Data Sci Discover Project

Ryan Greenup

September 3, 2020

Contents

1 Implementing the Power Walk

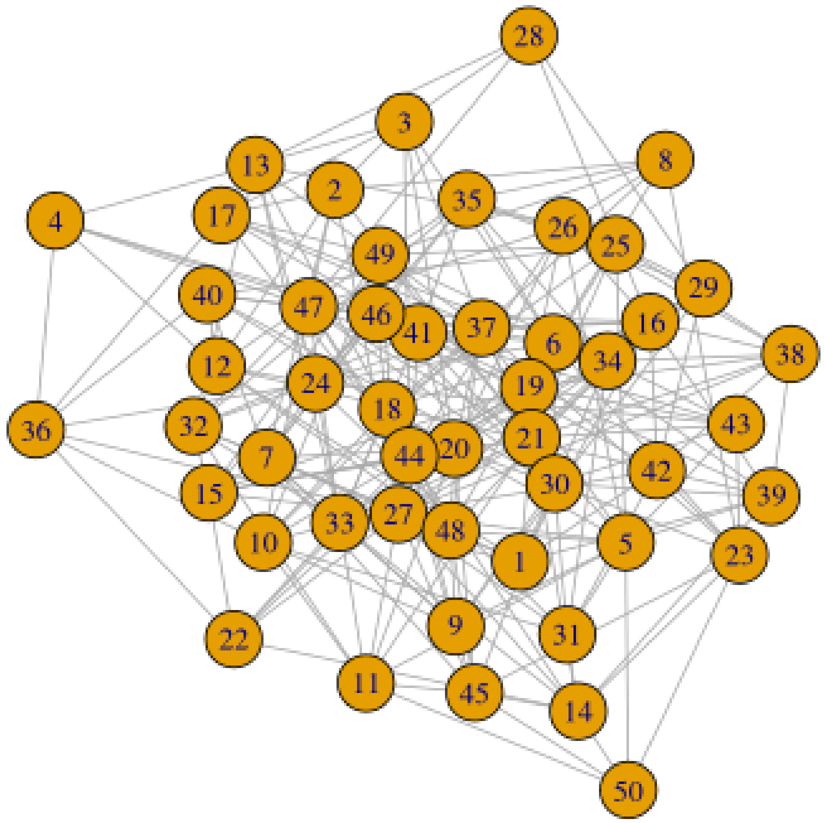
Load necessary packages etc:

```r
if (require("pacman")) {
    library(pacman)
  }else{
    install.packages("pacman")
    library(pacman)
  }
  pacman::p_load(Matrix, igraph, plotly, mise, docstring, expm)
```

TRUE
TRUE
TRUE
TRUE
TRUE
TRUE

Create an example Matrix:

```r
g1 <- igraph::erdos.renyi.game(n = 50, 0.2)
A <- igraph::get.adjacency(g1) # Row to column
plot(g1)
```



1.1 Using the Power Walk

1.1.1 Inspect the newly created matrix and create constants

```r
beta <- 0.843234
    <- beta
n    <- nrow(A)

str(A)
```

```
Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
  ..@ i       : int [1:504] 11 15 20 29 31 32 38 41 47 2 ...
  ..@ p       : int [1:51] 0 9 17 25 31 44 55 66 73 81 ...
  ..@ Dim     : int [1:2] 50 50
  ..@ Dimnames:List of 2
  .. ..$ : NULL
  .. ..$ : NULL
  ..@ x       : num [1:504] 1 1 1 1 1 1 1 1 1 1 ...
  ..@ factors : list()
```

1.1.2 Create a Diagonalised Scaling Matrix

```r
sparse_diag <- function(mat) {
  #' Diagonal Factors of Sparse Matrix
  #'
  #' Return a Diagonal Matrix containing either 1 / colsum() or 0 such that
  #' matrix multiplication with this matrix would have all columns
  #' 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
  # mat <- A
  ## 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)
}
```

1.1.3 Weight the Edges

Make the edges weighted with some real value

```r
weight_adjMat <- function(adjMat) {
  #' Weight Adjacency Matrix
  #'
  #' Randomly weights an adjacency matrix so that terms
  #' are Real (as opposed to natural) values.
  A@x*rnorm(length(A@x), 0, 0.1)
}
```

1.1.4 Create a Probability Transition Matrix

```r
adj_to_probTrans <- function(wadjmat, beta) {
  #' Adjacency to Probability Transition Matrix
  #'
  #' Returns a probability transition matrix from an input adjacency matrix
  #'
  #' Transposes an input matrix and then scales each column to sum to 1.
  #' Implemented with the Matrix dgCMatrix class in mind however also
  #' has logic to deal with a base matrix.
  #
  #' @param wadjmat A weighted adjacency matrix, ideally of the class dgCMatrix
  #' or atleast of the class matrix.
  #' @param beta The probability of following an edge

  wadjmat <- t(wadjmat) # transpose Assuming row->column (like igraph)
#  wadjmat <- A; beta <- 0.8

  if ("dgCMatrix" %in% class(wadjmat)) {

#    B    <- sparseMatrix(i = summary(wadjmat)$i, j = summary(wadjmat)$j, x =
    beta^wadjmat@x) # element wise exponentiation
#    Don't do this ^^ because it comes out with clipped off dimensions
    B    <- wadjmat
    B@x  <- beta^wadjmat@x # Element Wise exponentiation
    D_in <- sparse_diag(B)
    T = B %*% D_in
    return(T)

  } else if ("matrix" %in% class(wadjmat)) {
    print("WARNING: expected dgCMatrix but matrix detected")
    print("Attemptying to proceed anyway")
    for (i in ncol(wadjmat)) {
      #  wadjmat[, i] <- wadjmat[, i] / sum(wadjmat[, i])
    B    <- wadjmat
    B   <- beta^wadjmat # Element Wise exponentiation
    D_in <- sparse_diag(B)
    T = B %*% D_in
    return(as.matrix(T))
    }
    return(wadjmat)
  } else {
    print("ERROR: Require sparse wadjmatrix of class dgCWadjmatrix to")
  }
}

class(A)
(T <- adj_to_probTrans(A, beta = 0.843234)) %>% summary %>% head()
```

```
[1] "dgCMatrix"
attr(,"package")
[1] "Matrix"
50 x 50 sparse Matrix of class "dgCMatrix", with 504 entries
   i j          x
```

```
1 12 1 0.1111111
2 16 1 0.1111111
3 21 1 0.1111111
4 30 1 0.1111111
5 32 1 0.1111111
6 33 1 0.1111111
```

1.1.5 Implement the Power Method to find the Stationary Point

```r
## ** Power Method
p    <- rep(0, nrow(T))
p[1] <- 1
p_new   <- rep(0, nrow(T))
p_new[2]   <- 1

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



print(paste("The stationary point is"))
p %>% head()
```

```
[1] "The stationary point is"
6 x 1 Matrix of class "dgeMatrix"
           [,1]
[1,] 0.01785714
[2,] 0.01587302
[3,] 0.01587302
[4,] 0.01190476
[5,] 0.02579365
[6,] 0.02182540
```

1.2 Using Sparse Matrices

1.2.1 Theory

if I have:

- $\mathbf{O}_{i,j} := 0, \quad \forall i, j \leq n \in \mathbb{Z}^+$
- $\vec{p_i}$ as the state distribution, being a vector of length n

Then it can be shown (see (1)):

$$\mathbf{O}\mathbf{D}_{\mathbf{B}}^{-1}\vec{p_i} = \texttt{repeat}(\vec{p} \bullet \vec{\delta}^{\mathrm{T}}, \texttt{n})$$

where:

- $\vec{\delta_i} = \frac{1}{\texttt{colsums}(\mathbf{B})}$

This means we can do:

$$
\begin{aligned}
\vec{p_{i+1}} &= \mathbf{B}\mathbf{D}_{\mathbf{B}}^{-1}\vec{p_i} \\
&= (\mathbf{B} - \mathbf{O} + \mathbf{O})\,\mathbf{D}_{\mathbf{B}}^{-1}\vec{p_i} \\
&= \left((\mathbf{B} - \mathbf{O})\,\mathbf{D}_{\mathbf{B}}^{-1} + \mathbf{O}\mathbf{D}_{\mathbf{B}}^{-1}\right)\vec{p_i} \\
&= (\mathbf{B} - \mathbf{O})\,\mathbf{D}_{\mathbf{B}}^{-1}\vec{p_i} + \mathbf{O}\mathbf{D}_{\mathbf{B}}^{-1}\vec{p_i} \\
&= (\mathbf{B} - \mathbf{O})\,\mathbf{D}_{\mathbf{B}}^{-1}\vec{p_i} + \vec{\delta}'\vec{p_i}\vec{1}
\end{aligned}
$$

And so the the power method can be implemented using sparse matrices.

1. Solving the Background Probability Define $\vec{\delta}$ as the column sums of

$$
\begin{aligned}
\vec{\delta} &= \texttt{colsum}(\mathbf{B})^{-1} \\
&= \frac{1}{\vec{1^T}\boldsymbol{B}}
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{O}\mathbf{D}_{\mathbf{B}}^{-1}\overrightarrow{p_i} &= \begin{pmatrix} 1 & 1 & 1 & \\ 1 & 1 & 1 & \dots \\ 1 & 1 & 1 & \\ & \vdots & & \ddots \end{pmatrix} \begin{pmatrix} \delta_1 & 0 & 0 & \\ 0 & \delta_2 & 0 & \dots \\ 0 & 0 & \delta_3 & \\ 0 & \vdots & 0 & \ddots \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ \vdots \end{pmatrix} \\
&= \begin{pmatrix} \frac{p_1}{\delta 1} & + & \frac{p_2}{\delta_2} & + & \frac{p_3}{\delta_3} & \\ \frac{p_1}{\delta 1} & + & \frac{p_2}{\delta_2} & + & \frac{p_3}{\delta_3} & \dots \\ \frac{p_1}{\delta 1} & + & \frac{p_2}{\delta_2} & + & \frac{p_3}{\delta_3} & \\ & & \vdots & & & \ddots \end{pmatrix} \\
&= \begin{pmatrix} \sum_{i=1}^{n}[p_i\delta_i] \\ \sum_{i=1}^{n}[p_i\delta_i] \\ \sum_{i=1}^{n}[p_i\delta_i] \\ \vdots \end{pmatrix} \\
&= \begin{pmatrix} \overrightarrow{\delta^T} \bullet \vec{p} \\ \overrightarrow{\delta^T} \bullet \vec{p} \\ \overrightarrow{\delta^T} \bullet \vec{p} \\ \vdots \end{pmatrix} \\
&= \texttt{repeat}(\vec{p} \bullet \vec{\delta}^{\mathrm{T}}, \texttt{n})
\end{aligned} \tag{1}
$$

1.2.2 Set the value for B

```
B    <- A
B@x  <- ^(A@x) -1
```

1.2.3 Create the Scaling Matrix

The Transition probability matrix must sum to 1 so use the scaling matrix reduce the column sums.

```
B   <- 1/colSums(B)
Bt  <- t (B)
DB   <- diag (B)
```

1.2.4 Create the Trans Prob Mat

```
T <- B %*% DB
```

1.2.5 Implement the Power Walk

1. Set the Starting Values

```
p_new <- rep(1/n, n) # Uniform
p      <- rep(0, n)  # Zero
      <- 10^(-6)
```

2. Implement the loop

```
while (sum(abs(p_new - p)) >  ) {
(p <- as.vector(p_new)) # P should remain a vector
sum(p <- as.vector(p_new)) # P should remain a vector
    p_new <- T %*% p + rep (Bt %*% p, n)
}
```

```
Error in while (sum(abs(p_new - p))
) { :
  missing value where TRUE/FALSE needed
```

This error results because the $\vec{p_i} \to \infty$

3. Print the Stationary value

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

```
[1] -Inf -Inf -Inf -Inf -Inf -Inf
```