PageRank Algorithm

Max Turgeon

STAT 4690–Applied Multivariate Analysis

Quick history of search engines i

- At the beginning of times (circa 1995), search engines mostly ranked web pages based on their content:
    - Keywords they contained.
    - Link to other pages (i.e. hubs)
    - Who's behind the website
- For example, if I search for "Principal Component Analysis", you could score each web page depending on how often you see the keywords "principal", "component", "analysis", and you could increase the weight for websites from statisticians, or for websites on general statistical concepts.

Quick history of search engines ii

- In contrast, the PageRank algorithm proposes to look at *other* pages to understand how *important* a given web page is.

Motivation i

- The PageRank algorithm gives a score PR_i to each web page.
- This score depends on how often page i is linked to from a page j.
    - We'll write $j \to i$ to say that web page j links to web page i.

Motivation ii

- The score follows the two following rules:
  - Being linked to from an important web page should count **more**. In other words, PR_i should be proportional to PR_j.
  - Being linked to from a web page that has many, many links, should count **less**. In other words, PR_i should be inversely proportional to the number of links m_j going out of page j.

First version of the algorithm

- Let's look at a first attempt to implementing such a scoring rule.
    - We'll see later that it doesn't quite work...
- We use a **recursive** definition:

$$PR_i = \sum_{j \to i} \frac{PR_j}{m_j}.$$

- If we let L_{ij} be equal to 1 if $j \to i$ and 0 otherwise, we can rewrite this as:

$$PR_i = \sum_{j=1}^{n} \frac{L_{ij}}{m_j} PR_j.$$

Example i

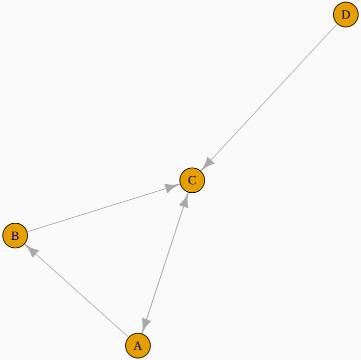
```r
(Lmat <- matrix(c(0, 0, 1, 0, 1, 0, 0, 0,
                  1, 1, 0, 1, 0, 0, 0, 0), ncol = 4,
                byrow = TRUE))
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    0    0    1    0
## [2,]    1    0    0    0
## [3,]    1    1    0    1
## [4,]    0    0    0    0
```

Example ii

```
# Alternatively
library(igraph)
edge_list <- c(c("A", "B"), c("A", "C"), c("B", "C"),
               c("C", "A"), c("D", "C"))
graph <- graph_from_edgelist(
  matrix(edge_list, ncol = 2,
         byrow = TRUE)
  )
plot(graph)
```

Example iii



Example iv

```r
m_vect <- colSums(Lmat)
# Initialize
pr_vect <- rep(0.25, 4)
for (i in seq_len(4)) {
  pr_vect[i] <- sum(Lmat[i,]*pr_vect/m_vect)
}
pr_vect
```

```
## [1] 0.250 0.125 0.500 0.000
```

Example v

```
# Repeat 5 more times
for(count in seq_len(5)) {
  for (i in seq_len(4)) {
    pr_vect[i] <- sum(Lmat[i,]*pr_vect/m_vect)
    }
}
pr_vect
```

```
## [1] 0.50 0.25 0.50 0.00
```

Matrix notation

- Let $p = (PR_1, \ldots, PR_n)$, let L be the matrix whose (i, j)-th entry is L_{ij}, and let M be the diagonal matrix whose i diagonal element is m_i.
- Then, we can write the equations $PR_i = \sum_{j=1}^{n} \frac{L_{ij}}{m_j} PR_j$ for all i as

$$p = LM^{-1}p.$$

- In other words, we can find the vector of scores by finding the eigenvector corresponding to eigenvalue 1 (if it exists!).

Example (cont’d) i

```
Amat <- Lmat %*% diag(1/m_vect)
decomp <- eigen(Amat)
decomp$values
```

```
## [1]  1.0+0.0i -0.5+0.5i -0.5-0.5i  0.0+0.0i
```

```
Re(decomp$vectors[,1])
```

```
## [1] -0.6666667 -0.3333333 -0.6666667  0.0000000
```

Example (cont'd) ii

```r
# Which is proportional to what we found
pr_vect
```

```
## [1] 0.50 0.25 0.50 0.00
```

```r
Re(decomp$vectors[,1])/-(4/3)
```

```
## [1] 0.50 0.25 0.50 0.00
```

Comment

- We have an intuitive definition of the score.
- Even if its recursive, we can still compute the score
    - Either through an iterative process.
    - Or as an eigenvector problem.
- But the solution is not very satisfactory on our example:
    - Looking at the graph, page C seems the most important.
    - But our solution gives page A as the most important.

Markov chains i

- To gain more insight into this problem, it's helpful to rephrase it as a discrete Markov process.
- Each web page is one of the states of the Markov chain.
- Imagine a random surfer, i.e. someone is randomly (and uniformly) go from one web page to another by following links.
  - For simplicity, assume that a given web page as at most one link to another one.

Markov chains ii

- Then the probability of going from page i to j in one step is given by

$$P_{ij} = P(\text{landing on } j \mid \text{currently on } i) = \begin{cases} \frac{1}{m_i} & \text{if } i \to j \\ 0 & \text{else} \end{cases}.$$

- Since each column of the matrix P sums to 1, it follows that $e = (1, \ldots, 1)$ is an eigenvector of P^T with eigenvalue 1.
    - Therefore, 1 is also an eigenvalue of P and there exists a vector p such that $Pp = p$.

Strongly connected Markov chains

- We say is a Markov chain is **strongly connected** (or **irreducible**) if you can get from any state from any other state.
- For strongly connected Markov chains, the stationary distribution exists and is unique.
- Unfortunately, our random surfer model does not lead to a strongly connected Markov chain in general:
    - You may have clusters of pages that do not link to one another.
    - You may have pages without outgoing links.

Second version of the algorithm i

- We will modify the random surfer model to create a strongly connected Markov chain.
- They will either click randomly on a link **or** randomly jump to another page (without clicking a link.)
- Let $0 < d < 1$ be a constant, and define

$$P_{ij} = P(\text{landing on } j \mid \text{currently on } i) = \begin{cases} \frac{1-d}{n} + \frac{d}{m_i} & \text{if } i \to j \\ \frac{1-d}{n} & \text{else} \end{cases}.$$

Second version of the algorithm ii

- In matrix form: let L, M be as above, let P be as we just redefined, and let E be the $n \times n$ matrix containing only 1s. Then we are looking for a vector p of scores such that

$$p = \left(\frac{(1-d)}{n} E + dLM^{-1} \right) p.$$

Example (cont’d) i

```r
d <- 0.85
n <- length(m_vect)
Amat <- (1 - d) * matrix(1, ncol = n, nrow = n)/n +
  d * Lmat %*% diag(1/m_vect)
decomp <- eigen(Amat)
decomp$values
```

```
## [1]  1.000000e+00+0.00e+00i -4.250000e-01+4.25e-01i
## [4] -1.422971e-17+0.00e+00i
```

```r
Re(decomp$vectors[,1])
```

Example (cont'd) ii

```
## [1] -0.64470397 -0.33889759 -0.68212419 -0.06489841
```

```r
page_rank(graph, damping = 0.85)$vector
```

```
##         A         B         C         D 
## 0.3725269 0.1958239 0.3941492 0.0375000
```

```r
Re(decomp$vectors[,1])/(sum(Re(decomp$vectors[,1])))
```

```
## [1] 0.3725269 0.1958239 0.3941492 0.0375000
```

Twitter example i

- Data comes from *Stanford Network Analysis Project*.
    - Downloaded from `https://eecs490.github.io/project-pagerank/index.html`
- It is data about connections between Twitter accounts
    - Who follows whom.
- I've randomly selected 10,000 nodes and only kept edges for which both nodes are in this subset.

Twitter example ii

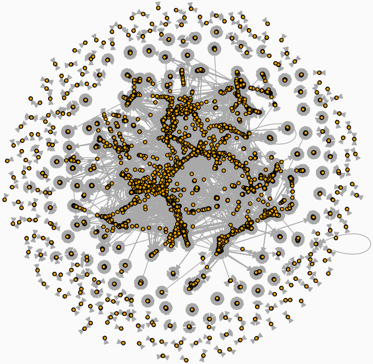
```r
library(tidyverse)
url <- paste0("https://raw.githubusercontent.com/",
        "turgeonmaxime/twitter_pagerank_STAT4690/",
              "master/edge_list_subset.csv")
edge_sub <- read_csv(url) %>%
  mutate(Node_Id_1 = as.character(Node_Id_1),
         Node_Id_2 = as.character(Node_Id_2)) %>%
  as.matrix()
```

Twitter example iii

```r
graph_twitter <- graph_from_edgelist(edge_sub)

plot(graph_twitter, vertex.label = NA,
     vertex.size = 2, edge.arrow.size = 0.5)
```

Twitter example iv



Twitter example v

```r
# PageRank algorithm
PR_vect <- page_rank(graph_twitter)$vector

# Visualize scores
library(colorspace)
quartiles <- quantile(PR_vect,
                      probs = seq(0, 1, length.out=5))
colours <- cut(PR_vect, breaks = quartiles,
               labels = sequential_hcl(4),
               include.lowest = TRUE)
```

Twitter example vi

```r
plot(graph_twitter, vertex.label = NA,
     vertex.size = 2, edge.arrow.size = 0.5,
     vertex.color = as.character(colours))
```

Twitter example vii