LINEAR ALGEBRA ASSIGNMENT-5

GAURAV MAHAJAN

PES1UG20CS150

SEC C

IMPLEMENTATION OF PAGE RANK ALGORITHM IN PYTHON:

1)GENERATING PAGE RANK PROBABILITIES IN THE FORM OF A MATRIX

```
def pagerank(G, alpha=0.85, personalization=None,
      max_iter=100, tol=1.0e-6, nstart=None, weight='weight',
      dangling=None):

  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()

  # Choose fixed starting vector if not given
  if nstart is None:
    x = dict.fromkeys(W, 1.0 / N)
  else:
    # Normalized nstart vector
    s = float(sum(nstart.values()))
    x = dict((k, v / s) for k, v in nstart.items())

  if personalization is None:

    # Assign uniform personalization vector if not given
    p = dict.fromkeys(W, 1.0 / N)
  else:
    missing = set(G) - set(personalization)
    if missing:
      raise NetworkXError('Personalization dictionary '
                'must have a value for every node. '
```

```
                'Missing nodes %s' % missing)
    s = float(sum(personalization.values()))
    p = dict((k, v / s) for k, v in personalization.items())

  if dangling is None:

    # Use personalization vector if dangling vector not specified
    dangling_weights = p
  else:
    missing = set(G) - set(dangling)
    if missing:
      raise NetworkXError('Dangling node dictionary '
                'must have a value for every node. '
                'Missing nodes %s' % missing)
    s = float(sum(dangling.values()))
    dangling_weights = dict((k, v/s) for k, v in dangling.items())
  dangling_nodes = [n for n in W if W.out_degree(n, weight=weight) == 0.0]

  # power iteration: make up to max_iter iterations
  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:

      # this matrix multiply looks odd because it is
      # doing a left multiply x^T=xlast^T*W
      for nbr in W[n]:
        x[nbr] += alpha * xlast[n] * W[n][nbr][weight]
      x[n] += danglesum * dangling_weights[n] + (1.0 - alpha) * p[n]

    # check convergence, l1 norm
    err = sum([abs(x[n] - xlast[n]) for n in x])
    if err < N*tol:
```

```
        return x
    raise NetworkXError('pagerank: power iteration failed to converge in %d iterations.' % max_iter)
```

```
[ ] import networkx as nx
    G=nx.barabasi_albert_graph(60,41)
    pr=nx.pagerank(G,0.4)
    pr
```

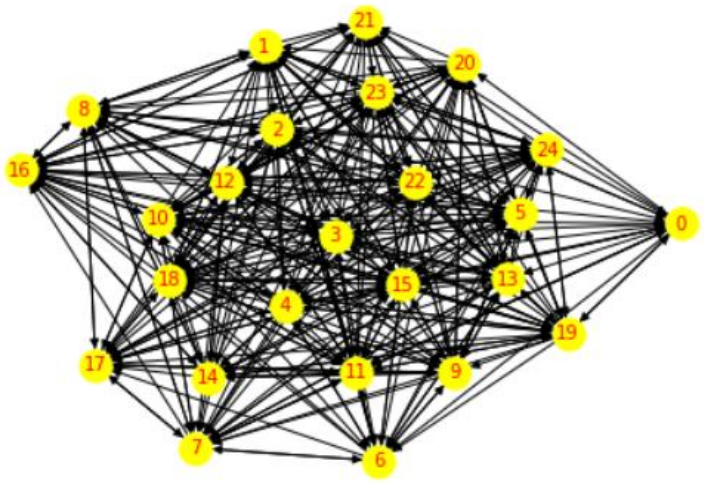
```
{0: 0.027974270717457006,
 1: 0.0127628368839034,
 2: 0.013176723444143786,
 3: 0.013177747527072622,
 4: 0.013187466898160255,
 5: 0.013161209959699736,
 6: 0.0127628368839034,
 7: 0.012562160948942153,
 8: 0.013370175461666607,
 9: 0.013367423527823493,
 10: 0.012966440442363685,
 11: 0.012572083959984303,
 12: 0.012981177194839124,
 13: 0.013357664225546649,
 14: 0.013162784091391478,
 15: 0.012794116114652159,
 16: 0.012746551076996878,
 17: 0.012966525434565043,
 18: 0.01299082083909732,
 19: 0.013358460968529162,
 20: 0.012559926628535813,
 21: 0.012385097979873872,
 22: 0.012981843309988613,
 23: 0.012985402118596601,
 24: 0.012961209517747601,
 25: 0.012972463934502566,
 26: 0.013162787965996983,
 27: 0.013166775312276354,
 28: 0.011946861285440464,
 29: 0.01297171164976596,
 30: 0.012767283970356874,
 31: 0.012742473235933041,
 32: 0.012774581194681032,
 33: 0.013165728970937722,
 34: 0.012559423699148328,
 35: 0.012574366399099425,
 36: 0.013178994660872476,
 37: 0.013162447872856397,
```

```
[ ]
    38: 0.012546618649433343,
    39: 0.0131727390026888,
    40: 0.013565732355562396,
    41: 0.013150392838513414,
    42: 0.02758678002625911,
    43: 0.02727992233732545,
    44: 0.026720909580798917,
    45: 0.026542949274437247,
    46: 0.026140850136958396,
    47: 0.02585214873990138,
    48: 0.02543710445857681,
    49: 0.02492353658663134,
    50: 0.02471539936160183,
    51: 0.024186424530845164,
    52: 0.023684717633201,
    53: 0.02352070217524788,
    54: 0.02306055382593085,
    55: 0.022830596519100375,
    56: 0.02288679193854571,
    57: 0.0222811536758942,
    58: 0.02177521090987354,
    59: 0.021719909135324508}
```

2.IMPLEMENTATION BY USING A DIRECTED GRAPH TO VERIFY THE ALGORITHM USING NETWORKX LIBRARY AND THE WRITTEN FUNCTION

```
import networkx as nx
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import operator
import random as rd

# created a directed graph
graph=nx.gnp_random_graph(25,0.6,directed=True)

#draw a graph
nx.draw(graph,with_labels=True,font_color='red',font_size=10,node_color='yellow')

#plot a graph

plt.show()

#number of nodes for graph
count=graph.number_of_nodes()
#graph neighbours of a node 1
print(list(graph.neighbors(1)))

#Page Rank Algorithm-Calculating random walk score
rank_dict={}
x=rd.randint(0,25)
for j in range(0,25):
  rank_dict[j]=0
rank_dict[x]=rank_dict[x]+1
for i in range(600000):
  list_n=list(graph.neighbors(x))
  if(len(list_n)==0):
    x=rd.randint(0,25)
    rank_dict[x]=rank_dict[x]+1
  else:
```

```
    x=rd.choice(list_n)
    rank_dict[x]=rank_dict[x]+1

print("Random Walk Score Updated")

#normalising values
for j in range(0,25):
  rank_dict[j]=rank_dict[j]/600000

#Page rank by networkx library
pagerank=nx.pagerank(graph)

#sorting both dictionaries based on items
pagerank_sorted=sorted(pagerank.items(),key=lambda v:(v[1],v[0]),reverse=True)

pagerank_sorted

#sorting the rank_dict based on values
rank_dict_sorted=sorted(rank_dict.items(),key=lambda v:(v[1],v[0]),reverse=True)

rank_dict_sorted

print("The order generated by our implementation algorithm is\n")
for i in rank_dict_sorted:
  print(i[0],end=" ")
print("\n\nThe order generated by networkx library is\n")
for i in pagerank_sorted:
  print(i[0],end=" ")
```




```
[0, 2, 3, 5, 6, 8, 11, 12, 13, 16, 18, 21, 22, 23, 24]
Random Walk Score Updated
The order generated by our implementation algorithm is

18 16 11 23 5 1 12 8 10 24 20 19 7 15 9 21 17 3 14 13 2 22 6 4 0

The order generated by networkx library is

18 16 11 23 1 12 5 10 8 24 7 20 19 15 9 17 21 3 14 13 2 22 6 4 0
```