HW01p

Michelle Chung

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.

```
install.packages("pacman", repos="http://cran.us.r-project.org")
```

```
##
## The downloaded binary packages are in
##  /var/folders/90/y2nf13b91_z8r2vdy16fl_vc0000gn/T//RtmpB0xiIR/downloaded_packages
```

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

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

```
# seq in lec01
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(v){
  v[length(v):1]
}
```

Test using the following code:

```
pacman::p_load(testthat)
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
numbers = c(0, 1, 2)
times = c(1250, 625, 625)
data <-rep(numbers, times)
data = sample(data)
R = matrix(data, nrow=n, ncol=n)
```

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

```
expect_equal(dim(R), c(n, n))
# test that the only unique values are 0, 1, 2
unique_values = c(unique(c(R)))
expect_setequal(unique_values, c(0, 1, 2))
# test that there are exactly 625 2's
count <- length(which(c(R) == 2))
expect_equal(625, count)
```

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

```
null_vector <- rep(NA,length(R))
?matrix
R <- matrix(sample(c(data,null_vector),
            replace = TRUE,
            size = 2500,
            prob = c(rep(0.7,length(data)),rep(0.3,length(data)))),
            nrow = n,
            ncol = n)
```

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.

```
R <- R[order(-rowSums(R, na.rm=TRUE)), ]
```

Test using the following code.

```
pacman::p_load(testthat)
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.

```
?rnorm
?rep
v <- c(rnorm(1000),mean=-10,variance=sqrt(10))
```

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

```
mean_of_v <- mean(v)
se_of_v <- (sd(v)/sqrt(1000))
```

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.

```
#TODO
fifth <- quantile(v, c(0.05))
quantiles_of_v <- quantile(v)
sd_of_v <- sd(v)
```

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.

```
#TODO
#expect_equal(..., tol = )
```

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.

```
?matrix
?list
rm
```

```
## function (..., list = character(), pos = -1, envir = as.environment(pos),
##     inherits = FALSE)
## {
##     dots <- match.call(expand.dots = FALSE)$...
##     if (length(dots) && !all(vapply(dots, function(x) is.symbol(x) ||
##         is.character(x), NA, USE.NAMES = FALSE)))
##         stop("... must contain names or character strings")
##     names <- vapply(dots, as.character, "")
##     if (length(names) == 0L)
##         names <- character()
##     list <- .Primitive("c")(list, names)
##     .Internal(remove(list, envir, inherits))
## }
## <bytecode: 0x7fa7c1c4eb48>
## <environment: namespace:base>
```

```
char_vector = c("A", "B", "C", "D", "E", "F", "G", "H")
my_list = list()
# TODO Figure out why this syntax doesn't work.
for(i in 1:8){
  my_list$char_vector[i] = matrix(data = c(rep(i,i*i)), nrow = i, ncol = i)
}
```

```
## Warning in my_list$char_vector[i] = matrix(data = c(rep(i, i * i)), nrow =
## i, : number of items to replace is not a multiple of replacement length

## Warning in my_list$char_vector[i] = matrix(data = c(rep(i, i * i)), nrow =
## i, : number of items to replace is not a multiple of replacement length

## Warning in my_list$char_vector[i] = matrix(data = c(rep(i, i * i)), nrow =
## i, : number of items to replace is not a multiple of replacement length

## Warning in my_list$char_vector[i] = matrix(data = c(rep(i, i * i)), nrow =
## i, : number of items to replace is not a multiple of replacement length

## Warning in my_list$char_vector[i] = matrix(data = c(rep(i, i * i)), nrow =
```

```
## i, : number of items to replace is not a multiple of replacement length

## Warning in my_list$char_vector[i] = matrix(data = c(rep(i, i * i)), nrow =
## i, : number of items to replace is not a multiple of replacement length

## Warning in my_list$char_vector[i] = matrix(data = c(rep(i, i * i)), nrow =
## i, : number of items to replace is not a multiple of replacement length
```

Test with the following uncomprehensive tests:

```
pacman::p_load(testthat)
#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)
```

```
## $char_vector
## 72 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.

`?lapply` returns an ordered list representing the range, y, of domain x. Length(y) is equal to length(x) and the elements y1...yn correspond to the elements x1...xn. `?object.size` returns an integer representing the size of the memory being allocated to store the R object.

Considering the exponential increase in the size of the matrices produced for my_list, the corresponding exponential increase in object.size makes sense.

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

```
rm
```

```
## function (..., list = character(), pos = -1, envir = as.environment(pos),
##     inherits = FALSE)
## {
##     dots <- match.call(expand.dots = FALSE)$...
##     if (length(dots) && !all(vapply(dots, function(x) is.symbol(x) ||
##         is.character(x), NA, USE.NAMES = FALSE)))
##         stop("... must contain names or character strings")
##     names <- vapply(dots, as.character, "")
##     if (length(names) == 0L)
##         names <- character()
##     list <- .Primitive("c")(list, names)
##     .Internal(remove(list, envir, inherits))
## }
## <bytecode: 0x7fa7c1c4eb48>
## <environment: namespace:base>
```

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.

```
rm
```

```
## function (..., list = character(), pos = -1, envir = as.environment(pos),
##     inherits = FALSE)
## {
##     dots <- match.call(expand.dots = FALSE)$...
##     if (length(dots) && !all(vapply(dots, function(x) is.symbol(x) ||
##         is.character(x), NA, USE.NAMES = FALSE)))
##         stop("... must contain names or character strings")
##     names <- vapply(dots, as.character, "")
##     if (length(names) == 0L)
##         names <- character()
##     list <- .Primitive("c")(list, names)
##     .Internal(remove(list, envir, inherits))
## }
## <bytecode: 0x7fa7c1c4eb48>
## <environment: namespace:base>
```

```
data(iris, package = "MASS")
```

```
## Warning in data(iris, package = "MASS"): data set 'iris' not found
```

```
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.

```
#?rbind
iris <- iris[iris$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".

```
nrow(iris[iris$Species=="versicolor",])
```

```
## [1] 50
```

```
y = c(rep(0,nrow(iris[iris$Species=="setosa",])),
      rep(1,nrow(iris[iris$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?

```
X = data.frame(iris$Sepal.Length)
Xy = cbind(X, y)
n = nrow(Xy)

max = 1000
w_vec = c(0, 0)

for (j in 1 : max){
  for (i in 1 : n){
    x_i = c(1, Xy[i, 1])
    yhat_i = ifelse(sum(x_i * w_vec) > 0, 1, 0)
    w_vec = w_vec + (y[i] - yhat_i) * x_i
  }
}
w_vec
```

```
## [1] -143.0   33.7
```

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.
##
```

Write your answer here in English.

Yes, it makes sense that the estimated value of the threshold parameter is 5.1 because the median is 5.0. The data set was exactly 50% setosa and 50% virginica, meaning that if the data is normally distributed, the threshold value would lie somewhere close to the middle of the data with setosas being on one side of the threshold value and virginica being on the other.

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
rm
```

```
## function (..., list = character(), pos = -1, envir = as.environment(pos),
##     inherits = FALSE)
## {
##     dots <- match.call(expand.dots = FALSE)$...
##     if (length(dots) && !all(vapply(dots, function(x) is.symbol(x) ||
##         is.character(x), NA, USE.NAMES = FALSE)))
##         stop("... must contain names or character strings")
##     names <- vapply(dots, as.character, "")
```

```
##     if (length(names) == 0L)
##         names <- character()
##     list <- .Primitive("c")(list, names)
##     .Internal(remove(list, envir, inherits))
## }
## <bytecode: 0x7fa7c1c4eb48>
## <environment: namespace:base>
```

```
X = data.frame(iris)
X$Species = NULL
e = c(rep(0,100))
Xy = cbind(X, y, e)

n = nrow(Xy)
w_vec = c(0, 0, 0, 0)
max = 1000
error = 0
error_count = 0

for (j in 1 : max){
  for (i in 1 : n){
    x_i = c(Xy[i,1], Xy[i,2], Xy[i,3], Xy[i,4])
    yhat_i = ifelse(sum(x_i * w_vec) > 0, 1, 0)
    e[i] = ifelse((yhat_i != y[i]), 1, 0)
    w_vec = w_vec + (Xy[i,5] - yhat_i) * x_i
  }
}
w_vec
```

```
## [1] -1.1 -3.6  5.2  2.2
```

```
total_error = sum(1==Xy[,"e"])
error_rate = total_error / nrow(Xy)
error_rate
```

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