from machine learning models
# pelase check the FB_Models.ipynb
```

Processing math: 100%