HW01p

Bryan Lliguicota

February 17, 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")} #installs pacman if necessary but does not load it!
```

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

```
pacman::p_load(devtools)
pacman::p_load_gh("r-lib/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 (v){
  tmp_vec<-c()
  size<- length(v)
  for (i in 1:size){
    tmp_vec[i]=v[size]
    size <- size -1
  }
  tmp_vec
}
```

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
R <- c(rep(1, 625), rep(2, 625), rep(0,1250 ))
R<- sample(R)
R <-matrix( R,nrow = n , ncol = n)
```

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

```
test<- (c(R))
expect_equal(dim(R), c(n, n))
#TO-DO test that the only unique values are 0, 1, 2
expect_equal(sort(unique(test)),c(0,1,2))
#TO-DO test that there are exactly 625 2's
test<-factor(test)
tmp<-summary(test)
expect_equal(tmp[[3]],625)
rm(test,tmp)
```

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

```
test<-c(R)
size <- (length(test)*.30);
test[1: size]<-NA
test<-sample(test)
R<-matrix(test,nrow=n,ncol=n)
rm(test, size)
```

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.

```
#insertion sort
for(i in 2:n){
  j<- i-1
  tmp<-i
    while((j>=1) && (sum(R[i , ], na.rm = TRUE) > sum(R[j , ], na.rm = TRUE))){
        R[tmp , ]<- R[j,]
        j<j-1
        tmp<- tmp -1
    }
  R[j+1,]=R[i, ]
}
rm(j,i,tmp)
```

Test using the following code.

```
?expect_gt
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_of_v <- median(v)
stand_err_of_v<-sd(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.

```
Five_perc_tile_of_v<-quantile(v, probs = 0.05)
test1<-Five_perc_tile_of_v[['5%']]
test2<-qnorm(c(.05),mean=-10,sd=sqrt(10))
expect_equal(test1,test2,tol=.1)
rm(test1,test2,Five_perc_tile_of_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.

```
inverse_quantile_obj = ecdf(v)
test1<-inverse_quantile_obj(-7000)
test2<-pnorm(c(-7000),mean=-10,sd= sqrt(10))
expect_equal(test1,test2, tol = .1)
rm(test1,test2)
```

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()
entry_names=c("A","B","C","D","E","F","G","H")
counter<-1
for(index in entry_names){
  tmp<-counter*counter
    if(index=="A"){
    my_list[[index]]<-array(1 : tmp, counter)
    }else{
    my_list[[index]]<-array(1 : tmp, dim = rep(counter,counter))
    }
  counter = counter +1
}
#my_list
?array
```

Test with the following uncomprehensive tests:

```
expect_equal(my_list$A, array(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
```

```
?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.

lappy returns a list of the same length as "my_list", each element in the list retruned by lapply has the function object.size applied to it. The function object.size() returns an estimate of the memory that us being used to store it. Given the dimensions of the array this does make sence because of the rate at which the number of elements in the array increase. What did seem suprising was the amount of memory taken by an array(1:1,1) (208 bytes) as compared to a vector c(1) which is only 48 bytes.

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
##
##
##
```

Provide a Summary of the colums and provide a few descriptive sentences about the distributions: The data set 'iris' is composed of 3 species (setosa,versicolor,virginica) and four measurment of each sampe. From the summary of the columes we can see for all 4 properties the max and min sepal.length,width as well as teh max and min petal lenth,width. Also the most common properties (length and width for the sepal and petal).

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.

```
rm_indicies<-c()
for(i in 1:150){
  if((iris[i,"Species"]=="virginica")){
      rm_indicies<- c(rm_indicies,i)
  }
}
iris<-iris[-rm_indicies,]
rm(i,rm_indicies)
summary(iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.000   1st Qu.:2.800   1st Qu.:1.500   1st Qu.:0.200
##  Median :5.400   Median :3.050   Median :2.450   Median :0.800
##  Mean   :5.471   Mean   :3.099   Mean   :2.861   Mean   :0.786
##  3rd Qu.:5.900   3rd Qu.:3.400   3rd Qu.:4.325   3rd Qu.:1.300
##  Max.   :7.000   Max.   :4.400   Max.   :5.100   Max.   :1.800
##        Species
##  setosa    :50
##  versicolor:50
##  virginica : 0
##
##
##
```

```
dim(iris)
```

```
## [1] 100   5
```

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.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?

```
#we want to perdict if the flower is a versicolor(1) or not (0)
#get the num of rows
n=nrow(iris)
```

```
#create a matrix to rep the threhold and num_of_error
#the threshold can be thought as part of an indicator function,
#we will have numerous thresholds (even though some repeat).
#Our i_th threshold in the i_th iteration is compared to the entire colume sepal.length
num_errors_by_parameter= matrix(NA,nrow=n,ncol=2)
colnames(num_errors_by_parameter)<-c("Threshold_param","num_of_errors")
#we create the y_logical vector, FALSE == 0 == not versicolor, TRUE ==1 ==versicolor, using our y vecto
y_logical = y==1
#for each i_th iteration we have a i_th thresold and a i_th num_of_err
#if for any i_th row  in the iris datatable the sepal.length is greater than the i_th threshold AND
# its not equal to the i_th value in y_logical, we incldue it in the sum of the num_err variable.
#num_err basically say our "perdication was wrong" based on the theshold and y_logical.
for(i in 1:n){
  threshold<-iris[i,"Sepal.Length"]
  num_err<-sum((iris[ ,"Sepal.Length"] > threshold) != y_logical)
  num_errors_by_parameter[i,]<-c(threshold,num_err)
}
best_row<- order(num_errors_by_parameter[,"num_of_errors"])[1]
num_errors_by_parameter[best_row,]
```

```
## Threshold_param   num_of_errors
##             5.4            11.0
```

```
#The Expected value of the threshold parameter is 5.4 and the number of errors are 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 == "versicolor", "Sepal.Length"])
```

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

```
rm(list = setdiff(ls(), "iris"))
```

Write your answer here in English.

Yes, this does make sense. For the "setosa" species we have a "sepal.length" threshold of 5.4. Looking at the summary (summary(iris[iris$Species == "setosa", "Sepal.Length"])) for sepal the mean lies below the threshold, whats even more the 3rd Quantile lies below the threshold which is a good sign. the Num of errors can also be understood as the Max sepal length is 5.8 cm. For the summary of versicolor, its noticible that on average their sepal length is greater than setosas length. A good sign is the 1st quantile which is greater than the threshold. The min sepal lenght of the versicolor species contributes to the num_of_errors for the 5.4cm 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?

```
X=iris[, 1: 4]
y_binary = as.numeric(iris$Species == "versicolor")
MAX_ITER = 1000
w_vec = c(5.4,rep(0, 4))
X1 = as.matrix(cbind(1, X[, 1:4, drop = FALSE]))
```

```
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_binary[i]
    w_vec = w_vec + (y_i - yhat_i) * x_i
  }
}
w_vec
```

```
##            1 Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##          4.4         -1.3         -4.1          5.2          2.2
```

```
y_binary
```

```
##   [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 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 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 1 1 1 1 1 1 1 1 1
```

Error Rate:

```
yhat = ifelse( X1 %*% w_vec > 0, 1, 0)
sum(y_binary != yhat[,1]) / length(y_binary)
```

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