In [1]:
```
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.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 networkx as nx
import pdb
from pandas import HDFStore,DataFrame
from pandas import read_hdf
from scipy.sparse.linalg import svds, eigs
import gc
from tqdm import tqdm

after_eda = 'data/after_eda'
fea_sample = 'data/fea_sample'
```

In [2]:
```
if os.path.isfile(f'{after_eda}/train_pos_after_eda.csv'):
    train_graph = nx.read_edgelist(f'{after_eda}/train_pos_after_eda.csv', delimiter=',', create_usi
ng=nx.DiGraph(), nodetype=int)
    print(nx.info(train_graph))
else:
    print('Please run EDA.ipynb file')
```

```
Name:
Type: DiGraph
Number of nodes: 1780722
Number of edges: 7550015
Average in degree:   4.2399
```

```
Average out degree:   4.2399
```

2. Similarity measures

2.1 Jaccard Distance:

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

In [3]:
```
def jaccard_for_followees(a,b):
    try:
        if len(set(train_graph.successors(a))) == 0 | len(set(train_graph.successors(b))) == 0:
            return 0
        sim = (len(set(train_graph.successors(a)).intersection(set(train_graph.successors(b))))) /\
                    (len(set(train_graph.successors(a)).union(set(train_graph.successors(b)))))
    except:
        return 0
    return sim
```

In [4]:
```
print(jaccard_for_followees(273084,1505602))
```

```
0.0
```

In [5]:
```
def jaccard_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)))))
/\
                    (len(set(train_graph.predecessors(a)).union(set(train_graph.predecessors(b)))))

        return sim
    except:
        return 0
```

In [6]:
```
print(jaccard_for_followers(1,2123))
```

```
0.0
```

2.2 Cosine distance

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

In [7]:
```
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_gr
aph.successors(b))))))
        return sim
    except:
        return 0
```

In [8]:
```
print(cosine_for_followees(273084,1505602))
```

```
0.0
```

In [9]:
```
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(trai
n_graph.predecessors(b)))))
        return sim
    except:
        return 0
```

In [10]:
```
print(cosine_for_followers(2,470294))
```

```
0.02886751345948129
```

3. Ranking Measures

3.1 Page Ranking

In [11]:
```
if not os.path.isfile(f'{fea_sample}/page_rank.p'):
    pr = nx.pagerank(train_graph, alpha=0.85)
    pickle.dump(pr, open(f'{fea_sample}/page_rank.p', 'wb'))
else:
    pr = pickle.load(open(f'{fea_sample}/page_rank.p', 'rb'))
```

In [12]:
```
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 [13]:
```
mean_pr = float(sum(pr.values())) / len(pr)
print(mean_pr)
```

```
5.615699699389075e-07
```

4. Other Graph Features

4.1 Shortest path:

In [14]:
```
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 [15]:
```
compute_shortest_path_length(77697, 826021)
```

Out[15]:
```
10
```

In [16]:
```
compute_shortest_path_length(669354,1635354)
```

Out[16]: -1

4.2 Checking for same community

In [17]:
```
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 [18]:
```
belongs_to_same_wcc(861, 1659750)
```

Out[18]: 0

4.3 Adamic/Adar Index:

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

In [19]:
```
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 [20]:
```
calc_adar_in(1,189226)
```

Out[20]: 0

4.4 Is persion was following back:

In [21]:
```
def follows_back(a,b):
    if train_graph.has_edge(b,a):
        return 1
    else:
        return 0
```

In [22]:
```
follows_back(1,189226)
```

Out[22]: 1

4.5 Katz Centrality:

Katz centrality computes the centrality for a node based on the centrality of its neighbors. It is a generalization of the eigenvector centrality. The Katz centrality for node `i` is

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

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

$$\lambda$$

.

The parameter

$$\beta$$

controls the initial centrality and

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

In [23]:
```
if not os.path.isfile(f'{fea_sample}/katz.p'):
    katz = nx.katz.katz_centrality(train_graph, alpha=0.005, beta=1)
    pickle.dump(katz, open(f'{fea_sample}/katz.p', 'wb'))
else:
    katz = pickle.load(open(f'{fea_sample}/katz.p', 'rb'))
```

In [24]:
```
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 [25]:
```
mean_katz = float(sum(katz.values())) / len(katz)
print(mean_katz)
```

```
0.0007483800935562018
```

4.6 Hits Score

In [26]:
```
if not os.path.isfile(f'{fea_sample}/hits.p'):
    hits = nx.hits(train_graph, max_iter=100, tol=1e-08, nstart=None, normalized=True)
    pickle.dump(hits, open(f'{fea_sample}/hits.p', 'wb'))
else:
    hits = pickle.load(open(f'{fea_sample}/hits.p', 'rb'))
```

In [27]:
```
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
```

In [28]:
```
mean_hits = float(sum(hits[0].values())) / len(hits[0])
print(mean_hits)
```

```
5.615699699344123e-07
```

5. Featurization

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

In [29]:
```
import random
if os.path.isfile(f'{after_eda}/train_after_eda.csv'):
    filename = f'{after_eda}/train_after_eda.csv'

    n_train =  15100028
    s = 100000
    skip_train = sorted(random.sample(range(1, n_train + 1), n_train-s))
```

In [30]:
```
if os.path.isfile(f'{after_eda}/train_after_eda.csv'):
    filename = f'{after_eda}/test_after_eda.csv'

    n_test = 3775006
    s = 50000 #desired sample size
    skip_test = sorted(random.sample(range(1,n_test+1),n_test-s))
```

In [31]:
```
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 [32]:
```
df_final_train = pd.read_csv(f'{after_eda}/train_after_eda.csv', skiprows=skip_train, names=['source
```

```
_node', 'destination_node'])
df_final_train['indicator_link'] = pd.read_csv('sandydata/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)
```

Out[32]:

	source_node	destination_node	indicator_link
0	273084	1505602	1
1	1610529	235333	1

In [33]:
```
df_final_test = pd.read_csv(f'{after_eda}/test_after_eda.csv', skiprows=skip_test, names=['source_no
de', 'destination_node'])
df_final_test['indicator_link'] = pd.read_csv('sandydata/test_y.csv', skiprows=skip_test, names=['in
dicator_link'])
print("Our test matrix size ",df_final_test.shape)
df_final_test.head(2)
```

```
Our test matrix size  (50002, 3)
```

Out[33]:

	source_node	destination_node	indicator_link
0	848424	784690	1
1	548473	1721521	1

5.2 Adding a set of features

In [34]:
```
if not os.path.isfile(f'{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_nod
e'],row['destination_node']), axis=1)
    df_final_test['jaccard_followers'] = df_final_test.apply(lambda row:
                                                             jaccard_for_followers(row['source_nod
e'],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_nod
```

```
e'],row['destination_node']), axis=1)
    df_final_test['jaccard_followees'] = df_final_test.apply(lambda row:
                                                             jaccard_for_followees(row['source_nod
e'],row['destination_node']), axis=1)

    #mapping cosine 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 cosine 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)
```

In [35]:

```
def compute_features_stage1(df_final):

    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, inte
r_followees
```

In [36]:
```
if not os.path.isfile(f'{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_f
inal_test)

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

5.3 Adding new set of features

In [37]:
```
if not os.path.isfile(f'{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'],ro
w['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_n
ode'],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_nod
e'],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(ro
w['source_node'],row['destination_node']),axis=1)

    hdf = HDFStore(f'{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(f'{fea_sample}/storage_sample_stage2.h5', 'train_df',mode='r')
    df_final_test = read_hdf(f'{fea_sample}/storage_sample_stage2.h5', 'test_df',mode='r')
```

In [38]:

```
Weight_in = {}
Weight_out = {}
for i in tqdm(train_graph.nodes()):
    s1 = set(train_graph.predecessors(i))
    w_in = 1/(np.sqrt(1+ len(s1)))
    Weight_in[i]=w_in

    s2 = set(train_graph.successors(i))
    w_out = 1/(np.sqrt(1+len(s2)))
    Weight_out[i]=w_out

mean_weight_in = np.mean(list(Weight_in.values()))
mean_weight_out = np.mean(list(Weight_out.values()))
```

```
100%|██████████| 1780722/1780722 [00:14<00:00, 120417.34it/s]
```

In [39]:
```
if not os.path.isfile(f'{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 [40]:
```
if not os.path.isfile(f'{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(f'{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(f'{fea_sample}/storage_sample_stage3.h5', 'train_df',mode='r')
    df_final_test = read_hdf(f'{fea_sample}/storage_sample_stage3.h5', 'test_df',mode='r')
```

In [41]:
```
def svd(x, S):
    try:
        z = sadj_dict[x]
        return S[z]
    except:
        return [0,0,0,0,0,0]
```

In [42]:
```
sadj_col = sorted(train_graph.nodes())
sadj_dict = {val:idx for idx, val in enumerate(sadj_col)}
```

In [43]:
```
Adj = nx.adjacency_matrix(train_graph, nodelist=sorted(train_graph.nodes())).asfptype()
```

In [44]:
```
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 [55]:
```
if not os.path.isfile(f'{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(f'{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(f'{fea_sample}/storage_sample_stage4.h5', 'train_df',mode='r')
    df_final_test = read_hdf(f'{fea_sample}/storage_sample_stage4.h5', 'test_df',mode='r')
```

Preferential Attachment

In [65]:
```
def preferential_for_followers(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)))*len((set(train_graph.successors(b))))
    except:
        return 0
    return sim
```

In [66]:
```
def preferential_for_followees(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)))*len((set(train_graph.predecessors(b))))
    except:
        return 0
    return sim
```

SVD DOT feature

In [49]:
```
def combine_svds(u_1, u_2, u_3, u_4, u_5, u_6, v_1, v_2, v_3, v_4, v_5, v_6):
    return np.array([u_1, u_2, u_3, u_4, u_5, u_6, v_1, v_2, v_3, v_4, v_5, v_6])
```

In [70]:
```
def svd_dot(svd_a, svd_b):
    return np.dot(svd_a, svd_b)
```

In [71]:
```
if not os.path.isfile(f'{fea_sample}/storage_sample_stage5.h5'):

    #preferential attachment for followers on train data
    df_final_train['preferential_followers'] = \
    df_final_train.apply(lambda row: preferential_for_followers(row['source_node'],row['destination_
node']),axis=1)

    #preferential attachment for followees on train data
    df_final_train['preferential_followees'] = \
    df_final_train.apply(lambda row: preferential_for_followees(row['source_node'],row['destination_
node']),axis=1)
    #=============================================================================================
============


    #preferential attachment for followrs on test data
    df_final_test['preferential_followers'] = \
    df_final_test.apply(lambda row: preferential_for_followers(row['source_node'],row['destination_n
ode']),axis=1)

    #preferential attachment for followees on test data
    df_final_test['preferential_followees'] = \
    df_final_test.apply(lambda row: preferential_for_followees(row['source_node'],row['destination_n
ode']),axis=1)

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


    #svd dot feature on train data
    svd_dot_train = []
    for i in range(len(df_final_train.svd_u_s_1)):

        svd_s = combine_svds(df_final_train.svd_u_s_1[i], df_final_train.svd_u_s_2[i], df_final_trai
n.svd_u_s_3[i],
                             df_final_train.svd_u_s_4[i], df_final_train.svd_u_s_5[i], df_final_train
.svd_u_s_6[i],
                             df_final_train.svd_v_s_1[i], df_final_train.svd_v_s_2[i], df_final_train
.svd_v_s_3[i],
                             df_final_train.svd_v_s_4[i], df_final_train.svd_v_s_5[i], df_final_train
.svd_v_s_6[i],)



        svd_d = combine_svds(df_final_train.svd_u_d_1[i], df_final_train.svd_u_d_2[i], df_final_trai
n.svd_u_d_3[i],
```

```
                                df_final_train.svd_u_d_4[i], df_final_train.svd_u_d_5[i], df_final_train
.svd_u_d_6[i],
                                df_final_train.svd_v_d_1[i], df_final_train.svd_v_d_2[i], df_final_train
.svd_v_d_3[i],
                                df_final_train.svd_v_d_4[i], df_final_train.svd_v_d_5[i], df_final_train
.svd_v_d_6[i],)


            svd_dot_train.append(svd_dot(svd_s, svd_d))
        df_final_train['svd_dot'] = svd_dot_train
        #=================================================================================================
============


        #svd dot feature on test data
        svd_dot_test = []
        for i in range(len(df_final_test.svd_u_s_1)):

            svd_s = combine_svds(df_final_test.svd_u_s_1[i], df_final_test.svd_u_s_2[i], df_final_test.s
vd_u_s_3[i],
                                df_final_test.svd_u_s_4[i], df_final_test.svd_u_s_5[i], df_final_test.sv
d_u_s_6[i],
                                df_final_test.svd_v_s_1[i], df_final_test.svd_v_s_2[i], df_final_test.sv
d_v_s_3[i],
                                df_final_test.svd_v_s_4[i], df_final_test.svd_v_s_5[i], df_final_test.sv
d_v_s_6[i],)



            svd_d = combine_svds(df_final_test.svd_u_d_1[i], df_final_test.svd_u_d_2[i], df_final_test.s
vd_u_d_3[i],
                                df_final_test.svd_u_d_4[i], df_final_test.svd_u_d_5[i], df_final_test.sv
d_u_d_6[i],
                                df_final_test.svd_v_d_1[i], df_final_test.svd_v_d_2[i], df_final_test.sv
d_v_d_3[i],
                                df_final_test.svd_v_d_4[i], df_final_test.svd_v_d_5[i], df_final_test.sv
d_v_d_6[i],)


            svd_dot_test.append(svd_dot(svd_s, svd_d))
        df_final_test['svd_dot'] = svd_dot_test
        #=================================================================================================
============

        hdf = HDFStore(f'{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(f'{fea_sample}/storage_sample_stage5.h5', 'train_df',mode='r')
    df_final_test = read_hdf(f'{fea_sample}/storage_sample_stage5.h5', 'test_df',mode='r')
```