# Assignment 5

CS834-F16: Introduction to Information Retrieval

Fall 2016

Erika Siregar

CS Department - Old Dominion University

December 17, 2016

## Question 10.3

`Compute five iterations of HITS (see Algorithm 3) and PageRank (see Figure 4.11) on the graph in Figure 10.3. Discuss how the PageRank scores compare to the hub and authority scores produced by HITS.`

### Answer

Figure 1 shows the directed graph from the textbook [1] on which we will calculate the scores of HITS and PageRank. Computing HITS (authorities and hubs) and PageRank scores are pretty easy since we can just utilize the Link Analysis procedure that is provided by python library 'networkx' [2].

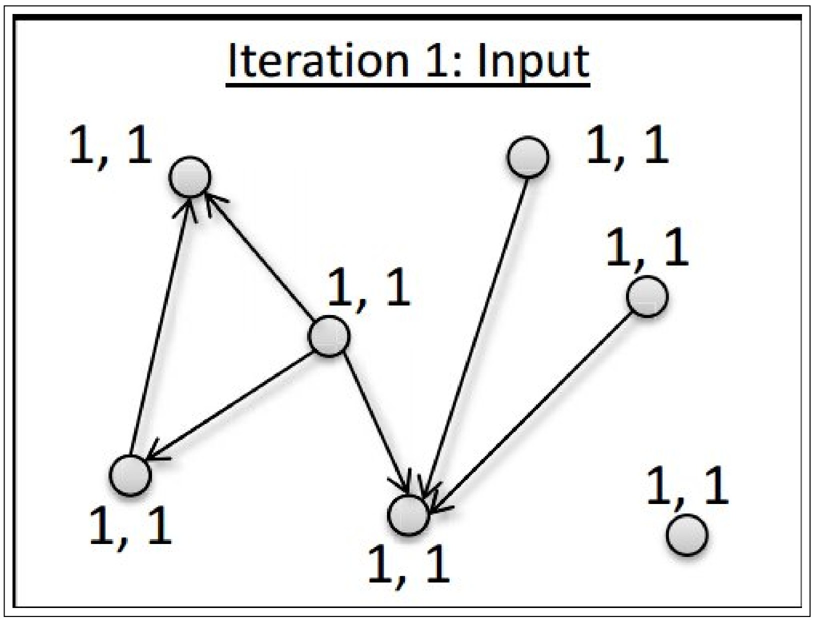


Figure 1: Figure 10.3 from the textbook [1]

Figure 2 shows the scores of HITS (authorities and hubs) and PageRank, which are obtained by running the code in listing 1. We only need to set the number of iterations.

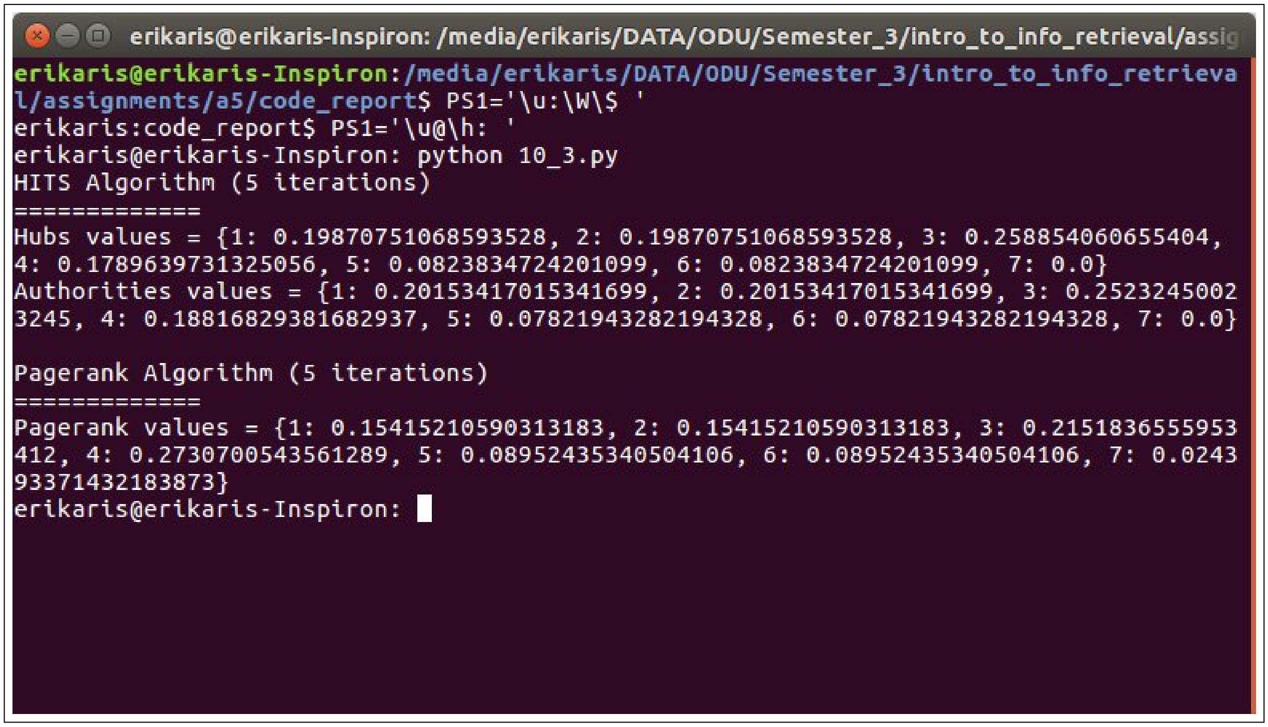


Figure 2: HITS and Pagerank for Figure 10.3 with 5 Iterations

To make the analysis and comparison easier, I transformed the output int figure 2 into a neat table format as can be seen on table 1. From table 1, we can see that, generally, the authorities values are linearly proportional to those of PageRank. After 5 iterations, node 3 gets the highest score for 'authorities' and the second highest score for 'PageRank'. Nodes 1 and 2 get lower 'authorities' score than that of node 3, but higher 'authorities' score compare to nodes 5 and 6. The same thing can also be concluded by comparing the PageRank scores for those five nodes (1, 2, 3, 5, and 6). The strange thing happens on node 4, where its 'authorities' score is lower than node 3, but its 'PageRank' score is higher than node 3. This anomaly takes place probably because we only do 5 iterations. Maybe, if we continue iterating until the values converge into certain number, this anomaly will not happen.

| **Node** | **Score** | | |
|---|---|---|---|
| | **Hubs** | **Authorities** | **PageRank** |
| 1 | 0.198707510685935 | 0.201534170153416 | 0.154152105903131 |
| 2 | 0.198707510685935 | 0.201534170153416 | 0.154152105903131 |
| 3 | 0.258854060655404 | 0.252324500232450 | 0.215183655595341 |
| 4 | 0.178963973132505 | 0.188168293816829 | 0.273070054356128 |
| 5 | 0.082383472420110 | 0.078219432821943 | 0.089524353405041 |
| 6 | 0.082383472420110 | 0.078219432821943 | 0.089524353405041 |
| 7 | 0.000000000000000 | 0.000000000000000 | 0.024393371432184 |

Table 1: HITS and Pagerank for Figure 10.3 with 5 Iterations

```
#!/usr/bin/python

import networkx as nx

```

```
def hits(G, iter=100, nstart=None, normalized=True):
if type(G) == nx.MultiGraph or type(G) == nx.MultiDiGraph:
raise Exception("hits() not defined for graphs with multiedges.")
if len(G) == 0:
return {},{}
# choose fixed starting vector if not given
if nstart is None:
h=dict.fromkeys(G,1.0/G.number_of_nodes())
else:
h=nstart
# normalize starting vector
s=1.0/sum(h.values())
for k in h:
h[k]*=s
i=0
while True: # power iteration: make up to max_iter iterations
if i >= iter: break

hlast=h
h=dict.fromkeys(hlast.keys(),0)
a=dict.fromkeys(hlast.keys(),0)
# this "matrix multiply" looks odd because it is
# doing a left multiply a^T=hlast^T*G
for n in h:
for nbr in G[n]:
a[nbr]+=hlast[n]*G[n][nbr].get('weight',1)
# now multiply h=Ga
for n in h:
for nbr in G[n]:
h[n]+=a[nbr]*G[n][nbr].get('weight',1)
# normalize vector
s=1.0/max(h.values())
for n in h: h[n]*=s
# normalize vector
s=1.0/max(a.values())
for n in a: a[n]*=s

i+=1
if normalized:
s = 1.0/sum(a.values())
for n in a:
a[n] *= s
s = 1.0/sum(h.values())
for n in h:
h[n] *= s
return h,a

def pagerank(G, alpha=0.85, personalization=None,
iter=100, 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 nx.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 nx.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(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]

return x

if __name__ == '__main__':
iter = 5
G = nx.Graph()

# Add 7 nodes
G.add_nodes_from(range(1,8))

# Add 6 edges
G.add_edges_from([(1,2), (3,1), (3,2), (3,4), (5,4), (6,4)])
```

```

# Compute hubs and authorities normalized values using hits
h, a = hits(G, iter=iter)

print 'HITS Algorithm ({} iterations)'.format(iter)
print '=================='
print 'Hubs values = {}'.format(h)
print 'Authorities values = {}'.format(a)
print ''

# Compute pagerank of each nodes
pr = pagerank(G, iter=iter)

print 'Pagerank Algorithm ({} iterations)'.format(iter)
print '=================='
print 'Pagerank values = {}'.format(pr)
```

Listing 1: Computing HITS and PageRank

---

## Question 0.3

### Answer:

answer 2

---

## Question 8.5

`question 3`

### Answer

answer 3

---

## Question 8.7

`question 4`

### Answer

answer 4

## References

[1] Bruce Croft, Donald Metzler, and Trevor Strohman. *Search Engines: Information Retrieval in Practice*. Addison-Wesley Publishing Company, USA, 1st edition, 2009.

[2] NetworkX Developers. Networkx - Link Analysis. https://networkx.github.io/documentation/networkx-1.9/reference/algorithms.link_analysis.html, 2016. [Online; accessed 14-December-2016].