HW01p

Rachel Hia

February 23, 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.

```
#TO-DO
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.

```
#TO-DO
v= seq(-100, 100, by =1)
```

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(x){
  temp = NULL
  for(i in x){
    temp= c(i,temp)
  }
temp
}
```

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
#TO-DO
values = c(rep(0,.5*n*n), rep(1, .25*n*n), rep(2,.25*n*n))
R = matrix(sample(values), nrow = n, ncol = n)
#table(R)
#R
```

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(n*n, sum(c(R)==0) + sum(c(R)==1) + sum(c(R)==2))
#tab = table(R)
#tab[3]
#TO-DO test that there are exactly 625 2's
expect_equal(625,sum(c(R)==2)) #this is the test for 625 2's
```

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

```
#TO-DO

R[sample(x = n*n, size = n*n*0.3)] = 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.

```
#TO-DO
r_names = c()
for (i in 1:n){
  r_names = c(r_names, sum(R[i,], na.rm = TRUE))
}
row.names(R) = r_names
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.

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

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

```
#TO-DO
avg = mean(v)
se = sd(v)/(sqrt(length(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
q1 = quantile(v,0.05)
q2 = qnorm(p = .05, mean = -10, sd = sqrt(10))
expect_equal(as.numeric(quantile(v,0.05)), expected = qnorm(0.05, mean = -10, sd = sqrt(10)), tolerance
```

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
inverse_quantile_obj = ecdf(v)
expect_equal(inverse_quantile_obj(-7000),pnorm(-7000, mean = -10, sd = sqrt(10)), tol = 0.05)
```

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
n=8
key = 'A'
my_list = list()
keys = c("A", "B", "C", "D", "E", "F", "G", "H")
for(i in 1:n){
  key = keys[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
## 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 later arrays, does it make sense given the dimensions of the arrays?

Answer here in English.

Object.size provides an estimate of the memory that is being used in each key of the my_list object. lapply returns the keys and the corresponding bytes connected to each key. The byte sizes do make sense for the later arrays. As the dimensions increase, the memory used increases expontentially.

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

```
#TO-DO
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.

```
#TO-DO
#data(iris, package = "MASS")
data(iris)
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
#iris
#summary(iris)
#View(iris)
```

The column heads of this data frame are Sepal.Length, Sepal.Width, Petal.Length, Petal.Width, and Species. There are 3 different species or irises. The Sepal.Length values range from 4.3 to 7.9 with a mean of 5.843. That means the average sepal.length for the three species is 5.843.

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

f = iris[iris$Species != "virginica", ]

#View(f)
```

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

```
##   [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [36] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [71] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 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?

```
#TO-DO
#threshold model -make a cut in sepal.length, everything above the cut is a one
#make the best possible cutoff to predict 1 or 0
#go through every data point in the set, find the one with the lowest amount of errors

#y_binary = as.numeric(f$Sepal.Length)

#y_binary = ifelse(y_binary == 1, 0, 1)
X1 = as.matrix(cbind(f[, 1, drop = FALSE]))
MAX_ITER = 100
w_vec = 0 #rep(0, 2)
#w_vec = y
for (iter in 1 : MAX_ITER){
  for (i in 1 : nrow(X1)){
    x_i = X1[i]
    yhat_i = ifelse(sum(x_i * w_vec) > 0, 1, 0)
    y_i = y[i]
    w_vec = w_vec + (y_i - yhat_i) * x_i
  }
}

#plot(iris$Sepal.Length)

#error rate
yhat = ifelse(X1 %*% 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 == "versicolor", "Sepal.Length"])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   4.900   5.600   5.900   5.936   6.300   7.000
```

Write your answer here in English. Yes. The summaries provide us enough information to be able to visualize the line between the two sets of Sepal.length data, setosa and versicolor. The min and max's for each are similar, meaning there would be some error because the data is not linearly separable. Plotting the Sepal.length data will show you that it is not linear separable. Using just Sepal.length is not a good enough way to identify species.

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?

```r
#TO-DO - use all 4 features - if 0 it may be separable
#code perceptron learning algorithm and get w as a result


X2 = as.matrix(
cbind(y, f[, 1, drop = FALSE], f[, 2, drop = FALSE], f[, 3, drop = FALSE], f[, 4, drop = FALSE]))
MAX_ITER = 100
w_vec = rep(0, 5)
#w_vec = y
for (iter in 1 : MAX_ITER){
  for (i in 1 : nrow(X2)){
    x_i = X2[i,]
    yhat_i = ifelse(sum(x_i * w_vec) > 0, 1, 0)
    y_i = y[i]
    w_vec = w_vec + (y_i - yhat_i) * x_i
  }
}
#w_vec


#error rate
yhat = ifelse(X2 %*% w_vec > 0, 1, 0)
sum(y != yhat) / length(y)
```

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

```r
#The 4 features together make it separable.
```