lab02.Rmd

Burhan A Hanif

February 8, 2019

Basic R Skills

First, install the package `testthat` (a widely accepted testing suite for R) from https://github.com/r-lib/testthat using `pacman`. If you are using Windows, this will be a long install, but you have to go through it for some of the stuff we are doing in class. LINUX (or MAC) is preferred for coding. If you can't get it to work, install this package from CRAN (still using `pacman`), but this is not recommended long term.

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

```
## Loading required package: pacman
```

```
#pacman::p_load(devtools)
#install_github("r-lib/testthat")
pacman::p_load(testthat)
```

- Use the `seq` function to create vector `v` consisting of all numbers from -100 to 100.

```
v=seq(-100,100)
```

Test using the following code:

```
expect_equal(v, -100 : 100)
```

If there are any errors, the `expect_equal` function will tell you about them. If there are no errors, then it will be silent.

- Create a function `my_reverse` which takes as required input a vector and returns the vector in reverse where the first entry is the last entry, etc. No function calls are allowed inside your function (otherwise that would defeat the purpose of the ex ercise).

```
my_reverse_mine = function(vec) {
  #revvec = rep(0 , length(vec))
  revvec=NULL
  for(i in vec)
    revvec=c(i,revvec)
  revvec
}
my_reverse_mine(v)
```

```
##   [1]  100   99   98   97   96   95   94   93   92   91   90   89   88   87
##  [15]   86   85   84   83   82   81   80   79   78   77   76   75   74   73
##  [29]   72   71   70   69   68   67   66   65   64   63   62   61   60   59
##  [43]   58   57   56   55   54   53   52   51   50   49   48   47   46   45
##  [57]   44   43   42   41   40   39   38   37   36   35   34   33   32   31
##  [71]   30   29   28   27   26   25   24   23   22   21   20   19   18   17
##  [85]   16   15   14   13   12   11   10    9    8    7    6    5    4    3
##  [99]    2    1    0   -1   -2   -3   -4   -5   -6   -7   -8   -9  -10  -11
## [113]  -12  -13  -14  -15  -16  -17  -18  -19  -20  -21  -22  -23  -24  -25
## [127]  -26  -27  -28  -29  -30  -31  -32  -33  -34  -35  -36  -37  -38  -39
## [141]  -40  -41  -42  -43  -44  -45  -46  -47  -48  -49  -50  -51  -52  -53
## [155]  -54  -55  -56  -57  -58  -59  -60  -61  -62  -63  -64  -65  -66  -67
```

```
## [169]  -68  -69  -70  -71  -72  -73  -74  -75  -76  -77  -78  -79  -80  -81
## [183]  -82  -83  -84  -85  -86  -87  -88  -89  -90  -91  -92  -93  -94  -95
## [197]  -96  -97  -98  -99 -100
```

```
length(my_reverse_mine(v))
```

```
## [1] 201
```

Test using the following code:

```
expect_equal(my_reverse_mine(c("A", "B", "C")), c("C", "B", "A"))
expect_equal(my_reverse_mine(v), rev(v))
```

- Let `n = 50`. Create a nxn matrix `R` of exactly 50% entries 0's, 25% 1's 25% 2's in random locations.

```
n = 50
x = sample((c(rep(0,1250) , rep(1,625), rep(2,625)) ))
R = matrix(x,n,n)
```

Test using the following and write two more tests as specified below:

```
expect_equal(dim(R), c(n, n))
#TO-DO test that the only unique values are 0, 1, 2
expect_equal((sum(c(R)==0))+(sum(c(R)==1))+(sum(c(R)==2)), n*n)
#TO-DO test that5 there are exactly 625 2's
count = 0
for(i in x)
  if (i==2)
    count=count+1
count
```

```
## [1] 625
```

```
expect_equal(count,625)
```

- Randomly punch holes (i.e. `NA`) values in this matrix so that approximately 30% of the entries are missing.

```
for (i in seq( 0 , n*n))
  if(rbinom(1,1,.3)[1]==1){R[i]=NA}
```

Use the testthat library to test that this worked correctly by ensuring the number of missing entries is between the 0.5%ile and 99.5%ile of the appropriate binomial.

```
missing= sum(is.na(c(R)))
expect_lt(missing , qbinom(.995, n*n , .3))
expect_gt(missing, qbinom(.005 , n*n , .3))
```

- Sort the rows matrix `R` by the largest row sum to lowest. Be careful about the NA's!

```
row_value = NULL
for ( i in 1:n){
  row_value = c(row_value, sum(R[i, ], na.rm = TRUE))

}
 rownames(R) = row_value
 R = R[order(rownames(R), decreasing = TRUE), ]
```

Test using the following code.

```r
for (i in 2 : n){
  expect_gte(sum(R[i - 1, ], na.rm = TRUE), sum(R[i, ], na.rm = TRUE))
}
```

- We will now learn the `apply` function. This is a handy function that saves writing for loops which should be eschewed in R. Use the apply function to compute a vector whose entries are the standard deviation of each row. Use the apply function to compute a vector whose entries are the standard deviation of each column. Be careful about the NA's!

```r
# i dont understand without making my own function so i made my own function
fun=function(mat){sd(mat,na.rm=TRUE)}
apply(R , 1,FUN=fun)
```

```
##        38        35        34        34        31        31        30
## 0.8328828 0.7878977 0.8337397 0.8006408 0.8268689 0.9000467 0.8417256
##        30        29        29        29        29        29        29
## 0.8767946 0.8219673 0.8801300 0.8199686 0.8574929 0.8574929 0.9169737
##        28        28        28        28        27        27        27
## 0.8600506 0.8700513 0.8979695 0.8870032 0.9138467 0.8044546 0.8800624
##        27        27        27        26        26        26        25
## 0.8352988 0.8800624 0.8058923 0.8572330 0.8572330 0.7809280 0.9064064
##        25        25        24        24        24        24        24
## 0.8321901 0.8981065 0.8280787 0.7988405 0.7189813 0.8148421 0.8280787
##        24        24        24        23        23        23        23
## 0.7928250 0.8618916 0.8280787 0.7983117 0.7583370 0.8884337 0.8185052
##        23        22        21        21        19        18        16
## 0.7675458 0.7390782 0.8321497 0.7833495 0.8567322 0.7844645 0.7184212
##        15
## 0.7390660
```

- Use the `apply` function to compute a vector whose entries are the count of entries that are 1 or 2 in each column. Try to do this in one line.

```r
test=apply(R>=1,2, sum,na.rm=TRUE)
test
```

```
##  [1] 17 17 16 19 11 14 16 12 14 23 24 25 16 16 23 20 16 20 18 21 18 14 17
## [24] 12 16 17 17 13 19 21 20 16 14 19 21 16 20 26 15 18 18 17 12 16 20 22
## [47] 11 15 16 16
```

- Use the `split` function to create a list whose keys are the column number and values are the vector of the columns. Look at the last example in the documentation `?split`.

```r
list_1=split(R,col(R))
```

- In one statement, use the `lapply` function to create a list whose keys are the column number and values are themselves a list with keys: "min" whose value is the minimum of the column, "max" whose value is the maximum of the column, "pct_missing" is the proportion of missingness in the column and "first_NA" whose value is the row number of the first time the NA appears. Use the `which` function.

```r
list_2=lapply(list_1, function(R) {
  minum = min (R,na.rm = TRUE)
  pct_missing = sum(is.na(R))/ length(R)*100
  firstnona=min(which(is.na(R)))
  maxium=max(R,na.rm=TRUE)
  c(minum,maxium,pct_missing,firstnona)})
list_2
```

```
## $`1`
## [1]  0  2 20 16
##
## $`2`
## [1]  0  2 24  3
##
## $`3`
## [1]  0  2 28 12
##
## $`4`
## [1]  0  2 20  2
##
## $`5`
## [1]  0  2 38  4
##
## $`6`
## [1]  0  2 32  1
##
## $`7`
## [1]  0  2 38  2
##
## $`8`
## [1]  0  2 36  4
##
## $`9`
## [1]  0  2 38  1
##
## $`10`
## [1]  0  2 30  1
##
## $`11`
## [1]  0  2 30  3
##
## $`12`
## [1]  0  2 20  3
##
## $`13`
## [1]  0  2 38 12
##
## $`14`
## [1]  0  2 34  3
##
## $`15`
## [1]  0  2 28  4
##
## $`16`
## [1]  0  2 30  3
##
## $`17`
## [1]  0  2 46  8
##
## $`18`
## [1]  0  2 28  1
##
```

```
## $`19`
## [1]  0  2 24  5
##
## $`20`
## [1]  0  2 32  4
##
## $`21`
## [1]  0  2 26  4
##
## $`22`
## [1]  0  2 26  6
##
## $`23`
## [1]  0  2 40  1
##
## $`24`
## [1]  0  2 42  3
##
## $`25`
## [1]  0  2 40  5
##
## $`26`
## [1]  0  2 36  1
##
## $`27`
## [1]  0  2 24  2
##
## $`28`
## [1]  0  2 36  8
##
## $`29`
## [1]  0  2 28 12
##
## $`30`
## [1]  0  2 24  1
##
## $`31`
## [1]  0  2 24  2
##
## $`32`
## [1]  0  2 28  1
##
## $`33`
## [1]  0  2 28  4
##
## $`34`
## [1]  0  2 24  2
##
## $`35`
## [1]  0  2 24  1
##
## $`36`
## [1]  0  2 42  1
##
```

```
## $`37`
## [1]  0  2 28  2
##
## $`38`
## [1]  0  2 26  4
##
## $`39`
## [1]  0  2 44  2
##
## $`40`
## [1]  0  2 24  8
##
## $`41`
## [1]  0  2 26  5
##
## $`42`
## [1]  0  2 34  2
##
## $`43`
## [1]  0  2 28  4
##
## $`44`
## [1]  0  2 24  3
##
## $`45`
## [1]  0  2 28  6
##
## $`46`
## [1]  0  2 24  8
##
## $`47`
## [1]  0  2 44  1
##
## $`48`
## [1]  0  2 30  1
##
## $`49`
## [1]  0  2 34  1
##
## $`50`
## [1]  0  2 32  2
```

- Create a vector `v` consisting of a sample of 1,000 iid normal realizations with mean -10 and variance 10.

```
v=rnorm(1000 , mean=-10 ,sd=sqrt(10) )
```

- Find the average of `v` and the standard error of `v`.

```
sum(v)/length(v)
```

```
## [1] -9.954832
```

```
se=sd(v)/sqrt(length(v))
se
```

```
## [1] 0.1022851
```

- Find the 5%ile of `v` and use the `qnorm` function to compute what it theoretically should be.

```
t = quantile(v , probs = .05)
q = qnorm(.05 ,-10 ,sqrt(10))
expect_equal(as.numeric(t), as.numeric(q), tol = se)
```

- Create a list named `my_list` with keys "A", "B", … where the entries are arrays of size 1, 2 x 2, 3 x 3 x 3, etc. Fill the array with the numbers 1, 2, 3, etc. Make 8 entries.

```
letters = c("A", "B" , "C" , "D" , "E" , "F" , "G","H")
my_list = list()
for(i in 1:8){
  key = letters[i]
  my_list[[key]]= array(seq(1, i) , dim = c(rep(i , i)))
}
```

Test with the following uncomprehensive tests:

```
expect_equal(my_list$A[1], 1)
expect_equal(my_list[[2]][, 1], 1 : 2)
expect_equal(dim(my_list[["H"]]), rep(8, 8))
```

Run the following code:

```
lapply(my_list, object.size)
```

```
## $A
## 224 bytes
##
## $B
## 232 bytes
##
## $C
## 352 bytes
##
## $D
## 1248 bytes
##
## $E
## 12744 bytes
##
## $F
## 186864 bytes
##
## $G
## 3294416 bytes
##
## $H
## 67109104 bytes
```

Use `?object.size` to read about what these functions do. Then explain the output you see above. For the later arrays, does it make sense given the dimensions of the arrays?

Answer here in English. lapply will apply to every element in a given list, the second input in our case object.size. object.size gets how much memory how much each input takes up.

Now cleanup the namespace by deleting all stored objects and functions:

```
rm(list=ls())
```

Basic Binary Classification Modeling

- Load the famous `iris` data frame into the namespace. Provide a summary of the columns and write a few descriptive sentences about the distributions using the code below and in English.

```
data("iris")
summary(iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width   
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100  
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300  
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300  
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199  
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800  
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500  
##        Species  
##  setosa    :50  
##  versicolor:50  
##  virginica :50  
##                 
##                 
##                 
```

The outcome metric is `Species`. This is what we will be trying to predict. However, we have only done binary classification in class (i.e. two classes). Thus the first order of business is to drop one class. Let's drop the level "virginica" from the data frame.

```
dt = iris[iris$Species!= "virginica", ]
```

Now create a vector `y` that is length the number of remaining rows in the data frame whose entries are 0 if "setosa" and 1 if "versicolor".

```
n = nrow(dt)
y = NULL
counter = 1
for(i in dt$Species) {
  if (i == "setosa") {
    y = c(0,y)
  }
  else{
    y=c(y,1)
  }
}
```

- Fit a threshold model to `y` using the feature `Sepal.Length`. Try to write your own code to do this. What is the estimated value of the threshold parameter? What is the total number of errors this model makes?

```
MAXiter = 1000
w = 0
X=as.matrix(dt$Sepal.Length)
for (iter in 1 : MAXiter){
  for( i in 1 : nrow(X)){
    x_i = X[i, ]
```

```
    yhat_i =ifelse(sum(x_i * w)> 0 , 1, 0)
    y_i = y[i]
    w = w + (y_i - yhat_i) * x_i
  }
}
w
```

```
## [1] 3.9
```

Does this make sense given the following summaries:

```
summary(iris[iris$Species == "setosa", "Sepal.Length"])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   4.300   4.800   5.000   5.006   5.200   5.800
```

```
summary(iris[iris$Species == "virginica", "Sepal.Length"])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   4.900   6.225   6.500   6.588   6.900   7.900
```

Write your answer here in English.

Our model is not the best since the two species are quite similiar and the sepal length is not a good feature to predict the species of the flower.

- What is the total number of errors this model makes (in-sample)?

```
yhat = ifelse(X %*% w>0,1,0)
  error=sum(y != yhat)/length(y)
  error
```

```
## [1] 0.5
```