Professor Sanchez Notes - For Loops

Stat 133

Warren Birkman

Table of contents

1 Iterations (part 1)

Say we have some 2D data table in R, a matrix `x`

```
set.seed(123)

x = matrix(runif(30), 10,3)
x
```

```
           [,1]       [,2]      [,3]
 [1,] 0.2875775 0.95683335 0.8895393
 [2,] 0.7883051 0.45333416 0.6928034
 [3,] 0.4089769 0.67757064 0.6405068
 [4,] 0.8830174 0.57263340 0.9942698
 [5,] 0.9404673 0.10292468 0.6557058
 [6,] 0.0455565 0.89982497 0.7085305
 [7,] 0.5281055 0.24608773 0.5440660
 [8,] 0.8924190 0.04205953 0.5941420
 [9,] 0.5514350 0.32792072 0.2891597
[10,] 0.4566147 0.95450365 0.1471136
```

If I want to get statistics for each column like $\bar{x}_1, \bar{x}_2, \bar{x}_3$

```
xmeans = c(
  mean(x[,1]),
  mean(x[,2]),
  mean(x[,3])
)
xmeans
```

```
[1] 0.5782475 0.5233693 0.6155837
```

What if we have 1000 columns, this would be boring and take forever to type out.

> Common step: $mean(X[,pos])$
>
> pos - takes the forms $1, 2, 3$

1.1 For Loops

```
x = matrix(runif(30), 10, 3)

xmeans = rep(0,ncol(x)) # -> [0,0,0]

# The vector is the sequence that defines what values the loop will iterate through
for (iterator in vector) {
  xmeans[iterator] = mean(x[,iterator])
}
```

```
set.seed(123)

x = matrix(runif(30), 10, 3)

xmeans = rep(0, ncol(x))

for (pos in 1:ncol(x)) {
  xmeans[pos] = mean(x[,pos])
}

xmeans
```

```
[1] 0.5782475 0.5233693 0.6155837
```

in (pos in 1:ncol(x)), the sequence 1:ncol(x) must contain the values that pos takes through iterations, so in this case ncol(x) returns 3, and 1:3 = c(1,2,3). pos takes the values 1, 2, 3, and this allows the values of xmeans to be set as the mean(x[,pos]) for each value of pos.

- Use a for loop when you know the number of times that a computation takes place

```
set.seed(123)

# NULL
xmeans = NULL
for (pos in 1:ncol(x)) {
  xmeans = c(xmeans, mean(x[,pos]))
}

xmeans
```

```
[1] 0.5782475 0.5233693 0.6155837
```

When xmeans = NULL, each time we append an element to the vector, a new copy of the entire vector is made, which includes the newly added element. This process is inefficient compared to strictly defining the size with xmeans = rep(0, ncol(X)).

```
set.seed(123)

# c()
xmeans = c()
for (pos in 1:ncol(x)) {
  xmeans = c(xmeans, mean(x[,pos]))
}

xmeans
```

```
[1] 0.5782475 0.5233693 0.6155837
```

1.2 Coding

```
set.seed(123)

X = matrix(runif(30), 10,3)
X
```

```
           [,1]       [,2]      [,3]
 [1,] 0.2875775 0.95683335 0.8895393
 [2,] 0.7883051 0.45333416 0.6928034
 [3,] 0.4089769 0.67757064 0.6405068
 [4,] 0.8830174 0.57263340 0.9942698
 [5,] 0.9404673 0.10292468 0.6557058
 [6,] 0.0455565 0.89982497 0.7085305
 [7,] 0.5281055 0.24608773 0.5440660
 [8,] 0.8924190 0.04205953 0.5941420
 [9,] 0.5514350 0.32792072 0.2891597
[10,] 0.4566147 0.95450365 0.1471136
```

```r
# output vector
xmeans = rep(0, ncol(X))

# for loop
for (pos in 1:ncol(X)) {
  print(paste('iteration:', pos))
  xmeans[pos] = mean(X[,pos])
}
```

```
[1] "iteration: 1"
[1] "iteration: 2"
[1] "iteration: 3"
```

```r
xmeans
```

```
[1] 0.5782475 0.5233693 0.6155837
```

1.3 `apply()` family functions

- `apply()` : apply a function on an R **array** (e.g. 2-dim, N-dim)
    - `MARGIN` - `1` is rows, `2` is columns
    - `FUN` - function we want to apply
- `lapply()` : apply a function on an R **list**
- `sweep()` : sweep elements of a matrix - goes element by element and applies a `FUN`ction with a set `STAT`

Some useRs refer to these functions as **vectorized loop functions**

```
apply(X, MARGIN = 2, FUN = mean)
```

```
[1] 0.5782475 0.5233693 0.6155837
```

```
apply(X, MARGIN=2, FUN= median)
```

```
[1] 0.5397703 0.5129838 0.6481063
```

```
apply(X, MARGIN=2, FUN=min)
```

```
[1] 0.04555650 0.04205953 0.14711365
```

```
range(X[,1])
```

```
[1] 0.0455565 0.9404673
```

`range()` computes the minimum and maximum, doesn't subtract the way we want it to, so we create the range function

```
myrange = function(x) {
  max(x) - min(x)
}
```

```
# using myrange
apply(X, MARGIN=2, FUN = myrange)
```

```
[1] 0.8949108 0.9147738 0.8471561
```

We can also use "anonymous" function (i.e. lambda functions)

```
apply(X, 2, function(y){max(y)-min(y)})
```

```
[1] 0.8949108 0.9147738 0.8471561
```

```
apply(X, 2, \(y) max(y)-min(y)) # another way to write anonymous function
```

```
[1] 0.8949108 0.9147738 0.8471561
```

1.4 Applying a computation to every element of a matrix

```
X * 2 # vectorization
```

```
           [,1]       [,2]      [,3]
 [1,] 0.5751550 1.91366669 1.7790786
 [2,] 1.5766103 0.90666831 1.3856068
 [3,] 0.8179538 1.35514127 1.2810136
 [4,] 1.7660348 1.14526680 1.9885396
 [5,] 1.8809346 0.20584937 1.3114116
 [6,] 0.0911130 1.79964994 1.4170609
 [7,] 1.0562110 0.49217547 1.0881320
 [8,] 1.7848381 0.08411907 1.1882840
 [9,] 1.1028700 0.65584144 0.5783195
[10,] 0.9132295 1.90900730 0.2942273
```

Say we have a matrix `X`, and we select element x_{ij}, and we want the mean-center $(x_{ij} - \bar{x}_j)$

```
# assuming that we have the vector of means
xmeans
```

```
[1] 0.5782475 0.5233693 0.6155837
```

```
sweep(X, MARGIN=2, STATS = xmeans, FUN="-")
```

```
             [,1]        [,2]        [,3]
 [1,] -0.29066998  0.43346406  0.27395562
 [2,]  0.21005763 -0.07003513  0.07721971
 [3,] -0.16927058  0.15420135  0.02492311
 [4,]  0.30476990  0.04926412  0.37868608
 [5,]  0.36221978 -0.42044460  0.04012210
 [6,] -0.53269101  0.37645569  0.09294677
 [7,] -0.05014202 -0.27728155 -0.07151768
 [8,]  0.31417154 -0.48130975 -0.02144168
 [9,] -0.02681249 -0.19544856 -0.32642396
[10,] -0.12163277  0.43113437 -0.46847005
```

```
# STATS is the stat we are using
# FUN = "-" tells R to use the minus function
```

```
sweep(X, MARGIN=2, STATS=5, FUN="+") # adds 5 to all elements in the matrix
```

```
          [,1]     [,2]     [,3]
 [1,] 5.287578 5.956833 5.889539
 [2,] 5.788305 5.453334 5.692803
 [3,] 5.408977 5.677571 5.640507
 [4,] 5.883017 5.572633 5.994270
 [5,] 5.940467 5.102925 5.655706
 [6,] 5.045556 5.899825 5.708530
 [7,] 5.528105 5.246088 5.544066
 [8,] 5.892419 5.042060 5.594142
 [9,] 5.551435 5.327921 5.289160
[10,] 5.456615 5.954504 5.147114
```