mayankgupta9968@gmail.com_25_1

March 3, 2020

Social network Graph Link Prediction - Facebook Challenge

[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 1. Reading Data

[2]:

```
if os.path.isfile('data/after_eda/train_pos_after_eda.csv'):
    train_graph=nx.read_edgelist('data/after_eda/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 2. Similarity measures

2.1 2.1 Jaccard Distance:

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

$$j = \frac{|X \cap Y|}{|X \cup Y|} \tag{1}$$

[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
```

[4]:

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

```
0.0
```

[5]:

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

```
0.0
```

[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
```

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

```
0
```

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

```
0
```

2.2 2.2 Cosine distance

$$CosineDistance = \frac{|X \cap Y|}{|X| \cdot |Y|} \tag{2}$$

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

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

```
0.0
```

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

```
0
```

[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
```

[13]:

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

```
0.02886751345948129
```

[14]:

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

```
0
```

2.3 3. Ranking Measures

https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.algorithms.link_an

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.

2.4 3.1 Page Ranking

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

[15]:

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

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

[17]:

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

```
5.615699699389075e-07
```

3 4. Other Graph Features

3.1 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.

[18]:

```
#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
```

[19]:

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

[19]:

```
10
```

[20]:

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

[20]:

```
-1
```

3.2 4.2 Checking for same community

[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
```

[22]:
```
belongs_to_same_wcc(861, 1659750)
```

[22]:
```
0
```

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

[23]:
```
0
```

3.3 4.3 Adamic/Adar Index:

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

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

[24]:

```
#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
```

[25]:

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

[25]:

```
0
```

[26]:

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

[26]:

```
0
```

3.4 4.4 Is persion was following back:

[27]:

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

[28]:

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

[28]:

```
1
```

[29]:

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

[29]:

```
0
```

3.5 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

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

[30]:

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

[31]:

```python
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
```

[32]:

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

```
0.0007483800935562018
```

3.6 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

[33]:

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

[34]:

```python
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
```

4 5. Featurization

4.1 5. 1 Reading a sample of Data from both train and test

[35]:
```
import random
if os.path.isfile('data/after_eda/train_after_eda.csv'):
    filename = "data/after_eda/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
```

[36]:
```
if os.path.isfile('data/after_eda/test_after_eda.csv'):
    filename = "data/after_eda/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
```

[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(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
```

[38]:
```
df_final_train = pd.read_csv('data/after_eda/train_after_eda.csv',␣
 ↪skiprows=skip_train, names=['source_node', 'destination_node'])
df_final_train['indicator_link'] = pd.read_csv('data/train_y.csv',␣
 ↪skiprows=skip_train, names=['indicator_link'])
print("Our train matrix size ",df_final_train.shape)
df_final_train.head(2)
```

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

[38]:
```
   source_node  destination_node  indicator_link
0       273084           1505602               1
1       527014           1605979               1
```

[39]:
```
df_final_test = pd.read_csv('data/after_eda/test_after_eda.csv',␣
 ↪skiprows=skip_test, names=['source_node', 'destination_node'])
df_final_test['indicator_link'] = pd.read_csv('data/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  (50002, 3)
```

[39]:
```
   source_node  destination_node  indicator_link
0       848424            784690               1
1       806059           1228951               1
```

4.2 5.2 Adding a set of features

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

jaccard_followers
jaccard_followees
cosine_followers
cosine_followees
num_followers_s
num_followees_s
num_followers_d
num_followees_d
inter_followers
inter_followees

[40]:
```
if not os.path.isfile('data/fea_sample/storage_sample_stage1.h5'):
    #mapping jaccrd followers to train and test data
    df_final_train['jaccard_followers'] = df_final_train.apply(lambda row:
                                                                        ␣

 ↪jaccard_for_followers(row['source_node'],row['destination_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 row:
                                                                        ␣

 ↪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 row:
                                            ␣
↪cosine_for_followers(row['source_node'],row['destination_node']),axis=1)
    df_final_test['cosine_followers'] = df_final_test.apply(lambda row:
                                            ␣
↪cosine_for_followers(row['source_node'],row['destination_node']),axis=1)

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

[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_followees_d,␣
 ↪inter_followers, inter_followees
```

[42]:
```
if not os.path.isfile('data/fea_sample/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('data/fea_sample/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('data/fea_sample/storage_sample_stage1.h5',␣
 ↪'train_df',mode='r')
    df_final_test = read_hdf('data/fea_sample/storage_sample_stage1.h5',␣
 ↪'test_df',mode='r')
```

4.3 5.3 Adding new set of features

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

adar index
is following back
belongs to same weakly connect components
shortest path between source and destination

[43]:
```
if not os.path.isfile('data/fea_sample/storage_sample_stage2.h5'):
    #mapping adar index on train
    df_final_train['adar_index'] = df_final_train.apply(lambda row:␣
 ↪calc_adar_in(row['source_node'],row['destination_node']),axis=1)
    #mapping adar index on test
    df_final_test['adar_index'] = df_final_test.apply(lambda row:␣
 ↪calc_adar_in(row['source_node'],row['destination_node']),axis=1)


   ␣
 ↪#--------------------------------------------------------------------------------------------
    #mapping followback or not on train
    df_final_train['follows_back'] = df_final_train.apply(lambda row:␣
 ↪follows_back(row['source_node'],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('data/fea_sample/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('data/fea_sample/storage_sample_stage2.h5',␣
 ↪'train_df',mode='r')
    df_final_test = read_hdf('data/fea_sample/storage_sample_stage2.h5',␣
 ↪'test_df',mode='r')
```

4.4 5.4 Adding new set of features

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

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
Page Ranking of source
Page Ranking of dest

katz of source
katz of dest
hubs of source
hubs of dest
authorities_s of source
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

$$W = \frac{1}{\sqrt{1 + |X|}} \tag{3}$$

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

[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

    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 [00:50<00:00, 35079.72it/s]
```

[45]:
```
if not os.path.isfile('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)
```

[46]:
```
if not os.path.isfile('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('data/fea_sample/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('data/fea_sample/storage_sample_stage3.h5',␣
 ↪'train_df',mode='r')
    df_final_test = read_hdf('data/fea_sample/storage_sample_stage3.h5',␣
 ↪'test_df',mode='r')
```

4.5 5.5 Adding new set of features

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

SVD features for both source and destination

[47]:
```
def svd(x, S):
    try:
        z = sadj_dict[x]
        return S[z]
    except:
        return [0,0,0,0,0,0]
```

[48]:
```
#for svd features to get feature vector creating a dict node val and index in␣
 ↪svd vector
sadj_col = sorted(train_graph.nodes())
sadj_dict = { val:idx for idx,val in enumerate(sadj_col)}
```

[49]:
```
Adj = nx.adjacency_matrix(train_graph,nodelist=sorted(train_graph.nodes())).
 ↪asfptype()
```

[50]:
```
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,)
```

[51]:
```
if not os.path.isfile('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('data/fea_sample/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('data/fea_sample/storage_sample_stage4.h5',␣
↪'train_df',mode='r')
    df_final_test = read_hdf('data/fea_sample/storage_sample_stage4.h5',␣
↪'test_df',mode='r')
```

[52]:
```
# print train data
df_final_train.head()
```

[52]:
```
   source_node  destination_node  indicator_link  jaccard_followers  \
0       273084           1505602               1                  0
1      1092078           1019460               1                  0
2      1430596            400599               1                  0
3      1013979           1628559               1                  0
4       197515            805550               1                  0
```

```
   jaccard_followees  cosine_followers  cosine_followees  num_followers_s  \
0           0.000000          0.000000          0.000000               11
1           0.000000          0.142857          0.000000                4
2           0.098039          0.051948          0.233126               49
3           0.333333          0.229081          0.524142               14
4           0.000000          0.161985          0.000000                7

   num_followers_d  num_followees_s  ...     svd_v_s_3     svd_v_s_4  \
0                6               15  ...  1.983704e-06  1.545078e-13
1                7                7  ...  2.118043e-11  1.521413e-13
2               11               46  ...  9.021667e-11  6.494928e-10
3                7               13  ...  8.142010e-15  1.718272e-16
4                7               12  ...  2.529855e-15  3.098996e-18

      svd_v_s_5     svd_v_s_6     svd_v_d_1     svd_v_d_2     svd_v_d_3  \
0  8.108401e-13  1.719703e-14 -1.355368e-12  4.675302e-13  1.128589e-06
1  1.477228e-12  1.347584e-14 -1.240513e-12  4.237680e-13  1.125696e-09
2  8.942299e-11  7.171148e-15 -4.091095e-13  4.076675e-14  2.783363e-12
3  2.777266e-15  1.597539e-18 -1.903970e-16  4.652690e-16  4.073271e-15
4  5.730760e-18  6.503282e-19 -4.760155e-19  2.910335e-16  1.515485e-15

      svd_v_d_4     svd_v_d_5     svd_v_d_6
0  6.616669e-14  9.771059e-13  4.160011e-14
1  1.917101e-12  1.483785e-12  2.649401e-12
2  4.809181e-13  9.748510e-14  1.847969e-16
3  8.574630e-17  1.387813e-15  7.993854e-19
4  1.854415e-18  3.395863e-18  3.250485e-19

[5 rows x 55 columns]
```

[53]:

```
# print test data
df_final_test.head()
```

[53]:

```
   source_node  destination_node  indicator_link  jaccard_followers  \
0       848424            784690               1                  0
1       182360            205736               1                  0
2       120585            539098               1                  0
3      1286685           1751018               1                  0
4      1284877            979430               1                  0

   jaccard_followees  cosine_followers  cosine_followees  num_followers_s  \
0           0.000000          0.029161          0.000000                6
1           0.187500          0.157485          0.358569                7
2           0.000000          0.051805          0.000000               73
3           0.021739          0.000000          0.050252               43
4           0.000000          0.000000          0.000000                1
```

```
   num_followers_d  num_followees_s  ...     svd_v_s_3     svd_v_s_4  \
0               14                6  ...  5.904829e-11  2.701538e-12
1               12                5  ...  7.053736e-15  4.758650e-16
2              122               28  ...  1.172482e-12  1.899497e-11
3               11               36  ...  2.061871e-09  1.434925e-11
4                6                0  ...  0.000000e+00  0.000000e+00

      svd_v_s_5     svd_v_s_6     svd_v_d_1     svd_v_d_2     svd_v_d_3  \
0  4.341594e-13  5.535489e-14 -9.994074e-10  5.791890e-10  3.512358e-07
1  1.077275e-13  1.262255e-18 -1.152091e-16  1.421397e-11  8.108408e-15
2  7.082778e-14  2.106447e-06 -1.208154e-12  2.721133e-14  1.976924e-12
3  2.782139e-10  2.050646e-14 -2.334162e-12  5.258642e-10  1.389547e-10
4  0.000000e+00  0.000000e+00 -1.949308e-13  1.340613e-14  1.431608e-13

      svd_v_d_4     svd_v_d_5     svd_v_d_6
0  2.486659e-09  2.771126e-09  1.727685e-12
1  5.273874e-16  1.418345e-13  1.925426e-18
2 -1.766527e-11  2.396330e-13  8.069914e-05
3  7.992433e-10  1.078892e-09  2.588075e-13
4  1.773713e-14  1.016923e-13  4.527517e-15

[5 rows x 55 columns]
```

[54]:

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

5 Added new features as per assignment

6 Add Preferential Attachment with followers and followees data of vertex.

[55]:

```
def preferential_attachment_followees(a, b):
    try:
        s1 = len(set(train_graph.successors(a)))
        d1 = len(set(train_graph.successors(b)))

        return s1 * d1
    except:
        return 0
```

[56]:

```
def preferential_attachment_followers(a, b):
    try:
        s2 = len(set(train_graph.predecessors(a)))
        d2 = len(set(train_graph.predecessors(b)))

        return s2 * d2
    except:
        return 0
```

[57]:

```
print(preferential_attachment_followees(273084, 1505602))
print(preferential_attachment_followers(273084, 1505602))
```

```
120
66
```

[58]:

```
if not os.path.isfile('data/fea_sample/storage_sample_stage5.h5'):
    ␣
 ↪#-------------------------------------------------------------------------------------------------
    #mapping preferential_attachment_followees on train
    df_final_train['preferential_attachment_followees'] = df_final_train.
 ↪apply(lambda row:␣
 ↪preferential_attachment_followees(row['source_node'],row['destination_node']),axis=1)
    #mapping preferential_attachment_followees on test
    df_final_test['preferential_attachment_followees'] = df_final_test.
 ↪apply(lambda row:␣
 ↪preferential_attachment_followees(row['source_node'],row['destination_node']),axis=1)

    ␣
 ↪#-------------------------------------------------------------------------------------------------
    #mapping preferential_attachment_followers on train
    df_final_train['preferential_attachment_followers'] = df_final_train.
 ↪apply(lambda row:␣
 ↪preferential_attachment_followers(row['source_node'],row['destination_node']),axis=1)

    #mapping preferential_attachment_followers on test
    df_final_test['preferential_attachment_followers'] = df_final_test.
 ↪apply(lambda row:␣
 ↪preferential_attachment_followers(row['source_node'],row['destination_node']),axis=1)

    ␣
 ↪#-------------------------------------------------------------------------------------------------

    hdf = HDFStore('data/fea_sample/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()
else:
    df_final_train = read_hdf('data/fea_sample/storage_sample_stage5.h5',␣
 ↪'train_df',mode='r')
    df_final_test = read_hdf('data/fea_sample/storage_sample_stage5.h5',␣
 ↪'test_df',mode='r')
```

[59]:

```
# print train data
df_final_train.head()
```

[59]:

```
   source_node  destination_node  indicator_link  jaccard_followers  \
0       273084           1505602               1                  0
1      1092078           1019460               1                  0
2      1430596            400599               1                  0
3      1013979           1628559               1                  0
4       197515            805550               1                  0

   jaccard_followees  cosine_followers  cosine_followees  num_followers_s  \
0           0.000000          0.000000          0.000000               11
1           0.000000          0.142857          0.000000                4
2           0.098039          0.051948          0.233126               49
3           0.333333          0.229081          0.524142               14
4           0.000000          0.161985          0.000000                7

   num_followers_d  num_followees_s  ...     svd_v_s_5     svd_v_s_6  \
0                6               15  ...  8.108401e-13  1.719703e-14
1                7                7  ...  1.477228e-12  1.347584e-14
2               11               46  ...  8.942299e-11  7.171148e-15
3                7               13  ...  2.777266e-15  1.597539e-18
4                7               12  ...  5.730760e-18  6.503282e-19

      svd_v_d_1     svd_v_d_2     svd_v_d_3     svd_v_d_4     svd_v_d_5  \
0 -1.355368e-12  4.675302e-13  1.128589e-06  6.616669e-14  9.771059e-13
1 -1.240513e-12  4.237680e-13  1.125696e-09  1.917101e-12  1.483785e-12
2 -4.091095e-13  4.076675e-14  2.783363e-12  4.809181e-13  9.748510e-14
3 -1.903970e-16  4.652690e-16  4.073271e-15  8.574630e-17  1.387813e-15
4 -4.760155e-19  2.910335e-16  1.515485e-15  1.854415e-18  3.395863e-18

      svd_v_d_6  preferential_attachment_followees  \
0  4.160011e-14                                120
1  2.649401e-12                                  0
2  1.847969e-16                                460
3  7.993854e-19                                 91
4  3.250485e-19                                  0

   preferential_attachment_followers
0                                 66
1                                 28
2                                539
3                                 98
4                                 49

[5 rows x 57 columns]
```

[60]:

```
# print test data
df_final_test.head()
```

[60]:

```
   source_node  destination_node  indicator_link  jaccard_followers  \
0       848424            784690               1                  0
1       182360            205736               1                  0
2       120585            539098               1                  0
3      1286685           1751018               1                  0
4      1284877            979430               1                  0

   jaccard_followees  cosine_followers  cosine_followees  num_followers_s  \
0           0.000000          0.029161          0.000000                6
1           0.187500          0.157485          0.358569                7
2           0.000000          0.051805          0.000000               73
3           0.021739          0.000000          0.050252               43
4           0.000000          0.000000          0.000000                1

   num_followers_d  num_followees_s  ...     svd_v_s_5     svd_v_s_6  \
0               14                6  ...  4.341594e-13  5.535489e-14
1               12                5  ...  1.077275e-13  1.262255e-18
2              122               28  ...  7.082778e-14  2.106447e-06
3               11               36  ...  2.782139e-10  2.050646e-14
4                6                0  ...  0.000000e+00  0.000000e+00

      svd_v_d_1     svd_v_d_2     svd_v_d_3     svd_v_d_4     svd_v_d_5  \
0 -9.994074e-10  5.791890e-10  3.512358e-07  2.486659e-09  2.771126e-09
1 -1.152091e-16  1.421397e-11  8.108408e-15  5.273874e-16  1.418345e-13
2 -1.208154e-12  2.721133e-14  1.976924e-12 -1.766527e-11  2.396330e-13
3 -2.334162e-12  5.258642e-10  1.389547e-10  7.992433e-10  1.078892e-09
4 -1.949308e-13  1.340613e-14  1.431608e-13  1.773713e-14  1.016923e-13

      svd_v_d_6  preferential_attachment_followees  \
0  1.727685e-12                                 54
1  1.925426e-18                                 70
2  8.069914e-05                                  0
3  2.588075e-13                                396
4  4.527517e-15                                  0

   preferential_attachment_followers
0                                 84
1                                 84
2                               8906
3                                473
4                                  6

[5 rows x 57 columns]
```

[61]:

```
# print(df_final_train[1:2]['num_followees_d'])
```

[62]:

```
df_final_train.columns
```

[62]:

```
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',
       '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',
       'preferential_attachment_followees',
       'preferential_attachment_followers'],
      dtype='object')
```

7 Add feature called svd_dot. you can calculate svd_dot as Dot product between sourse node svd and destination node svd features.

[63]:

```
# Collect svd data for train
source_u1 = df_final_train['svd_u_s_1']
source_u2 = df_final_train['svd_u_s_2']
source_u3 = df_final_train['svd_u_s_3']
source_u4 = df_final_train['svd_u_s_4']
source_u5 = df_final_train['svd_u_s_5']
source_u6 = df_final_train['svd_u_s_6']
destination_u1 = df_final_train['svd_u_d_1']
destination_u2 = df_final_train['svd_u_d_2']
destination_u3 = df_final_train['svd_u_d_3']
destination_u4 = df_final_train['svd_u_d_4']
destination_u5 = df_final_train['svd_u_d_5']
destination_u6 = df_final_train['svd_u_d_6']
source_v1 = df_final_train['svd_v_s_1']
source_v2 = df_final_train['svd_v_s_2']
source_v3 = df_final_train['svd_v_s_3']
source_v4 = df_final_train['svd_v_s_4']
source_v5 = df_final_train['svd_v_s_5']
source_v6 = df_final_train['svd_v_s_6']
destination_v1 = df_final_train['svd_v_d_1']
destination_v2 = df_final_train['svd_v_d_2']
destination_v3 = df_final_train['svd_v_d_3']
destination_v4 = df_final_train['svd_v_d_4']
destination_v5 = df_final_train['svd_v_d_5']
destination_v6 = df_final_train['svd_v_d_6']
```

[64]:
```
type(source_u1)
```

[64]:
```
pandas.core.series.Series
```

[65]:
```
svd_dot_product = []
for i in range(len(np.array(source_u1))):
    source = []
    destination = []

    source.append(np.array(source_u1[i]))
    source.append(np.array(source_u2[i]))
    source.append(np.array(source_u3[i]))
    source.append(np.array(source_u4[i]))
    source.append(np.array(source_u5[i]))
    source.append(np.array(source_u6[i]))
    source.append(np.array(source_v1[i]))
    source.append(np.array(source_v2[i]))
    source.append(np.array(source_v3[i]))
    source.append(np.array(source_v4[i]))
    source.append(np.array(source_v5[i]))
    source.append(np.array(source_v6[i]))

    destination.append(np.array(destination_u1[i]))
    destination.append(np.array(destination_u2[i]))
    destination.append(np.array(destination_u3[i]))
    destination.append(np.array(destination_u4[i]))
    destination.append(np.array(destination_u5[i]))
    destination.append(np.array(destination_u6[i]))
    destination.append(np.array(destination_v1[i]))
    destination.append(np.array(destination_v2[i]))
    destination.append(np.array(destination_v3[i]))
    destination.append(np.array(destination_v4[i]))
    destination.append(np.array(destination_v5[i]))
    destination.append(np.array(destination_v6[i]))

    svd_dot_product.append(np.dot(source,destination))
df_final_train['svd_dot_product'] = svd_dot_product
```

[66]:
```
# print train data
df_final_train.head()
```

[66]:
```
   source_node  destination_node  indicator_link  jaccard_followers  \
0       273084           1505602               1                  0
1      1092078           1019460               1                  0
2      1430596            400599               1                  0
3      1013979           1628559               1                  0
4       197515            805550               1                  0

   jaccard_followees  cosine_followers  cosine_followees  num_followers_s  \
0          0.000000         0.000000         0.000000                11
1          0.000000         0.142857         0.000000                 4
2          0.098039         0.051948         0.233126                49
3          0.333333         0.229081         0.524142                14
4          0.000000         0.161985         0.000000                 7

   num_followers_d  num_followees_s  ...      svd_v_s_6     svd_v_d_1  \
0                6               15  ...  1.719703e-14 -1.355368e-12
1                7                7  ...  1.347584e-14 -1.240513e-12
2               11               46  ...  7.171148e-15 -4.091095e-13
3                7               13  ...  1.597539e-18 -1.903970e-16
4                7               12  ...  6.503282e-19 -4.760155e-19

      svd_v_d_2     svd_v_d_3     svd_v_d_4     svd_v_d_5     svd_v_d_6  \
0  4.675302e-13  1.128589e-06  6.616669e-14  9.771059e-13  4.160011e-14
1  4.237680e-13  1.125696e-09  1.917101e-12  1.483785e-12  2.649401e-12
2  4.076675e-14  2.783363e-12  4.809181e-13  9.748510e-14  1.847969e-16
3  4.652690e-16  4.073271e-15  8.574630e-17  1.387813e-15  7.993854e-19
4  2.910335e-16  1.515485e-15  1.854415e-18  3.395863e-18  3.250485e-19

   preferential_attachment_followees  preferential_attachment_followers  \
0                                120                                 66
1                                  0                                 28
2                                460                                539
3                                 91                                 98
4                                  0                                 49

   svd_dot_product
0     1.338835e-11
1     2.384645e-20
2     1.252459e-21
3     2.609823e-28
4     3.974762e-30

[5 rows x 58 columns]
```

[67]:
```
# Collect svd data for test
source_u1 = df_final_test['svd_u_s_1']
source_u2 = df_final_test['svd_u_s_2']
source_u3 = df_final_test['svd_u_s_3']
source_u4 = df_final_test['svd_u_s_4']
source_u5 = df_final_test['svd_u_s_5']
source_u6 = df_final_test['svd_u_s_6']
destination_u1 = df_final_test['svd_u_d_1']
destination_u2 = df_final_test['svd_u_d_2']
destination_u3 = df_final_test['svd_u_d_3']
destination_u4 = df_final_test['svd_u_d_4']
```

```
destination_u5 = df_final_test['svd_u_d_5']
destination_u6 = df_final_test['svd_u_d_6']
source_v1 = df_final_test['svd_v_s_1']
source_v2 = df_final_test['svd_v_s_2']
source_v3 = df_final_test['svd_v_s_3']
source_v4 = df_final_test['svd_v_s_4']
source_v5 = df_final_test['svd_v_s_5']
source_v6 = df_final_test['svd_v_s_6']
destination_v1 = df_final_test['svd_v_d_1']
destination_v2 = df_final_test['svd_v_d_2']
destination_v3 = df_final_test['svd_v_d_3']
destination_v4 = df_final_test['svd_v_d_4']
destination_v5 = df_final_test['svd_v_d_5']
destination_v6 = df_final_test['svd_v_d_6']
```

[68]:

```
svd_dot_product = []
for i in range(len(np.array(source_u1))):
    source = []
    destination = []

    source.append(np.array(source_u1[i]))
    source.append(np.array(source_u2[i]))
    source.append(np.array(source_u3[i]))
    source.append(np.array(source_u4[i]))
    source.append(np.array(source_u5[i]))
    source.append(np.array(source_u6[i]))
    source.append(np.array(source_v1[i]))
    source.append(np.array(source_v2[i]))
    source.append(np.array(source_v3[i]))
    source.append(np.array(source_v4[i]))
    source.append(np.array(source_v5[i]))
    source.append(np.array(source_v6[i]))

    destination.append(np.array(destination_u1[i]))
    destination.append(np.array(destination_u2[i]))
    destination.append(np.array(destination_u3[i]))
    destination.append(np.array(destination_u4[i]))
    destination.append(np.array(destination_u5[i]))
    destination.append(np.array(destination_u6[i]))
    destination.append(np.array(destination_v1[i]))
    destination.append(np.array(destination_v2[i]))
    destination.append(np.array(destination_v3[i]))
    destination.append(np.array(destination_v4[i]))
    destination.append(np.array(destination_v5[i]))
    destination.append(np.array(destination_v6[i]))

    svd_dot_product.append(np.dot(source,destination))
```

```
df_final_test['svd_dot_product'] = svd_dot_product
```

[69]:
```
# print test data
df_final_test.head()
```

[69]:
```
   source_node  destination_node  indicator_link  jaccard_followers  \
0       848424            784690               1                  0
1       182360            205736               1                  0
2       120585            539098               1                  0
3      1286685           1751018               1                  0
4      1284877            979430               1                  0

   jaccard_followees  cosine_followers  cosine_followees  num_followers_s  \
0           0.000000          0.029161          0.000000                6
1           0.187500          0.157485          0.358569                7
2           0.000000          0.051805          0.000000               73
3           0.021739          0.000000          0.050252               43
4           0.000000          0.000000          0.000000                1

   num_followers_d  num_followees_s  ...     svd_v_s_6     svd_v_d_1  \
0               14                6  ...  5.535489e-14 -9.994074e-10
1               12                5  ...  1.262255e-18 -1.152091e-16
2              122               28  ...  2.106447e-06 -1.208154e-12
3               11               36  ...  2.050646e-14 -2.334162e-12
4                6                0  ...  0.000000e+00 -1.949308e-13

      svd_v_d_2     svd_v_d_3     svd_v_d_4     svd_v_d_5     svd_v_d_6  \
0  5.791890e-10  3.512358e-07  2.486659e-09  2.771126e-09  1.727685e-12
1  1.421397e-11  8.108408e-15  5.273874e-16  1.418345e-13  1.925426e-18
2  2.721133e-14  1.976924e-12 -1.766527e-11  2.396330e-13  8.069914e-05
3  5.258642e-10  1.389547e-10  7.992433e-10  1.078892e-09  2.588075e-13
4  1.340613e-14  1.431608e-13  1.773713e-14  1.016923e-13  4.527517e-15

   preferential_attachment_followees  preferential_attachment_followers  \
0                                 54                                 84
1                                 70                                 84
2                                  0                               8906
3                                396                                473
4                                  0                                  6

   svd_dot_product
0     2.083237e-17
1     3.215717e-22
2     1.699884e-10
3     6.821693e-19
4    -1.575727e-30

[5 rows x 58 columns]
```

[70]:

```
if not os.path.isfile('data/fea_sample/storage_sample_stage6.h5'):
    hdf = HDFStore('data/fea_sample/storage_sample_stage6.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('data/fea_sample/storage_sample_stage6.h5',␣
 ↪'train_df',mode='r')
    df_final_test = read_hdf('data/fea_sample/storage_sample_stage6.h5',␣
 ↪'test_df',mode='r')
```

[71]:

```
# print train data
df_final_train.head()
```

[71]:

```
   source_node  destination_node  indicator_link  jaccard_followers  \
0       273084           1505602               1                  0
1      1092078           1019460               1                  0
2      1430596            400599               1                  0
3      1013979           1628559               1                  0
4       197515            805550               1                  0

   jaccard_followees  cosine_followers  cosine_followees  num_followers_s  \
0           0.000000          0.000000          0.000000               11
1           0.000000          0.142857          0.000000                4
2           0.098039          0.051948          0.233126               49
3           0.333333          0.229081          0.524142               14
4           0.000000          0.161985          0.000000                7

   num_followers_d  num_followees_s  ...      svd_v_s_6     svd_v_d_1  \
0                6               15  ...  1.719703e-14 -1.355368e-12
1                7                7  ...  1.347584e-14 -1.240513e-12
2               11               46  ...  7.171148e-15 -4.091095e-13
3                7               13  ...  1.597539e-18 -1.903970e-16
4                7               12  ...  6.503282e-19 -4.760155e-19

      svd_v_d_2     svd_v_d_3     svd_v_d_4     svd_v_d_5     svd_v_d_6  \
0  4.675302e-13  1.128589e-06  6.616669e-14  9.771059e-13  4.160011e-14
1  4.237680e-13  1.125696e-09  1.917101e-12  1.483785e-12  2.649401e-12
2  4.076675e-14  2.783363e-12  4.809181e-13  9.748510e-14  1.847969e-16
3  4.652690e-16  4.073271e-15  8.574630e-17  1.387813e-15  7.993854e-19
4  2.910335e-16  1.515485e-15  1.854415e-18  3.395863e-18  3.250485e-19

   preferential_attachment_followees  preferential_attachment_followers  \
0                                120                                 66
1                                  0                                 28
2                                460                                539
3                                 91                                 98
4                                  0                                 49
```

```
   svd_dot_product
0     1.338835e-11
1     2.384645e-20
2     1.252459e-21
3     2.609823e-28
4     3.974762e-30

[5 rows x 58 columns]
```

[72]:
```
# print test data
df_final_test.head()
```

[72]:
```
   source_node  destination_node  indicator_link  jaccard_followers  \
0       848424            784690               1                  0
1       182360            205736               1                  0
2       120585            539098               1                  0
3      1286685           1751018               1                  0
4      1284877            979430               1                  0

   jaccard_followees  cosine_followers  cosine_followees  num_followers_s  \
0           0.000000          0.029161          0.000000                6
1           0.187500          0.157485          0.358569                7
2           0.000000          0.051805          0.000000               73
3           0.021739          0.000000          0.050252               43
4           0.000000          0.000000          0.000000                1

   num_followers_d  num_followees_s  ...     svd_v_s_6     svd_v_d_1  \
0               14                6  ...  5.535489e-14 -9.994074e-10
1               12                5  ...  1.262255e-18 -1.152091e-16
2              122               28  ...  2.106447e-06 -1.208154e-12
3               11               36  ...  2.050646e-14 -2.334162e-12
4                6                0  ...  0.000000e+00 -1.949308e-13

      svd_v_d_2     svd_v_d_3     svd_v_d_4     svd_v_d_5     svd_v_d_6  \
0  5.791890e-10  3.512358e-07  2.486659e-09  2.771126e-09  1.727685e-12
1  1.421397e-11  8.108408e-15  5.273874e-16  1.418345e-13  1.925426e-18
2  2.721133e-14  1.976924e-12 -1.766527e-11  2.396330e-13  8.069914e-05
3  5.258642e-10  1.389547e-10  7.992433e-10  1.078892e-09  2.588075e-13
4  1.340613e-14  1.431608e-13  1.773713e-14  1.016923e-13  4.527517e-15

   preferential_attachment_followees  preferential_attachment_followers  \
0                                 54                                 84
1                                 70                                 84
2                                  0                               8906
3                                396                                473
4                                  0                                  6
```

```
   svd_dot_product
0     2.083237e-17
1     3.215717e-22
2     1.699884e-10
3     6.821693e-19
4    -1.575727e-30

[5 rows x 58 columns]
```

[ ]:

```

```