Data Sci Discover Project

Ryan Greenup

August 7, 2020

Contents

Working

```
if (require("pacman")) {
    library(pacman)
  }else{
    install.packages("pacman")
    library(pacman)
  }
```

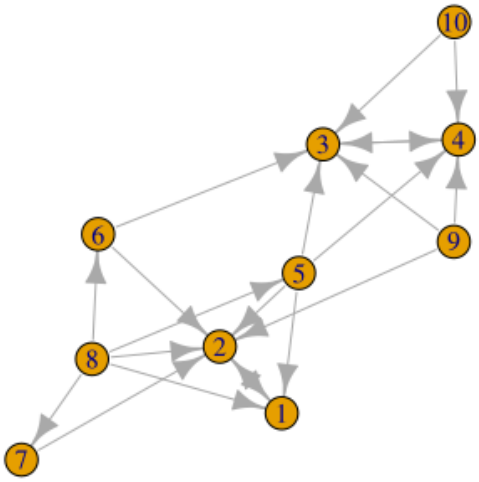
```
  pacman::p_load(tidyverse, Matrix, igraph, plotly, mise, docstring)
```

Implementing Page Rank Method on a small graph

Example Graph

Consider the following Graph taken from the paper:

```
g1 <- igraph::graph.formula(1++2, 1+-8, 1+-5, 2+-5, 2+-7, 2+-8, 2+-6,
  ↪  2+-9, 3++4, 3+-5, 3+-6, 3+-9, 3+-10, 4+-9, 4+-10, 4+-5, 5+-8,
  ↪  6+-8, 7+-8)
plot(g1)
```



1. Adjacency Matrix The adjacency Matrix is given by:

```
A <- igraph::get.adjacency(g1, names = TRUE, sparse = FALSE) %>%
  as.matrix()

## Adjust the Order
(A <- A[order(as.integer(row.names(A))),
  ↪  order(as.integer(colnames(A)))])
```

```
##    1 2 3 4 5 6 7 8 9 10
## 1  0 1 0 0 0 0 0 0 0  0
## 2  1 0 0 0 0 0 0 0 0  0
## 3  0 0 0 1 0 0 0 0 0  0
## 4  0 0 1 0 0 0 0 0 0  0
## 5  1 1 1 1 0 0 0 0 0  0
```

```
## 6  0 1 1 0 0 0 0 0 0  0
## 7  0 1 0 0 0 0 0 0 0  0
## 8  1 1 0 0 1 1 1 0 0  0
## 9  0 1 1 1 0 0 0 0 0  0
## 10 0 0 1 1 0 0 0 0 0  0
```

2. State Distribution The state distribution is the transpose of the adjacency matrix:

```
(p0 <- t(A))
```

```
##    1 2 3 4 5 6 7 8 9 10
## 1  0 1 0 0 1 0 0 1 0  0
## 2  1 0 0 0 1 1 1 1 1  0
## 3  0 0 0 1 1 1 0 0 1  1
## 4  0 0 1 0 1 0 0 0 1  1
## 5  0 0 0 0 0 0 0 1 0  0
## 6  0 0 0 0 0 0 0 1 0  0
## 7  0 0 0 0 0 0 0 1 0  0
## 8  0 0 0 0 0 0 0 0 0  0
## 9  0 0 0 0 0 0 0 0 0  0
## 10 0 0 0 0 0 0 0 0 0  0
```

3. Probability Transition Matrix The probability transition matrix is such that each column of the initial state distribution (i.e.ăthe transposed adjacency matrix) is scaled to 1.

```
p0 %*% diag(1/colSums(p0))
```

```
##    [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]      [,9] [,10]
## 1     0    1    0    0 0.25  0.0    0  0.2 0.0000000   0.0
## 2     1    0    0    0 0.25  0.5    1  0.2 0.3333333   0.0
## 3     0    0    0    1 0.25  0.5    0  0.0 0.3333333   0.5
## 4     0    0    1    0 0.25  0.0    0  0.0 0.3333333   0.5
## 5     0    0    0    0 0.00  0.0    0  0.2 0.0000000   0.0
## 6     0    0    0    0 0.00  0.0    0  0.2 0.0000000   0.0
## 7     0    0    0    0 0.00  0.0    0  0.2 0.0000000   0.0
## 8     0    0    0    0 0.00  0.0    0  0.0 0.0000000   0.0
## 9     0    0    0    0 0.00  0.0    0  0.0 0.0000000   0.0
## 10    0    0    0    0 0.00  0.0    0  0.0 0.0000000   0.0
```

(a) Create a Function

```
adj_to_probTrans <- function(adjMat) {
  t(adjMat) %*% diag(1/colSums(t(adjMat)))
}

(T <- adj_to_probTrans(A)) %>% round(2)
```

```
##    [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## 1     0    1    0    0 0.25  0.0    0  0.2 0.00   0.0
## 2     1    0    0    0 0.25  0.5    1  0.2 0.33   0.0
## 3     0    0    0    1 0.25  0.5    0  0.0 0.33   0.5
## 4     0    0    1    0 0.25  0.0    0  0.0 0.33   0.5
## 5     0    0    0    0 0.00  0.0    0  0.2 0.00   0.0
## 6     0    0    0    0 0.00  0.0    0  0.2 0.00   0.0
## 7     0    0    0    0 0.00  0.0    0  0.2 0.00   0.0
## 8     0    0    0    0 0.00  0.0    0  0.0 0.00   0.0
## 9     0    0    0    0 0.00  0.0    0  0.0 0.00   0.0
## 10    0    0    0    0 0.00  0.0    0  0.0 0.00   0.0
```

Page Rank Random Surfer

The random surfer page rank method modifies the probability transition matrix T so that the method works also for non-ergodic graphs by introducing the possibility of a random jump, we'll call the surfer transition matrix S:

$$S = \lambda T + (1 - \lambda) B : \tag{1}$$

$$\tag{2}$$

$$B = \begin{bmatrix} \frac{1}{N} & \frac{1}{N} & \cdots & \frac{1}{N} \\ \frac{1}{N} & \frac{1}{N} & \cdots & \frac{1}{N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{N} & \frac{1}{N} & \cdots & \frac{1}{N} \end{bmatrix} \tag{3}$$

$$N = ||V|| \tag{4}$$

$$\lambda \in [0, 1] \tag{5}$$

```
B <- matrix(rep(1/nrow(T), length.out = nrow(T)**2), nrow = nrow(T))
l <- 0.8

S <- l*T+(1-l)*B
```

1. Eigen Value Method The eigenvector corresponding to the the eigenvalue of 1 will be the stationary point:

```
eigen(S, symmetric = FALSE)
```

```
## eigen() decomposition
## $values
##  [1]  1.000000e+00  8.000000e-01 -8.000000e-01 -8.000000e-01  3.117428e-09
##  [6] -3.117428e-09  1.233252e-17 -6.831762e-18  1.351981e-18  1.827902e-34
##
## $vectors
##              [,1]          [,2]          [,3]          [,4]          [,5]
##  [1,] -0.48531271  5.000000e-01  1.074642e-03  7.070996e-01  6.735753e-01
##  [2,] -0.52732002  5.000000e-01 -1.074642e-03 -7.070996e-01  9.622505e-02
##  [3,] -0.49152601 -5.000000e-01  7.071060e-01  3.197134e-03  9.622505e-02
##  [4,] -0.47977477 -5.000000e-01 -7.071060e-01 -3.197134e-03  2.886751e-01
##  [5,] -0.05288058  7.620569e-17  1.212133e-16 -3.419297e-18 -3.849002e-01
##  [6,] -0.05288058  7.620569e-17 -7.099634e-17 -3.419297e-18 -3.849002e-01
##  [7,] -0.05288058  7.620569e-17  2.943750e-17 -3.419297e-18 -3.849002e-01
##  [8,] -0.04558671  6.926804e-17 -1.948070e-18 -4.183296e-17 -7.499367e-09
##  [9,] -0.04558671  6.926804e-17  7.359376e-18 -4.183296e-17 -7.499367e-09
## [10,] -0.04558671  6.926804e-17  7.359376e-18 -4.183296e-17 -7.499367e-09
##                [,6]          [,7]          [,8]          [,9]         [,10]
##  [1,] -6.735753e-01 -2.658112e-01  5.357798e-01 -4.171123e-01  2.122431e-01
##  [2,] -9.622504e-02  1.896313e-01 -1.330801e-01 -7.334472e-02  1.241240e-01
##  [3,] -9.622504e-02  1.990137e-01 -1.665251e-01  2.591495e-01 -1.598069e-04
##  [4,] -2.886751e-01 -3.972519e-02  1.099005e-02 -1.886035e-02  6.690506e-02
##  [5,]  3.849002e-01 -7.585250e-01  5.323206e-01  2.933789e-01 -4.964959e-01
##  [6,]  3.849002e-01  4.774778e-01 -3.550303e-01  5.560197e-01 -1.341297e-01
##  [7,]  3.849002e-01  2.204565e-01 -5.047228e-01  1.987554e-01  1.561493e-01
##  [8,] -7.499367e-09 -1.208463e-16  1.668414e-16 -2.685569e-16  3.416862e-17
##  [9,] -7.499367e-09 -1.125898e-02  4.013399e-02 -3.989931e-01 -5.316106e-01
## [10,] -7.499367e-09 -1.125898e-02  4.013399e-02 -3.989931e-01  6.029746e-01
```

So in this case ~~the~~ a stationary point is $\langle -0.49, -0.53, -0.49, -0.48, -0.05, -0.05, -0.05, -0.04, -0.04, -0.04 \rangle$ which can be verified:

$$1\vec{p} = S\vec{p}$$

```
(p     <- eigen(S)$values[1] * eigen(S)$vectors[,1])
```

```
##  [1] -0.48531271 -0.52732002 -0.49152601 -0.47977477 -0.05288058 -0.05288058
##  [7] -0.05288058 -0.04558671 -0.04558671 -0.04558671
```

```
(p_new <- S %*% p)
```

```
##            [,1]
## 1  -0.48531271
## 2  -0.52732002
## 3  -0.49152601
## 4  -0.47977477
## 5  -0.05288058
## 6  -0.05288058
## 7  -0.05288058
## 8  -0.04558671
## 9  -0.04558671
## 10 -0.04558671
```

However this vector does not sum to 1 so the scale should be adjusted (for probabilities the vector should sum to 1):

```
(p_new <- p_new/sum(p_new))
```

```
##          [,1]
## 1  0.2129185
## 2  0.2313481
## 3  0.2156444
## 4  0.2104889
## 5  0.0232000
## 6  0.0232000
## 7  0.0232000
## 8  0.0200000
## 9  0.0200000
## 10 0.0200000
```

2. Power Value Method Using the power method should give the same result, which it indeed does, but for the scale:

```
p_new <- p_new *123456789

while (sum(round(p, 9) != round(p_new, 9))) {
    (p     <- p_new)
    (p_new <- S %*% p)
}

p_new
```

```
##         [,1]
## 1   26286237
## 2   28561500
## 3   26622771
## 4   25986282
## 5    2864198
## 6    2864198
## 7    2864198
## 8    2469136
## 9    2469136
## 10   2469136
```

```
p
```

```
##         [,1]
## 1   26286237
## 2   28561500
## 3   26622771
## 4   25986282
## 5    2864198
## 6    2864198
## 7    2864198
## 8    2469136
## 9    2469136
## 10   2469136
```

This answer is however identical in direction, if it scaled to 1 the same value will be returned:

```
(p_new <- p_new/sum(p_new))
```

```
##          [,1]
## 1   0.2129185
## 2   0.2313481
## 3   0.2156444
## 4   0.2104889
## 5   0.0232000
## 6   0.0232000
## 7   0.0232000
## 8   0.0200000
## 9   0.0200000
## 10  0.0200000
```

3. Scaling However if the initial state sums to 1, then the scale of the stationary vector will also sum to 1.

```
p     <- c(1, 0, 0, 0, 0, 0, 0, 0, 0, 0)
p_new <- S %*% p

while (sum(round(p, 9) != round(p_new, 9))) {
    (p     <- p_new)
    (p_new <- S %*% p)
}

cbind(p_new, p)
```

```
##          [,1]      [,2]
## 1  0.2129185 0.2129185
## 2  0.2313481 0.2313481
## 3  0.2156444 0.2156444
## 4  0.2104889 0.2104889
## 5  0.0232000 0.0232000
## 6  0.0232000 0.0232000
## 7  0.0232000 0.0232000
## 8  0.0200000 0.0200000
## 9  0.0200000 0.0200000
## 10 0.0200000 0.0200000
```

Implementing Page Rank on a much Larger Graph

Creating the Probability Transition Matrix

Implementing the page rank method on a larger graph requires the use of more efficient form of matrix storage.

An adjacency matrix (atleast in the context of graphs relating to webpages and social networks) will contain elements that are mostly zero because the number of edges leaving any vertex will tend to be significantly less than the total number of vertices.

A matrix exhibiting this property is known as a sparse matrix CITE

The properties of a sparse matrix can be implemented in order to improve performance, one such method to acheive this is *Compressed Sparse Row* (CSR) storage, which involves creating a seperate array of values and corresponding indices. CITE

This is implemented by the Matrix package in ***R***. CITE

An sparse matrix can be created using the following syntax, which will return a matrix of the class `dgCMatrix`:

```r
library(Matrix)
## Create Example Matrix
n <- 20
m <- 10^6
i <- sample(1:m, size = n); j <- sample(1:m, size = n); x <- rpois(n,
  ↪  lambda = 90)
A <- sparseMatrix(i, j, x = x, dims = c(m, m))

summary(A)
```

```
1000000 x 1000000 sparse Matrix of class "dgCMatrix", with 20 entries
        i      j   x
1  803589  66922 118
2   61426  83355  97
3  401058 103999  71
4  610432 206922  84
5  542888 217196  69
6  821769 291405  79
7  187782 364814  74
8  152229 451810 104
9  614645 462031  82
10 776459 566334  91
11 288279 630438  97
12 233553 631441  84
13 139900 649740  83
14 381442 681415  87
15 578270 755635  99
16 175521 775788  98
17  57981 809115  89
18 821120 809688 103
19 541818 976802  78
20 595348 993420  85
```

As before in section 3, the probability transition matrix can be found by:

1. Transposing the adjacency matrix, then
2. Scaling the columns to one

To implement this for a sparseMatrix of the class `dgCMatrix`, the same technique of multiplying by a diagonalised matrix may be implemented, however to create this new matrix, a new `sparseMatrix` will need to be created using the properties of the original matrix, this can be done like so:

```r
sparse_diag <- function(mat) {
 #' Diagonal Factors of Sparse Matrix
 #'
 #' Return a Diagonal Matrix of the 1 / colsum() such that
 #' matrix multiplication with this matrix would have all column sums
 #' sum to 1
 #'
 #' This should take the transpose of an adjacency matrix in and the
  ↪  output
 #' can be multiplied by the original matrix to scale it to 1.
 #' i

 ## Get the Dimensions
 n <- nrow(mat)

 ## Make a Diagonal Matrix of Column Sums
 D <- sparseMatrix(i = 1:n, j = 1:n, x = colSums(mat), dims = c(n,n))

 ## Throw away explicit Zeroes
 D <- drop0(D)

 ## Inverse the Values
 D@x <- 1/D@x

 ## Return the Diagonal Matrix
 return(D)
}
D <- sparse_diag(t(A))
summary(D)
```

```
1000000 x 1000000 sparse Matrix of class "dgCMatrix", with 20 entries
        i      j           x
1   57981  57981 0.011235955
2   61426  61426 0.010309278
3  139900 139900 0.012048193
4  152229 152229 0.009615385
5  175521 175521 0.010204082
6  187782 187782 0.013513514
7  233553 233553 0.011904762
8  288279 288279 0.010309278
9  381442 381442 0.011494253
10 401058 401058 0.014084507
11 541818 541818 0.012820513
12 542888 542888 0.014492754
13 578270 578270 0.010101010
14 595348 595348 0.011764706
```

```
15 610432 610432 0.011904762
16 614645 614645 0.012195122
17 776459 776459 0.010989011
18 803589 803589 0.008474576
19 821120 821120 0.009708738
20 821769 821769 0.012658228
```

and hence the probability transition matrix may be implemented by performing matrix multiplication accordingly:

```
summary(t(A) %*% D)
```

```
1000000 x 1000000 sparse Matrix of class "dgCMatrix", with 20 entries
        i      j x
1  809115  57981 1
2   83355  61426 1
3  649740 139900 1
4  451810 152229 1
5  775788 175521 1
6  364814 187782 1
7  631441 233553 1
8  630438 288279 1
9  681415 381442 1
10 103999 401058 1
11 976802 541818 1
12 217196 542888 1
13 755635 578270 1
14 993420 595348 1
15 206922 610432 1
16 462031 614645 1
17 566334 776459 1
18  66922 803589 1
19 809688 821120 1
20 291405 821769 1
```

Solving the Random Surfer via the Power Method

Solving the eigenvalues for such a large matrix will not feasible, instead the power method will need to be used to find the stationary point.

However, creating a matrix of background probabilites (denoted by `B` is section .2) will not be feasible, it would simply be too large, instead some algebra can be used to reduce B from a matrix into a vector containing only $\frac{1-\alpha}{N}$.

The power method is given by:

$$\vec{p} = \mathbf{S}\vec{p} \tag{6}$$

where:

$$S = \alpha \mathbf{T} + (1-\alpha)\,\mathbf{B} \tag{7}$$
$$\vec{p} = (\alpha \mathbf{T} + (1-\alpha)\,\mathbf{B})\,\vec{p} \tag{8}$$
$$= \alpha \mathbf{T}\vec{p} + (1-\alpha)\,\mathbf{B}\vec{p} \tag{9}$$

Let $\mathbf{F} = \mathbf{B}\vec{p}$, consider the value of $\mathbf{F}$:

$$\mathbf{F} = \begin{bmatrix} \frac{1}{N} & \frac{1}{N} & \cdots & \frac{1}{N} \\ \frac{1}{N} & \frac{1}{N} & \cdots & \frac{1}{N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{N} & \frac{1}{N} & \cdots & \frac{1}{N} \end{bmatrix} \begin{bmatrix} \vec{p_1} \\ \vec{p_2} \\ \vdots \\ \vec{p_m} \end{bmatrix} \tag{10}$$
$$= \begin{bmatrix} (\sum_{i=0}^{m} [p_i]) \times \frac{1}{N} \\ (\sum_{i=0}^{m} [p_i]) \times \frac{1}{N} \\ \vdots \\ (\sum_{i=0}^{m} [p_i]) \times \frac{1}{N} \end{bmatrix} \tag{11}$$
Probabilities sum to 1 and hence: (12)
$$= \begin{bmatrix} \frac{1}{N} \\ \frac{1}{N} \\ \frac{1}{N} \\ \vdots \\ \frac{1}{N} \end{bmatrix} \tag{13}$$

So instead the power method can be implemented by performing an algorithm to the effect of:

```
## Find Stationary point of random surfer
N     <- nrow(A)
alpha <- 0.8
F     <- rep((1-alpha)/N, nrow(A))  ## A nx1 vector of (1-alpha)/N

## Solve using the power method
p     <- rep(0, length.out = ncol(T)); p[1] <- 1
p_new <- alpha*T %*% p + F

## use a Counter to debug
i <- 0
while (sum(round(p, 9) != round(p_new, 9))) {
    p     <- p_new
    p_new <- alpha*T %*% p + F
    (i <- i+1) %>% print()
}

p %>% head() %>% print()
```