PageRank Lecture Note

Keshi Dai

June 22, 2009

1 Motivation

Back in 1990s, the occurrence of the keyword is the only important rule to judge if a document is relevant or not. The document with the highest number of occurrences of keywords receives the highest score based on the traditional text retrieval model. This approach works fine on text retrieval, but it has its flaws. It only looks the content of a document, but ignore its influence. All documents in the collection are seen as equally important. In the large-scale web, this may undermine the retrieval quality. For example, if you search "harvard" in your browser, you would expect that your search engine ranks the homepage of the Harvard University as the most relevant page. Suppose word "harvard" appears much more often in a Harvard student's homepage than in "www.harvard.edu" because that student listed all courses he has taken in Harvard, and all papers he has published, etc, which all contain "harvard". Should we consider this student's homepage is more relevant than "www.harvard.edu" to our query. In worst scenario, if we create a web page that contains "harvard" a million times. Should we consider this page is relevant to the query "harvard"? The answer is of course not.

2 PageRank

In 1998, Larry Page and Sergey Brin, two graduate students at Stanford University, has invented the PageRank algorithm that model the structure of pages on the web and quantize the importance of each page. PageRank is one of the most known and influential algorithms for computing the relevance of web pages, and is used by Google, the most successful search engine on the web. The basic idea of PageRank is that the importance of a web page depends on the pages that link to it. For instance, we create a web page i that includes a hyperlink to web page j. If there are a lot pages also link to j, we then consider j is important on the web. On the hand, if j only has one in-link, however, this link is from an authoritative web page k (like www.google.com, www.yahoo.com, or www.bing.com), we also think j is important because k can transfer its popularity or authority to j.

Suppose for instance we have the following directed graph based on a tiny web (see Figure 1) that have only 6 pages, one for each node. When web page i references j, we add a directed edge between node i and j in the graph. In PageRank model, each page should transfer evenly its importance to the pages that it links to. For example, page A has 3 out-links, so it will pass on $\frac{1}{3}$ of its importance to B, C, and F. In general, if a page has k

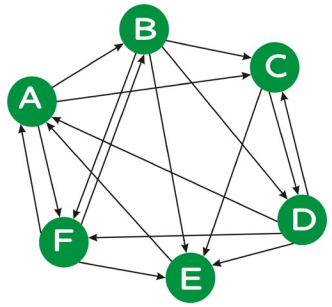

Figure 1: A tiny web with 6 pages

out-links, it will pass on $\frac{1}{k}$ of its importance to each of the pages that it links to. According to this importance transition rule, we can define the transition matrix of the graph, say P,

$$P = \begin{bmatrix} 0 & 0 & 0 & \frac{1}{4} & 1 & \frac{1}{3} \\ \frac{1}{3} & 0 & 0 & 0 & 0 & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{4} & 0 & \frac{1}{4} & 0 & 0 \\ 0 & \frac{1}{4} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{4} & 0 & \frac{1}{4} & 0 & 0 \end{bmatrix}$$

Starting with the uniform distribution, the importance of each node is $\frac{1}{6}$. Let π denote the initial PageRank value vector, having all entries equal to $\frac{1}{6}$. Because each incoming link increase the PageRank value of a web page, we update the rank of each page by adding to the current value the importance of the incoming links. This is the same as multiplying the matrix P with v. Nummeric computation give [1]:

$$\pi \begin{pmatrix} 0.167 \\ 0.167 \\ 0.167 \\ 0.167 \\ 0.167 \\ 0.167 \end{pmatrix}, P\pi = \begin{pmatrix} 0.264 \\ 0.111 \\ 0.139 \\ 0.125 \\ 0.222 \\ 0.139 \end{pmatrix}, P^2\pi = \begin{pmatrix} 0.300 \\ 0.134 \\ 0.147 \\ 0.097 \\ 0.175 \\ 0.147 \end{pmatrix}, \cdots, P^{12}\pi = \begin{pmatrix} 0.265 \\ 0.138 \\ 0.150 \\ 0.110 \\ 0.187 \\ 0.150 \end{pmatrix}, P^{13}\pi = \begin{pmatrix} 0.265 \\ 0.138 \\ 0.150 \\ 0.110 \\ 0.187 \\ 0.150 \end{pmatrix}$$

We can the sequences of iterations π, $P\pi$, $\cdots$, $P^k\pi$ tends to converge to the value $\pi^* = \begin{pmatrix} 0.265 \\ 0.138 \\ 0.150 \\ 0.110 \\ 0.187 \\ 0.150 \end{pmatrix}$. This is the PageRank vector of our web graph.

3 Markov chain

The popularity of a web page can also be viewed as the probability of visiting this page during a random surfer on the Internet. A web page with high populairty has more chances

of being visited than a web page with low populairty. Because a popular page has many pages linking to it, if you visit one of them, you will have a chance of vising this popular page. We can model this process as a random walk on a Markov chain. All pages starts with the uniform distribution, so $\pi = [0.167, 0.167, 0.167, 0.167, 0.167, 0.167]^T$ and P is the transition matrix of this Markov chain. The probability that page i will be visited after one step is equal to $P\pi$,. The probability that page i will be visited after k steps is $P^k\pi$. The sequence $P\pi$, $P^2\pi$, $P^3\pi$, $\cdots$, $P^k\pi$, $\cdots$ converges in our example to a unique probabilistic vector v^*, and π^* is the stationary distribution and it will be our PageRank values.

4 Egeinvector

We can also model this problem in the linear algebra point of view [3]. Let x_1, x_2, $\cdots$, x_6 be the importance of 6 pages in our graph. Because the importance of a page is summation of importances from all pages that link to it, we get:

$$\begin{cases} x_1 = \frac{1}{4} \cdot x_4 + 1 \cdot x_5 + \frac{1}{3} \cdot x_6 \\ x_2 = \frac{1}{3} \cdot x_1 + \frac{1}{3} \cdot x_6 \\ x_3 = \frac{1}{3} \cdot x_1 + \frac{1}{4} \cdot x_2 + \frac{1}{4} \cdot x_4 \\ x_4 = \frac{1}{4} \cdot x_2 + \frac{1}{2} \cdot x_3 \\ x_5 = \frac{1}{4} \cdot x_2 + \frac{1}{2} \cdot x_3 + \frac{1}{4} \cdot x_4 + \frac{1}{3} \cdot x_6 \\ x_6 = \frac{1}{3} \cdot x_1 + \frac{1}{4} \cdot x_2 + \frac{1}{4} \cdot x_4 \end{cases}$$

This is equivalent to solve the equations $P \cdot \pi = \pi$, where $\pi = [x_1, x_2, x_3, x_4, x_5, x_6]^T$. We know that π is the eigenvector corresponding to the eigenvalue 1. Normalizing π, so it would be the unique eigenvector with the sum of all entries equal to 1, also known as the probabilistic eigenvector. It is also our PageRank vector.

5 Dangling nodes and disconnected components

Extend our simple example, suppose that there some pages that do not have any out-links (we call them dangling nodes), our random surfer will get stuck on these pages, and the importance received by these pages cannot be propagated. In the other senario, if our web graph has two disconnected components, the random surfer that starts from one component has no way to get into the other component. All pages in other component will receive 0 importance.

Dangling nodes and disconnected components actually are quite common on the Internet, considering the large scale of the web. In order to deal with these two problems, a positive constant d between 0 and 1 (typically 0.15) is introduced, which we call the damping factor [2]. Now we modify previous transition matrix based on d into $P' = (1-d) \cdot P + d \cdot R$, where

$$R = \frac{1}{N} \cdot \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix}$$

This new transition matrix models the random walk as follows: most of the time, a surfer will follow links from a page if that page has outgoing links. A smaller, but positive

percentage of the time, the surfer will dump the current page and choose arbitrarily a different page from the web, and "teleports" there. The damping factor d reflects the probability that the surfer quits the current page and "teleports" to a new one. Since every page can be teleported, each page has $\frac{1}{n}$ probability to be chosen. This justifies the structure of R.

6 Implementation

The PageRank formula based on the previous discussion is as follows:

$$PR(p_i) = \frac{1-d}{N} + d(\sum_{p_j \text{ links to } p_i} \frac{PR(p_j)}{L(p_j)} + \sum_{p_j \text{ has no out-links}} \frac{PR(p_j)}{N})$$

Here is the pseudocode of my implementation of PageRank algorithm:

Algorithm 1 PageRank algorithm

```
1: procedure PageRank(G, iteration)        ▷ G: inlink file, iteration: # of iteration
2:     d ← 0.85                                              ▷ damping factor: 0.85
3:     oh ← G                                          ▷ get outlink count hash from G
4:     ih ← G                                           ▷ get inlink hash from G
5:     N ← G                                            ▷ get # of pages from G
6:     for all p in the graph do
7:         opg[p] ← 1/N                                      ▷ initialize PageRank
8:     end for
9:     while iteration > 0 do
10:        dp ← 0
11:        for all p that has no out-links do
12:            dp ← dp + d * opg[p]/N          ▷ get PageRank from pages without out-links
13:        end for
14:        for all p in the graph do
15:            npg[p] ← dp + (1−d)/N                ▷ get PageRank from random jump
16:            for all ip in ih[p] do
17:                npg[p] ← npg[p] + d*opg[ip]/oh[ip]        ▷ get PageRank from inlinks
18:            end for
19:        end for
20:        opg ← npg                                          ▷ update PageRank
21:        iteration ← iteration − 1
22:    end while
23: end procedure
```

A PageRank Source Code

```
#! /usr/bin/python

"""
```

```
 Who:    Keshi Dai
 What:   PageRank.py
 When:   06/20/09
 Usage: PageRank.py [-t] [-i iteration_num] inlink_file > output
"""

import sys

if len(sys.argv)==1:
    print >> sys.stderr,
       "Usage: PageRank.py [-t] [-i iteration_num] inlink_file > output\n"
    sys.exit()

if sys.argv[1] == "-t":
    teleport = True
    if sys.argv[2] == "-i":
        iternum = int(sys.argv[3])
        inlink_file_name = sys.argv[4]
    else:
        iternum = 10
        inlink_file_name = sys.argv[2]
else:
    teleport = False
    inlink_file_name = sys.argv[1]

#damping factor
d=0.85

inlink_file = open(inlink_file_name, 'r')
outlink_count = {}
inlinks = {}
oldpagerank = {}
newpagerank = {}
docids = {}
dangling_docs = {}

print >> sys.stderr, "Processing inlink file", inlink_file_name, "....."

inlink_docnum = 0
outlink_docnum = 0
docnum = 0

for line in inlink_file:
    line = line.strip();
    nodes = line.split(" ");
    inlink_docnum += 1
```

```
    inlinks[nodes[0]] = [];
    inlinks[nodes[0]] = tuple(nodes[1:]);
    if not docids.has_key(nodes[0]):
        docids[nodes[0]] = 1
        docnum += 1
    for node in nodes[1:]:
        if outlink_count.has_key(node):
            outlink_count[node] += 1
        else:
            outlink_count[node] = 1
            outlink_docnum += 1
        if not docids.has_key(node):
            docids[node] = 1;
            docnum += 1

print >> sys.stderr, "Number of Documents:", docnum
print >> sys.stderr, "Number of Documents with in-links:", inlink_docnum
print >> sys.stderr, "Number of Documents with out-links:", outlink_docnum

for key in docids.keys():
    if not inlinks.has_key(key):
        inlinks[key] = ()

    oldpagerank[key] = 1.0/docnum
    if not outlink_count.has_key(key):
        dangling_docs[key] = 1
print >> sys.stderr, "Number of Dangling Documents:", len(dangling_docs)

while iternum > 0:
    if teleport:
        dp = 0
        for key in dangling_docs.keys():
            dp += d*oldpagerank[key]/docnum

    for key in oldpagerank.keys():
        if teleport:
            newpagerank[key] = (1-d)/docnum + dp
        else:
            newpagerank[key] = (1-d)/docnum
        for inlink in inlinks[key]:
            if outlink_count.has_key(inlink):
                newpagerank[key] += d*oldpagerank[inlink]/outlink_count[inlink]

    for key in newpagerank.keys():
        oldpagerank[key] = newpagerank[key]
    iternum -= 1
```

```
    print >> sys.stderr, "PageRank iteration remaining", iternum
for key in newpagerank.keys():
    print key, newpagerank[key]
```

References

[1] Pagerank algorithm. `http://www.math.cornell.edu/~mec/Winter2009/RalucaRemus/Lecture3/lecture3.html`.

[2] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

[3] K. Bryan and T. Leise. The$ 25,000,000,000 Eigenvector: The Linear Algebra behind Google. *SIAM REVIEW*, 48(3):569, 2006.