ST340 Lab 3: Markov chains & PageRank, K-means

2019–20

1: PageRank

Inspect and run the following code, which sets up each object we need in PageRank.

- `links[,1]` denotes "from" page
- `links[,2]` denotes "to" page

Note: there are no links from a page to itself. `A.sparse` is a sparse matrix with `A[i,j] = 1` whenever `links[k,]` $= (i, j)$ for some `k`.

```
library(Matrix) # Load the sparse Matrix library
```

```
## Warning: package 'Matrix' was built under R version 3.5.2
```

```
load("web-google.rdata") # Load the Google web data:
n <- max(links)
numlinks <- dim(links)[1]
# Turn the data into an adjacency matrix:
A.sparse <- sparseMatrix(i=links[,1], j=links[,2], x=rep(1,numlinks), dims=c(n,n))

outlinks <- rep(0,n)
for (i in 1:numlinks) {
  # Calculate outlinks where `outlinks[i]` is the number of outlinks for page `i`:
  outlinks[links[i,1]] <- outlinks[links[i,1]] + 1
}

d <- outlinks == 0 # Calculate `d`: the binary vector that is 1 when page `i` has no outlinks

vsH <- rep(0,numlinks)
for (k in 1:numlinks) {
  # Calculate the values to assign to `H`. These are nonzero whenever links[k,]` = (i,j)
  # for some k but are normalized so that sum(A[i,]) = 1 whenever sum(A[i,]) > 0
  # (i.e. apart from when i is a dangling node).
  vsH[k] <- 1/outlinks[links[k,1]]
}

# Construct H as a sparse matrix
H.sparse <- sparseMatrix(i=links[,1], j=links[,2], x=vsH, dims=c(n,n))

alpha <- .85
w <- rep(1/n,n) # (these are not sensible choices of w and p, but they "work")
p <- log(1+(1:n))/sum(log(1+(1:n)))
```

2: Power Iteration

(a) We will do power iteration for `m` iterations:

```
m <- 150
```

(b) We will compute the difference between probability vectors after each iteration:

```r
diffs <- rep(0,m)
```

(c) Start with the uniform distribution:

```r
muT <- t(rep(1/n,n))
for (i in 1:m) {
  muT.old <- muT # store the old value of muT

  # compute the new value of muT = muT.old%*%G

  ## write your code here


  # compute the difference
  diff <- sum(abs(muT-muT.old))
  diffs[i] <- diff
  print(paste("iteration ",i,": ||muT-muT.old||_1 = ",diff,sep=""))
}
```

(d) Rank the pages!

```r
ranking <- sort(muT, decreasing=TRUE, index.return=TRUE)$ix
```

3: K-means

(a) Write a k-means algorithm:

```r
my.kmeans <- function(xs,K,ZZZZzzzz=0) {
  # find the number of columns and rows of xs
  p <- dim(xs)[2]
  n <- dim(xs)[1]

  # initialize the clusters
  minx <- min(xs)
  maxx <- max(xs)
  cs <- xs[sample(n,K),]

  # zs will be modified in the code
  zs <- rep(0,n)

  converged <- FALSE
  while(!converged) {
    cs.old <- cs
    zs.old <- zs

    # update zs

    ## write your code here

    plot(xs,col=zs,pch=20)
    points(cs,pch=20,col="blue",cex=2)

    Sys.sleep(ZZZZzzzz)
```

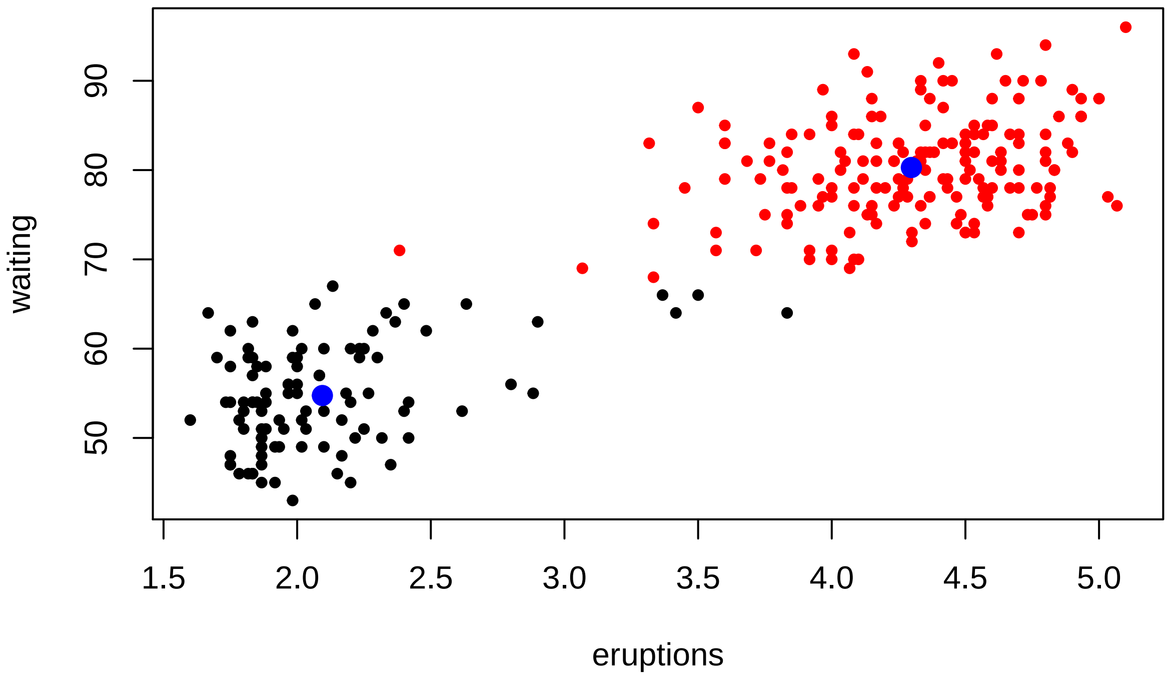

Figure 1: Built-in `kmeans` function for the Old Faithful dataset.

```
    # update cs

    ## write your code here

    plot(xs,col=zs,pch=20)
    points(cs,pch=20,col="blue",cex=2)

    Sys.sleep(ZZZZzzzz)

    if (all(zs==zs.old)) converged <- TRUE
  }
  return(list(cs=cs,zs=zs))
}
```

(b) Check that it works:

```
xs <- faithful
my.kmeans.out <- my.kmeans(xs, K=2, ZZZZzzzz=1)
```

(c) Check the output against the output of `kmeans(faithful,2)`:

```
par(mfrow=c(1,2))
plot(xs, col=my.kmeans.out$zs, pch=20)
points(my.kmeans.out$cs ,pch=20, col="blue", cex=2)

builtin.kmeans.out <- kmeans(xs,2)
plot(xs,col=builtin.kmeans.out$cluster,pch=20)
points(builtin.kmeans.out$centers,pch=20,col="blue",cex=2)
```