HW01p

Darshan Patel

February 22, 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 Saturday 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.

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

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

```r
pacman::p_load(testthat)
```

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

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

Test using the following code:

```r
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).

```r
my_reverse = function(vec){
  newvec = c()
  counter = 1
  newvec
  for(i in length(vec):1){
    newvec[counter] = vec[i]
    counter = counter + 1
  }
  return(newvec)
}
```

Test using the following code:

```r
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 $n \times n$ matrix `R` of exactly 50% entries 0's, 25% 1's 25% 2's in random locations.

```r
n = 50
values = c(rep(0,0.5*n*n), rep(1,0.25*n*n), rep(2,0.25*n*n))
R = matrix(sample(values), n, n)
```

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

```r
expect_equal(dim(R), c(n, n))

#Test 1
test_one = function(m){
  for(r in 1:nrow(m)){
    for(c in 1:ncol(m)){
      if(m[r,c] != 0 && m[r,c] != 1 && m[r,c] != 2){
        return("Your parameter space has been disturbed by values that are not 0, 1, 2.")
      }
    }
  }
  return("All values are within the parameter space of 0, 1, 2.")
}
test_one(R)
```

```
## [1] "All values are within the parameter space of 0, 1, 2."
```

```r
#Test 2
test_two = function(m){
  twos = 0
  for(r in 1:nrow(m)){
    for(c in 1:ncol(m)){
      if(m[r,c] == 2){
        twos = twos + 1
      }
    }
  }
  if(twos != 625){
    return("Your sampling method needs revision. There are not exactly 625 2's.")
  }
  else{
    return("There are exactly 625 2's.")
  }
}
test_two(R)
```

```
## [1] "There are exactly 625 2's."
```

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

```r
R[sample(x = n*n, size = n*n*0.3)] = NA
```

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

```r
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.

```
r_new = c()
for(i in 1:n){
  r_new = c(r_new, sum(R[i,], na.rm = TRUE))
}
row.names(R) = r_new
R = R[order(rownames(R), decreasing = TRUE),]
```

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, mean = -10, sd = sqrt(10))
```

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

```
avg = mean(v)
se = sd(v)/length(v)
```

The average of `v` is:

```
avg
```

```
## [1] -10.19591
```

while the standard error of `v` is:

```
se
```

```
## [1] 0.003195031
```

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.

```
q_1 = quantile(v,0.05)
```

The 5%ile of `v` is:

```
q_1
```

```
##        5%
## -15.38966
```

Is it correct?

```
expect_equal(as.numeric(q_1),qnorm(.05,mean = -10, sd = sqrt(10)), tol = 0.1)
```

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.

```
inverse_quantile_obj = ecdf(v)
sample_q = inverse_quantile_obj(-7000)
```

The sample quantile corresponding to the value -7000 of `v` is:

```
sample_q
```

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

Is it correct?

```
expect_equal(sample_q,pnorm(-7000, mean = -10, sd = sqrt(10)))
```

7. Create a list named `my_list` with keys "A", "B", ... where the entries are arrays of size 1, 2×2, $3 \times 3 \times 3$, etc. Fill the array with the numbers 1, 2, 3, etc. Make 8 entries.

```
my_list = list()
keys = c("A", "B", "C", "D", "E", "F", "G", "H")
for(i in 1:length(keys)){
  my_list[[keys[i]]] = array(data = seq(1:i), dim = 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
## 208 bytes
##
## $B
## 216 bytes
##
## $C
## 336 bytes
##
## $D
## 1232 bytes
##
## $E
## 12728 bytes
##
## $F
## 186848 bytes
##
## $G
## 3294400 bytes
##
## $H
## 67109088 bytes
```

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

Answer: The 'lapply' function applies a certain function to each of the elements in the first parameter. Here the parameter was the 'my_list' object that I created. The second parameter it took was the 'object.size' function which returns the amount of memory used to store that object. Together, the output shows how much memory is being allocated for storage of each element in the list given. For the latter arrays, the output does make sense given the dimensions are increasing exponentially.

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.

```
?iris
table = iris
summary(table)
```

```
##   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  
##                 
##                 
##                 
```

Answer: The columns of the `iris` data frame are `Sepal.Length`, `Sepal.Width`, `Petal.Length`, `Petal.Width` and `Species`. The sepal lengths vary from 4.3 cm to 7.9 cm with a median of 1.300 cm and average of 5.843 cm. The petal widths vary from 0.1 cm to 2.5 cm with a median of 1.3 cm and average of 1.199 cm. The data is taken from 3 types of species: setosa, versicolor, and virginica. In addition, 50 flowers of each species were selected for this data 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.

```
table = table[table$Species != "virginica",]
```

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

```
y = nrow(table)
for(i in 1:nrow(table)){
  if(table$Species[i] == "versicolor"){
    y[i] = 1
  }
  else{
    y[i] = 0
  }
}
```

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?

```
x = as.matrix(cbind(table[,1,drop = FALSE]))
max_iter = 20
w_vec = 0

for(i in 1:max_iter){
  for(j in 1:nrow(x)){
    x_i = x[j]
    y_hat_i = ifelse(sum(x_i * w_vec) > 0, 1, 0)
    w_vec = w_vec + (y[j] - y_hat_i) * x_i
  }
}
```

The estimated value of the threshold parameter is:

```
w_vec
```

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

while the error rate of the model is:

```
yhat = ifelse(x %*% w_vec > 0, 1, 0)
sum(y != yhat) / length(y)
```

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

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.

Answer: The threshold model produced a sensible result that is backed up by the statistical summary. It is noted that the range of 4.9 cm and 5.8 cm will have overlapping data from both species. Thus the lengths will be in close proximity with each other in this narrow range despite being of one of the two species. Therefore the dataset will not be `linearly separable.`

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?

```
x_2 = as.matrix(cbind(y, table[,1,drop = FALSE],
                      table[, 2, drop = FALSE],
                      table[,3, drop = FALSE],
                      table[,4, drop = FALSE]))
max_iter = 100
w_vec = rep(0,5)

for(i in 1:max_iter){
  for(j in 1:nrow(x_2)){
    x_i = x_2[j,]
    y_hat_i = ifelse(sum(x_i * w_vec) > 0, 1, 0)
    w_vec = w_vec + (y[j] - y_hat_i) * x_i
  }
}
```

The estimated parameters are:

```
w_vec
```

```
##            y Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##          2.0         -1.1         -3.6          5.2          2.2
```

while the error rate of this model is:

```
yhat = ifelse(x_2 %*% w_vec > 0, 1, 0)
sum(y != yhat)/length(y)
```

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

0.0 error? Sounds like overfitting to me..