DEPARTMENT OF COMPUTER ENGINEERING

Semester	T.E. Semester V – Computer Engineering
Subject	Data Warehousing and Mining
Subject Professor In-charge	Prof. Kavita P Shirsat
Laboratory	Lab 313-A

Student Name	Aditya Oza	
Roll Number	20102A0031	
Grade and Subject Teacher's Signature		

Experiment Title	PBL – Implementation of Page Rank Algorithm	
Resources / Apparatus Required	Hardware: Computer system	Software: Python
Objectives (Skill Set / Knowledge Tested / Imparted)	Implementation of Page Rank Algorithm on real life dataset	

Code

```
# Commented out IP

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# %matplotlib inline

import networkx as nx

G_tmp = nx.read_edgelist('../input/google-web-graph/web-Google.txt', create_using = nx.DiGraph)

print(nx.info(G_tmp))

# # The simplest method, but not available in Kaggle environment.

# G = max(nx.weakly_connected_components(G_tmp), key=len)

# The alternate method

c = sorted(nx.weakly_connected_components(G_tmp), key=len, reverse=True)

wcc_set = c[0]

G = G_tmp.subgraph(wcc_set)

print(nx.info(G))
```

	`` # Your code here, you can add cells if necessary def mypagerank(G, alpha=0.85, max_iter=100, tol=1.0e-6, weight='weight'): if len(G) == 0: return {} if not G.is_directed(): D = G.to_directed() else: D = G # Create a copy in (right) stochastic form W = nx.stochastic_graph(D, weight=weight) N = W.number_of_nodes() x = dict.fromkeys(W, 1.0 / N) #和为 1 # Assign uniform personalization vector if not given p = dict.fromkeys(W, 1.0 / N) # Use personalization vector if dangling vector not specified dangling_weights = p dangling_nodes = [n for n in W if W.out_degree(n, weight=weight) == 0.0] for _ in range(max_iter): xlast = x x = dict.fromkeys(xlast.keys(), 0) danglesum = alpha * sum(xlast[n] for n in dangling_nodes) for n in x: for nbr in W[n]: x[nbr] += alpha * xlast[n] * W[n][nbr][weight] for n in x: x[n] += danglesum * dangling_weights[n] + (1.0 - alpha) * p[n] err = sum([abs(x[n] - xlast[n]) for n in x]) if err < N*tol: return x ```

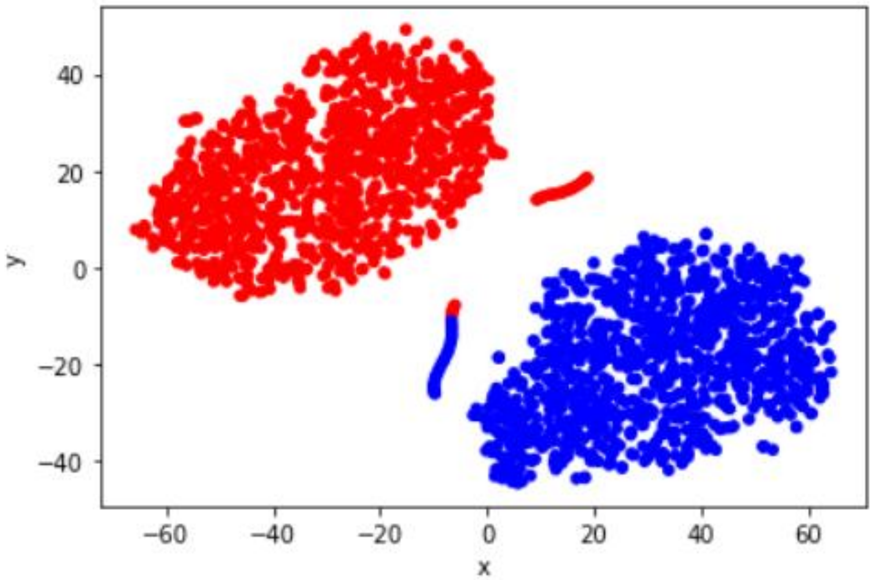
	pr = mypagerank(G, alpha=0.85) pr_v = nx.pagerank(G, alpha=0.85) from sklearn.metrics.pairwise import cosine_similarity print(cosine_similarity([list(pr.values())],[list(pr_v.values())])) # Due to the significant internal refactoring of gensim 4.0.0, we need to degrade it for consistency !pip install gensim==3.6.0 !pip install nodevectors # Commented out IPython magic to ensure Python compatibility. # Common libraries may be used import pandas as pd import numpy as np import matplotlib.pyplot as plt import sklearn # %matplotlib inline # Libaries for graph processing import nodevectors import networkx as nx class Node2Vec(nodevectors.Node2Vec): def __init__(self, p = 1, q = 1,d = 32, w = 10): super().__init__(n_components = d, walklen = w, epochs = 50, return_weight = 1.0 / p, neighbor_weight = 1.0 / q, threads = 0, w2vparams = {'window': 4, 'negative': 5,

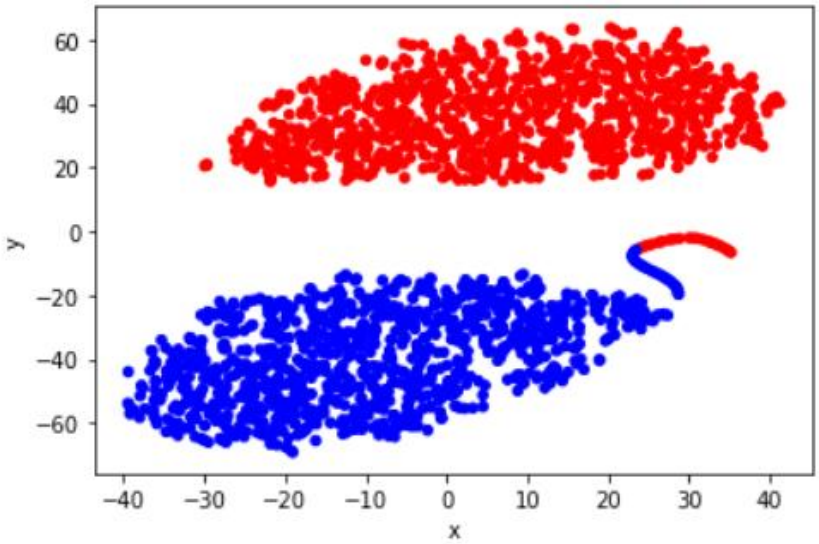
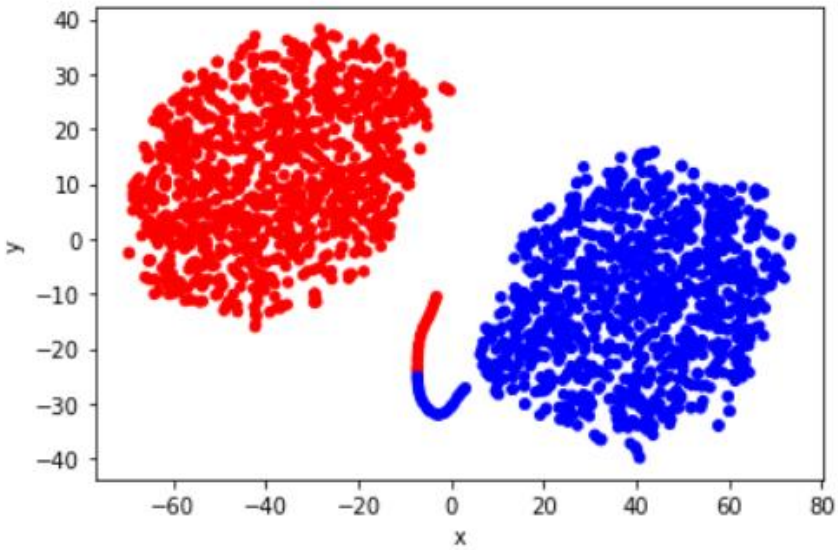
```
                'iter': 10,

                'ns_exponent': 0.5,

                'batch_words': 128})

# A toy example, not for your task

toy_barbell = nx.barbell_graph(7, 2)

nx.draw_kamada_kawai(toy_barbell)

# Use Node2Vec class to embed nodes

n2v = Node2Vec(p = 1, q = 1, d = 2)

n2v.fit(toy_barbell)

embeddings = []

for node in toy_barbell.nodes:

  embeddings.append(list(n2v.predict(node)))

# Construct a pandas dataframe with the 2D embeddings from node2vec.

# We can easily divide the nodes into two clusters, and the groudtruth is denoted by distinct colors.

toy_colors = ['red'] * 8 + ['blue'] * 8

df = pd.DataFrame(embeddings, columns = ['x', 'y']) # Create pandas dataframe from the list of node embeddings

df.plot.scatter(x = 'x', y = 'y', c = toy_colors)

# A barbell graph for your task 2.1

barbell_1 = nx.barbell_graph(1000, 0)


# Your code here, you can add cells if necessary

n2v_1 = Node2Vec(p=1,q=1,d=10,w=10)

n2v_1.fit(barbell_1)

embeddings_1 = []

for node in barbell_1.nodes():

  embeddings_1.append(list(n2v_1.predict(node)))

# Your code here, you can add cells if necessary

import numpy as np

def compute_sim(n):

  '''
```

```
    n: the input node id
    '''
    cos_sims = []
    query_vec = np.array(embeddings_1[n])
    query_norm = np.linalg.norm(query_vec)
    for vec in embeddings_1:
        vec = np.array(vec)
        vec_norm = np.linalg.norm(vec)
        cos_sim = query_vec.dot(vec)/(query_norm*vec_norm)
        cos_sims.append(cos_sim)
    return cos_sims


#choose node id 5
cos_sims = compute_sim(5)


print(cos_sims)
# A barbell graph for your task 2.2
barbell_2 = nx.barbell_graph(1000, 50)
from sklearn.manifold import TSNE


def embed_plot(p,q,d,w,barbell):
    n2v = Node2Vec(p,q,d,w)
    n2v.fit(barbell)
    embeddings = []
    for node in barbell.nodes():
        embeddings.append(list(n2v.predict(node)))


    embeddings = np.array(embeddings)
    D2_embeddings = TSNE(n_components=2).fit_transform(embeddings)
    D2_embeddings = list(D2_embeddings)
```

```
    colors = ['red'] * 1025 + ['blue'] * 1025

    df = pd.DataFrame(D2_embeddings, columns = ['x', 'y']) # Create pandas dataframe from the list of
node embeddings

    df.plot.scatter(x = 'x', y = 'y', c = colors)



"""First trial: p=1,q=1,d=10,w=10"""



embed_plot(1,1,10,10,barbell_2)



"""Second trial: p=0.5, q=2, w=10"""



embed_plot(0.5,2,10,10,barbell_2)



"""Third trial: p=0.25, q=4, w=10"""



embed_plot(0.25,4,10,10,barbell_2)
```

```
[0.95672506, 0.9835775, 0.98355144, 0.9702097, 0.99063337, 1.0, 0.9942202, 0.988
51377, 0.9902109, 0.9579891, 0.97085565, 0.98084563, 0.9699146, 0.9924959, 0.985
111, 0.9611278, 0.9907803, 0.98641557, 0.9908302, 0.98803556, 0.99191266, 0.9762
144, 0.95510256, 0.97601366, 0.98544335, 0.9829972, 0.9642163, 0.96940297, 0.976
59653, 0.982286, 0.989187, 0.97506166, 0.9928044, 0.9776463, 0.9816332, 0.969260
4, 0.95689666, 0.97758186, 0.9829802, 0.981093, 0.97566545, 0.9901889, 0.9646456
, 0.98076254, 0.98994803, 0.9840095, 0.96004117, 0.99151266, 0.98865116, 0.97819
334, 0.9831172, 0.97796935, 0.97251517, 0.9912047, 0.9738431, 0.9899773, 0.98336
44, 0.98403305, 0.975035, 0.98180586, 0.9861538, 0.9873361, 0.9810986, 0.9796065
7, 0.9528672, 0.98424786, 0.9672934, 0.98248214, 0.9922681, 0.97492903, 0.985441
7, 0.9838424, 0.9694085, 0.9730892, 0.98502976, 0.9717703, 0.9864025, 0.98701626
, 0.9831852, 0.96711445, 0.97423816, 0.9798533, 0.976392, 0.98550534, 0.9785192,
 0.97512656, 0.9906792, 0.98273706, 0.9867744, 0.97545457, 0.99167955, 0.9787987
5, 0.97744584, 0.9722532, 0.9810087, 0.9740188, 0.978048, 0.9710262, 0.9844933,
0.97886914, 0.9874, 0.9910613, 0.9839432, 0.98785335, 0.9815363, 0.9808569, 0.97
75812, 0.9790733, 0.98155254, 0.9832307, 0.97688013, 0.97351843, 0.9842701, 0.97
447515, 0.9801656, 0.9838896, 0.9898756, 0.984159, 0.97369695, 0.98195547, 0.978
45197, 0.9797942, 0.98412967, 0.98280305, 0.97393644, 0.985969, 0.970695, 0.9922
2004, 0.98851407, 0.965459, 0.9936619, 0.98692304, 0.9788446, 0.9638142, 0.99158
59, 0.9887868, 0.9815921, 0.96270186, 0.97222954, 0.96962464, 0.9837628, 0.98060
53, 0.9879781, 0.9859664, 0.9803971, 0.9863131, 0.98765665, 0.9793824, 0.9826618
, 0.9896312, 0.9683084, 0.9828146, 0.9762269, 0.9834861, 0.9662713, 0.9911355, 0
.97520196, 0.9796913, 0.97166634, 0.9870896, 0.96262205, 0.9840186, 0.9809916, 0
.9761732, 0.9759456, 0.980602, 0.99046487, 0.9807757, 0.9852128, 0.99422145, 0.9
9159926, 0.9812934, 0.97970134, 0.9808714, 0.97261757, 0.98245114, 0.98587054, 0
.9959522, 0.9735435, 0.974296, 0.98040265, 0.9858981, 0.9771569, 0.98739386, 0.9
884011, 0.97370654, 0.98452234, 0.9899687, 0.97918606, 0.9850293, 0.9813132, 0.9
697845, 0.98976976, 0.99465615, 0.98116773, 0.9617691, 0.9964411, 0.9776869, 0.9
7909087, 0.97713983, 0.9938255, 0.98760617, 0.9769511, 0.9802676, 0.9650055, 0.9
874911, 0.9808624, 0.95571756, 0.9756189, 0.9766422, 0.9846311, 0.9835517, 0.980
37684, 0.9944757, 0.9947375, 0.98199975, 0.9728734, 0.99228054, 0.9715355, 0.982
70917, 0.9921328, 0.96188956, 0.979321, 0.97665447, 0.978815, 0.9925843, 0.97889
```

	03, 0.984678, 0.98235005, 0.99033976, 0.9747431, 0.98447895, 0.9679201, 0.990896 94, 0.98919165, 0.9645096, 0.97539634, 0.98814636, 0.98089814, 0.9873306, 0.9749 112, 0.96806836, 0.982021, 0.97712946, 0.99249846, 0.9739715, 0.96597534, 0.9882 1396, 0.98215127, 0.9910907, 0.9882724, 0.99607193, 0.9803965, 0.98939663, 0.992 30886, 0.9827938, 0.9924194, 0.9852215, 0.98448604, 0.9751929, 0.9793551, 0.9835 403, 0.9850054, 0.98553675, 0.9917322, 0.9832089, 0.98069304, 0.9832111, 0.98947 173, 0.99104714, 0.98468405, 0.98680365, 0.9912558, 0.9515881, 0.9779005, 0.9713 6146, 0.99464947, 0.9739354, 0.9893597, 0.98351365, 0.9740021, 0.9933302, 0.9864 3476, 0.97747934, 0.96692383, 0.96790254, 0.9831924, 0.97509634, 0.9884275, 0.98 63235, 0.9765905, 0.9678444, 0.97780436, 0.9693288, 0.99210364, 0.99411863, 0.98 50874, 0.97669446, 0.9912981, 0.9792682, 0.9703832, 0.99337626, 0.9741824, 0.981 34816, 0.9844287, 0.980898, 0.9789061, 0.96286416, 0.9905712, 0.98780245, 0.9819 2596, 0.97988844, 0.98745596, 0.98777723, 0.98939335, 0.98448855, 0.98515373, 0. 9808034, 0.97073156, 0.98569477, 0.994569, 0.98668677, 0.9781069, 0.98400724, 0. 98579365, 0.9731597, 0.96913207, 0.97764194, 0.97762966, 0.9859257, 0.99321944, 0.9749978, 0.994336, 0.97486395, 0.9800872, 0.9903623, 0.97996175, 0.975894, 0.9 692136, 0.9834468, 0.9764151, 0.9943515, 0.9829119, 0.99291253, 0.9699505, 0.988 5174, 0.9855702, 0.9898215, 0.96912074, 0.99533087, 0.98052526, 0.9827248, 0.989 65675, 0.9830105, 0.97474676, 0.98950773, 0.9815063, 0.97495216, 0.9714467, 0.98 27548, 0.98239356, 0.9762178, 0.9847832, 0.98268706, 0.9886109, 0.98574626, 0.98 101145, 0.9894659, 0.9835525, 0.97860986, 0.9902803, 0.990384, 0.9788547, 0.9899 675, 0.9766853, 0.9735909, 0.9800588, 0.9702967, 0.98236316, 0.9811739, 0.991798 5, 0.98490036, 0.99247247, 0.9564395, 0.97178286, 0.9635578, 0.989279, 0.9898608 3, 0.9596758, 0.9917356, 0.9706052, 0.96442413, 0.9719298, 0.9664798, 0.985185, 0.9793149, 0.99015206, 0.9790742, 0.96572727, 0.98095375, 0.9827991, 0.9880378, 0.9915198,0.124704055, 0.1220743, 0.13026327, 0.11422056] 

	 
Coclusion	We have successfully implemented page rank algorithm on real life dataset.