HW01p

Zeke Luger

February 18, 2018

Welcome to HW01p where the "p" stands for "practice" meaning you will use R to solve practical problems. This homework is due 11:59 PM Satuday 2/24/18.

You should have RStudio installed to edit this file. You will write code in places marked "TO-DO" to complete the problems. Some of this will be a pure programming assignment. The tools for the solutions to these problems can be found in the class practice lectures. I want you to use the methods I taught you, not for you to google and come up with whatever works. You won't learn that way.

To "hand in" the homework, you should compile or publish this file into a PDF that includes output of your code. Once it's done, push by the deadline.

R Basics

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(testthat)
```

1. 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.

2. 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 exercise).

```
my_reverse = function(x) {

    y=x
    lengthx = 0

    for(j in x) lengthx = lengthx + 1
    for(i in 1:lengthx) y[i] = x[lengthx-i+1]

    y

}
```

Test using the following code:

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

3. 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
z = c( rep(0, 1250), rep(1, 625), rep(2, 625) )
zrandom = sample(z)
R= matrix(zrandom, 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
all(R %in% c(0,1,2))
```

```
## [1] TRUE
```

```
#TO-DO test that there are exactly 625 2's
table(R)[3] == 625
```

```
##    2
## TRUE
```

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

```
holes = sample(1:2500, 750)
R[holes] = NA
```

Test using the following code. Note this test may fail 1/100 times.

```
num_missing_in_R = sum(is.na(c(R)))
expect_lt(num_missing_in_R, qbinom(0.995, n^2, 0.3))
expect_gt(num_missing_in_R, qbinom(0.005, n^2, 0.3))
```

5. Sort the rows matrix `R` by the largest row sum to lowest. See 2/3 way through practice lecture 3 for a hint.

```
rowsum = rowSums(R, na.rm = TRUE)
rowsum
```

```
##  [1] 37 18 24 25 28 17 26 30 21 30 23 18 16 31 22 37 24 31 28 21 30 18 25
## [24] 32 17 30 23 29 20 25 20 28 20 27 32 33 23 26 31 27 24 36 18 26 32 25
## [47] 26 20 37 26
```

```
roworder = order(rowsum, decreasing = TRUE)
roworder
```

```
##  [1]  1 16 49 42 36 24 35 45 14 18 39  8 10 21 26 28  5 19 32 34 40  7 38
## [24] 44 47 50  4 23 30 46  3 17 41 11 27 37 15  9 20 29 31 33 48  2 12 22
## [47] 43  6 25 13
```

```
R = R[roworder,]
```

Test using the following code.

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

```
}
```

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

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

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

```
mean(v)
```

```
## [1] -9.945599
```

```
sd(v)
```

```
## [1] 3.15379
```

Find the 5%ile of `v` and use the `qnorm` function as part of a test to ensure it is correct based on probability theory.

```
v_quantile = as.numeric(quantile(v, .05))
v_quantile
```

```
## [1] -14.935
```

```
norm_quantile = qnorm(.05, -10, sqrt(10))
norm_quantile
```

```
## [1] -15.20148
```

```
expect_equal(v_quantile, norm_quantile, tolerance = 1.0)
```

Find the sample quantile corresponding to the value -7000 of `v` and use the `pnorm` function as part of a test to ensure it is correct based on probability theory.

```
invq = ecdf(v)
v_cdf = invq(-7000)
v_cdf
```

```
## [1] 0
```

```
norm_cdf = pnorm(-7000, -10, sqrt(10))
norm_cdf
```

```
## [1] 0
```

```
expect_equal(v_cdf, norm_cdf, tol = 1)
```

7. 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.

```
my_list = list()
my_list$A = 1
my_list$B = array(1:2^2, c(2,2))
my_list$C = array(1:3^3, c(3,3,3))
my_list$D = array(1:4^4, c(4,4,4,4))
my_list$E = array(1:5^5, c(5,5,5,5,5))
my_list$F = array(1:6^6, c(6,6,6,6,6,6))
my_list$G = array(1:7^7, c(7,7,7,7,7,7,7))
my_list$H = array(1:8^8, c(8,8,8,8,8,8,8,8))
```

Test with the following uncomprehensive tests:

```
expect_equal(my_list$A, 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
## 48 bytes
##
## $B
## 216 bytes
##
## $C
## 336 bytes
##
## $D
## 1232 bytes
##
## $E
## 12728 bytes
##
## $F
## 186848 bytes
##
## $G
## 3294400 bytes
##
## $H
## 67109088 bytes
```

```
?lapply
?object.size
```

Use `?lapply` and `?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.

The lapply() applies a function, specified in its second argument to each element in the list or vector that is its first argument. Thus, we applied the object.size() function to each element in my_list. The object.size() function determines the amount of memory used to store a specified R object. As you can see, the size of each element in my_list is approximately propportional to the length of the vector that fills the array.

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

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

Basic Binary Classification Modeling

8. 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
##
##
##
```

This dataset was collected by Edgar Anderson in 1935, and analyzed by Ronald Fisher in an study for the journal Annals of Eugenics (yikes). It measures the petal and sepal length and width of 3 species of the genus Iris, and also includes their species. For those who don't know, a flower's sepals are the (often) leafy green bits below the petals. The dataset is included in R's built-in datasets, and is a commonly used practice set.

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.

```
novirginica = which(iris$Species != "virginica")
iris2 = iris[novirginica,]
```

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".

```
y = as.integer(factor(iris2$Species)) - 1
```

9. 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?

```
yhat = integer(length(y))
num_errors = integer(length(y))

for(i in 1:length(y)) {

    #set the sepal length threshold
    threshold = iris2$Sepal.Length[i]

    #make a vector of the expected values based on the threshold
```

```
    for(j in 1:length(y)) {

        #if (iris2$Sepal.Length[j] > threshold) yhat[j] = 1
        #else yhat[j] = 0
        yhat[j] = as.integer(iris2$Sepal.Length[j] > threshold)

    }

    num_errors[i] = sum(abs( yhat - y ))

}

best_indices = which(num_errors == min(num_errors))
best_threshold = iris2$Sepal.Length[best_indices][1]

answers = list()
#return the threshold with the minimum absolute error
answers$best_threshold = best_threshold
#also return the best models total error
answers$min_num_errors = min(num_errors)

answers
```

```
## $best_threshold
## [1] 5.4
##
## $min_num_errors
## [1] 11
```

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.

The threshold makes sense because the third quartile for setosa is 5.2, and the first quartile for virginica is 6.225. This means that if my algorithm chose a threshold between 5.2 and 6.225, it would get a maximum of 25 errors. That is as long as it also chooses virginica for Sepal.Length values greater than the threshold and setosa for values less than threshold.

10. Fit a perceptron model explaining `y` using all three features. Try to write your own code to do this. Provide the estimated parameters (i.e. the four entries of the weight vector)? What is the total number

of errors this model makes?

```
#Initializing
yhat = integer(length(y))

w = c(0,0,0,0,0)

x = matrix(NA, length(y), 5)
     x[,1] = rep(1, length(y))
     x[,2] = iris2$ Sepal.Length
     x[,3] = iris2$ Sepal.Width
     x[,4] = iris2$ Petal.Length
     x[,5] = iris2$ Petal.Width

total_errors = 100

iterations = 0

weights_by_iteration = matrix(NA, 400, 5)

#Algorithm
while(total_errors >= 1) {

     for (i in 1:length(y)) {

          yhat[i] = as.integer( w %*% x[i,] > 0 )

          w = w + (y[i] - yhat[i]) * x[i,]

          total_errors= sum(abs(y-yhat))

          iterations = iterations +1

          weights_by_iteration[iterations,] = w

     }
}


#Side-by-side results
ymatrix = matrix(NA, length(y), 2)
     ymatrix[,1] = y
     ymatrix[,2] = yhat
#ymatrix

#Weights by iteration
#weights_by_iteration[1:200,]
#weights_by_iteration[201:400,]

#Answers
answers = list(
     "total errors = " = sum(abs(y-yhat)),
     "iterations = " = iterations,
     "augment weight = " = w[1],
```

```
    "Sepal.Length weight = " = w[2],
    "Sepal.Width weight = " = w[3],
    "Petal.Length weight = " = w[4],
    "Petal.Width weight = " = w[5]
)
answers
```

```
## $`total errors = `
## [1] 0
##
## $`iterations = `
## [1] 400
##
## $`augment weight = `
## [1] -1
##
## $`Sepal.Length weight = `
## [1] -1.1
##
## $`Sepal.Width weight = `
## [1] -3.6
##
## $`Petal.Length weight = `
## [1] 5.2
##
## $`Petal.Width weight = `
## [1] 2.2
```