HW01p

Angel Montero

February 24, 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. p

```
#TO-DO
library('testthat')
```

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

```
#TO-DO
v <- seq(201)-101
```

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).

```
#TO-DO
my_reverse = function (v) {
  v[length(v):1]
}
```

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
sample_space = c(0,1,2)
dist = c(0.5, 0.25, 0.25)
```

```
random_matrix = rep(sample_space, (n^2)*dist)

shuffled_random_matrix = sample(random_matrix)
shuffled_random_matrix
table(shuffled_random_matrix)

R = matrix(shuffled_random_matrix, nrow = n, ncol = n)
R
#TO-DO
```

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
unique.values = function(mat) {
  as.numeric( dimnames(table(R))[[1]])
}
expect_equal(sort(unique.values(R)), sort(c(2,1,0)))

#TO-DO test that there are exactly 625 2's
expect_equal(table(R)["2"][[1]], 625)
```

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

```
# let p (success) the probability that a location in the matrix will have a hole
holes = rbinom(length(R), size = 1, prob = 0.3)
R[(holes==1)]=NA

#TO-DO
```

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.

```
#TO-DO
R_row_sums = rowSums(R,na.rm=TRUE)
ordered_sums = order(R_row_sums)
reversed = my_reverse(ordered_sums)

R = R[reversed,]
```

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))
#TO-DO
```

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

```
# STD error function
se = function(x) {
  sd(x)/sqrt(length(x))
}
mean(v) #mean
se(v)
```

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.

```
#TO-DO
#expect_equal(..., tol = )

sample_quant = qnorm(0.05,mean(v),sd(v)) #  The 5%ile of the sample

percent_below_quant = length ( v[ (v < sample_quant)] ) / length(v) # the calculated % of the observati

expect_equal(percent_below_quant, 0.05, tolerance=0.005)
```

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.

```
#TO-DO
#expect_equal(..., tol = )

inverse_quantile_obj = ecdf(v)

inv=inverse_quantile_obj(-7000)
```

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.

```
#TO-DO

keys = c("A","B","C","D","E","F","G","H")

expect_equal(length(keys), 8) #sanity check

matrixify = function(n) {
  x = c(1:n^n)
  dim(x) = rep(n,n)
  x
}

my_list = lapply(1:8, matrixify)
names(my_list) = keys
```

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)
```

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.

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.

```
summary(iris)
```

The iris dataset has four attributes and 1 outcome column. The attributes measure the width and length of the sepal and petals, which are parts of the flower. The outcome is the species of the flower. There are three species of flowers: *iris setosa*, *versicolor,* and *virginica*. The length of the sepals ranges from 4.3 to 7.9 cm and the sepal widths from 2.0 to 4.4 cm.

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.

```
#TO-DO
iris = 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".

```
#TO-DO
y = as.numeric(iris$Species == "versicolor")
```

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?

```
#TO-DO

# We can look at the sepal length distribution of setosa, and versicolor. And see if they are far apart

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

mean(sepal)
sd(sepal)

#one way would be to pick as treshold the sepal length under which 95% of the setosas fall
treshold = qnorm(0.95, mean(sepal),sd(sepal))

                     # what we call                          # what it actually is
num_errors = sum( (iris$Sepal.Length < treshold) != (iris$Species == "setosa")  )

# our error rate
error_rate = num_errors/length(iris$Species)
```

Does this make sense given the following summaries:

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

I think it does make sense, if we plot the distribution of the sepal length by species, the mean and distributions of sepal length for setosa and versicolor are separated. We can take advantage of that separation by calling anything below a given sepal length "setosa" and anything above versicolor.

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?

```
#TO-DO
```

```
#attrs1 = cbind(1,iris[c(1,2,3,4)])
attrs1 = as.matrix(cbind(1, iris[, c(1,2,3,4), drop = FALSE]))


y_binary = as.numeric(iris$Species == "setosa")

#y_binary = ifelse(y_binary == 1, 0, 1)

MAX_ITER = 1000
w_vec = rep(0, 5) #intialize a 2-dim vector

for (iter in 1 : MAX_ITER){
  for (i in 1 : nrow(attrs1)){
    x_i = attrs1[i, ]
    yhat_i = ifelse(sum(x_i * w_vec) > 0, 1, 0)
    y_i = y_binary[i]
    w_vec = w_vec + (y_i - yhat_i) * x_i
  }
}
w_vec
```

What is our error rate?

```
yhat = ifelse(attrs1 %*% w_vec > 0, 1, 0)

sum(y_binary != yhat) / length(y_binary)
```

The error rate is 0. This is probably not a good sign, because it means we are overfitting the data.