HW01p

Greg Maghakian

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.

```
pacman::p_load(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(x){
 count=0
  for(i in x){
   count=count+1
 }


   y=NULL
  for(i in count: 1){
    y=c(y,x[i])
  }
  y
}
```

Test using the following code:

```
expect_equal(my_reverse(c("A", "B", "C")), c("C", "B", "A"))
expect_equal(my_reverse(v), rev(v))
x=-100:100
expect_equal(my_reverse(x), rev(x))
```

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
seq1=sample(c(rep(0,1250),rep(1,625),rep(2,625)),2500)

table(seq1)
```

```
## seq1
##    0    1    2
## 1250  625  625
```

```
R=matrix(seq1, nrow=n, ncol=n)

#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

test=function(x){
  for(i in x){
  if(i !=0 && i !=1 && i!= 2)
    count=1
  else
    count=0}
count
  }
test(R)
```

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

```
expect_equal(test(R), 0)
#TO-DO test that there are exactly 625 2's

test_1=function(x){
  count=0
  for(i in x){
    if(i==2)
      count=count+1
  }
  count
}
test_1(R)
```

```
## [1] 625
```

```
expect_equal(test_1(R), 625)
```

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

```
q=rbinom(2500,1,.3)

table(q)
```

```
## q
##    0    1
## 1781  719
```

```
for(i in 1:length(q)){
  if(q[i]==1){q[i]=NA}

  }

table(q)
```

```
## q
##    0
## 1781
```

```
R=c(R)

for(i in 1:2500){
  if(is.na(q[i])){
    R[i]=NA
  }

}


R=matrix(R , 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.

```
copy0=R

for(i in 1:length(R)){
  if(is.na(copy0[i])) {
    copy0[i]=0

  }
}
t=rowSums(copy0)
t
```

```
##  [1] 25 31 22 25 21 25 28 34 30 29 26 23 25 26 30 25 28 21 30 24 24 28 28
## [24] 30 22 24 27 27 33 21 29 28 20 36 24 25 29 20 36 22 25 32 19 22 26 29
## [47] 38 26 24 33
```

```
sums0=order(t,decreasing=TRUE)
  sums0
```

```
##  [1] 47 34 39  8 29 50 42  2  9 15 19 24 10 31 37 46  7 17 22 23 32 27 28
## [24] 11 14 45 48  1  4  6 13 16 36 41 20 21 26 35 49 12  3 25 40 44  5 18
## [47] 30 33 38 43
```

```
for(i in 1:n){
  if(sums0[i]!=i)
    copy0[i,]=R[sums0[i],]
}



rowSums(copy0,na.rm=TRUE)
```

```
##  [1] 38 36 36 34 33 33 32 31 30 30 30 30 29 29 29 29 28 28 28 28 28 27 27
## [24] 26 26 26 26 25 25 25 25 25 25 25 24 24 24 24 24 23 22 22 22 22 21 21
## [47] 21 20 20 19
```

```
R=copy0
```

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.

```
n=1000
v=round(rnorm(n=1000, mean=-10, sd=sqrt(10)))
mean(v)
```

```
## [1] -10.159
```

```
var(v)
```

```
## [1] 9.857577
```

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

```
avg= sum(v)/length(v)
mean(v)
```

```
## [1] -10.159
```

```
stderror=sd(v)/sqrt(length(v))
stderror
```

```
## [1] 0.09928533
```

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.

```
percentile=quantile(v, probs = 0.05)
percentile
```

```
##  5%
## -15
```

```
x=qnorm(.05, mean=-10, sd=sqrt(10))
expect_equal(as.numeric(x), as.numeric(percentile), tol=stderror)


x
```

```
## [1] -15.20148
```

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 = ecdf(v)
t=inverse_quantile(-7000)
t
```

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

```
p=pnorm(-7000,-10,sqrt(10))
p
```

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

```
expect_equal(t,p,tol=stderror)
```

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(A=array(1:1,dim=c(1,1)),B=array(1:4,dim=c(2,2)), C=array(1:9,dim=rep(3,3)), D=array(1:16,

#alternatively


length=8
my_list=list()
for(i in 1:length){
  my_list[[LETTERS[i]]]=array(1:i^2, dim=rep(i,i))
}
```

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
```

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?

lapply is basically a for loop that takes your vector/ list and applys a function to it for each element in that list. object.size tells us, in this case for the output above, the amount of allocated memory for the arrays of size 1...8x8x8....For the later arrays it makes sense as well as the allocated memory is larger.

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
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
##
##
##
```

```
IQR(iris$Sepal.Length)
```

```
## [1] 1.3
```

```
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
```

```
#
```

The data frame iris, through using summary(iris) gives us a great description of what the data frame holds. There are 150 rows from which there are 3 species of iris flowers that are 50 setosa, 50 versicolor, and 50 virginica. The names of the columns are Sepal. Length, Sepal Width, Petal. Length, Petal. Width in cm and Species. We can view the specific mean, min, max, and IQR for these certain aspects. For example, the average for Petal Length is 2.861.

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".

```
n=nrow(iris)
y=0
for(i in 1:n){
  if(iris[i,5]=="versicolor")
    y[i]=1
  else
    y[i]=0
}
is.vector(y)
```

```
## [1] TRUE
```

```
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 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
##  [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
```

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
iterations=1000
w=0
X =as.matrix(iris[,1])

for (iter in 1 : iterations){
  for (i in 1 : nrow(X)){
    x_i = X[i, ]
    yhat_i = ifelse(sum(x_i * w) > 0, 1, 0)
    yreal = y[i]
    w = w + (yreal - yhat_i) * x_i
  }
}
w
```

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

```
yhat = ifelse(X %*% w > 0, 1, 0)
```

```
error=sum(y != yhat) / length(y)

#
error
```

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

This model makes 50 errors as the error rate is .50

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.

We can see that the mean sepal length and summary in general for sepal length for both setosa and versicolor species are about the same. This allows us to understand perhaps why the error rate is pretty high as it is hard to determine flower species from this characteristic. Perhaps also our model is bad because our algorithm could be better as well.

10. Fit a perceptron model explaining `y` using all four 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


iterations=1000
wvec=c(0,0,0,0,0)
X1 =as.matrix(cbind(1,(iris[,1:4])))

for (iter in 1 : iterations){
  for (i in 1 : nrow(X1)){
    x_i = X1[i, ]
    yhat_i = ifelse(sum(x_i * wvec) > 0, 1, 0)
    yreal = y[i]
    wvec = wvec + (yreal - yhat_i) * x_i
  }
}
wvec
```

```
##            1 Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##         -1.0         -1.1         -3.6          5.2          2.2
```

```
yhat = ifelse(X1 %*% wvec > 0, 1, 0)

error=sum(y != yhat) / length(y)



error
```

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

The total number of errors is zero. Could this data be linearally separable? I believe so.