Recitation 9 functions, regression, and subclassification

Seamus Wagner

October 25, 2021

Based off of feedback from most of you, a bit of review for function calling, a review of the past lab, and some outlines for the code we reviewed for regression are in order. There are still 2 recitations I would like to give based on feedback: text analytics, and GIS work. Those will be conditional as long as course coding is going well. I will start with some overview of function calling, then we will go over the subclassification code and talk about the homework, then we will discuss the upcoming lab, answer any of your questions, and review the demo code.

Function calling

Function calling is something important that we reinforce in multiple ways throughout the course, but maybe don't check for intuition well enough. The goal of functions (as I see it) is to provide a clean way to execute a task repeatedly and transparently in R. They typically take some sort of input (vectors, matrix, dataframe, etc.) and then perform a task and produce a result (single value, vector, matrix, data frame, etc.). Functions are incredibly versatile and are used to create most of the packages we use in our daily R analysis. Thinking of writing a function, what goes between the regular parentheses () are the inputs, what the function needs to operate using. What comes between the curly braces {} are the actions to be executed. I think one confusing aspect based on my discussions with some of you is how functions interact with iteration commands. Functions return a value typically (though you can scale it). Typically, we want a isolated function to take some data, transform it and compute something from it. We then want to feed that function into an iteration commands (for loop/apply functions, etc.) and replicate the function over and over. One issue when writing a function with iteration in mind is that you sometimes build the iteration as inputs, which is not ideal. You do not want your function to have 1:1000 in it, you want to feed you function into a apply function 1:1000 times. We will go over some of the functions from the subclassification demo below.

The structure of the demo codes typically walk through every part of a function outside of it being within a function. We then copy/paste that code into a function and it runs well. Keep that in mind, that William builds a function and then works backwards to go over each part. That may not be what you do.

```
rm(list=ls())
set.seed(187654)
# Let's start with the actual function and then the iteration

# Here is the function isolated, each line runs and it will give a dataset
# Yet if you run this and don't run it through an apply function, it doesn't return
# anything
simulate_fake_dataset <- function(iteration) {
  n_strata <- 5000
  strata <- data.table(
    X = 1:n_strata,
```

```
    p_Y0 = plogis(-1 + rnorm(n_strata)),
    p_Y1 = plogis( 1 + rnorm(n_strata)))
  strata[, p_D := plogis(2 * (p_Y1 - p_Y0))]
  n_obs <- 50000
  gods_data <- data.table(
    i = 1:n_obs,
    X = sample(n_strata, n_obs, replace = TRUE, prob = .01 + rpois(n_strata, 1)))
  gods_data <- merge(gods_data, strata, by = "X", all.x = TRUE, sort = FALSE)
  gods_data[, Y1 := rbinom(.N, 1, p_Y1)]
  gods_data[, Y0 := rbinom(.N, 1, p_Y0)]
  gods_data[, D := rbinom(.N, 1, p_D)]
  gods_data[, Y := D * Y1 + (1 - D) * Y0]
  gods_data
}
```

```
# This returns nothing
simulate_fake_dataset()

# Yet this returns a list of 1
test <- lapply(1, simulate_fake_dataset)
head(test)
```

```
## [[1]]
##           X     i       p_Y0      p_Y1       p_D Y1 Y0 D Y
##     1: 3765     1 0.09189249 0.3511646 0.6268073  0  0 1 0
##     2: 3452     2 0.41669441 0.8379979 0.6990140  1  0 1 1
##     3: 3247     3 0.27297492 0.2201131 0.4735937  0  0 0 0
##     4: 4430     4 0.15454628 0.3423229 0.5928001  0  0 0 0
##     5: 2135     5 0.03751430 0.2822914 0.6200014  0  0 0 0
##    ---
## 49996: 1035 49996 0.27291398 0.5969118 0.6565586  1  0 1 1
## 49997:  683 49997 0.07205515 0.5161625 0.7085216  1  0 0 0
## 49998: 4278 49998 0.41246853 0.4505128 0.5190130  0  0 1 0
## 49999: 2684 49999 0.45540098 0.5863899 0.5651224  1  1 0 1
## 50000: 2676 50000 0.21036410 0.7503542 0.7464902  0  0 1 0
```

If we wanted to run this for 1 iteration, we could supply the inputs in the function command instead of using the input for iteration. In this case, we drop the defined n_strata and n_obs from within the function. Instead, we supply these values as inputs in the function. This means that the function is designed to iterate 1 time, not multiple.

```
simulate_fake_dataset1 <- function(n_strata, n_obs) {
  strata <- data.table(
    X = 1:n_strata,
    p_Y0 = plogis(-1 + rnorm(n_strata)),
    p_Y1 = plogis( 1 + rnorm(n_strata)))
  strata[, p_D := plogis(2 * (p_Y1 - p_Y0))]
  gods_data <- data.table(
    i = 1:n_obs,
    X = sample(n_strata, n_obs, replace = TRUE, prob = .01 + rpois(n_strata, 1)))
  gods_data <- merge(gods_data, strata, by = "X", all.x = TRUE, sort = FALSE)
  gods_data[, Y1 := rbinom(.N, 1, p_Y1)]
  gods_data[, Y0 := rbinom(.N, 1, p_Y0)]
```

```
  gods_data[, D := rbinom(.N, 1, p_D)]
  gods_data[, Y := D * Y1 + (1 - D) * Y0]
  gods_data
}
test1 <- simulate_fake_dataset1(n_strata = 5000, n_obs = 50000)
head(test1)
```

```
##       X i      p_Y0      p_Y1       p_D Y1 Y0 D Y
## 1: 4373 1 0.4253732 0.8527517 0.7015641  1  1 0 1
## 2: 4942 2 0.6118041 0.5876743 0.4879374  1  1 1 1
## 3: 3303 3 0.1481868 0.7848349 0.7813065  1  0 1 1
## 4: 3615 4 0.2003985 0.9370459 0.8135577  1  0 1 1
## 5: 1169 5 0.4269002 0.9647316 0.7456723  1  0 1 1
## 6:  321 6 0.6635216 0.3442582 0.3455796  1  1 1 1
```

```
simulate_fake_dataset2 <- function(n_strata, n_obs) {
  strata <- data.table(
    X = 1:n_strata,
    p_Y0 = plogis(-1 + rnorm(n_strata)),
    p_Y1 = plogis( 1 + rnorm(n_strata)))
  strata[, p_D := plogis(2 * (p_Y1 - p_Y0))]
  gods_data <- data.table(
    i = 1:n_obs,
    X = sample(n_strata, n_obs, replace = TRUE, prob = .01 + rpois(n_strata, 1)))
  gods_data <- merge(gods_data, strata, by = "X", all.x = TRUE, sort = FALSE)
  gods_data[, Y1 := rbinom(.N, 1, p_Y1)]
  gods_data[, Y0 := rbinom(.N, 1, p_Y0)]
  gods_data[, D := rbinom(.N, 1, p_D)]
  gods_data[, Y := D * Y1 + (1 - D) * Y0]
  gods_data
}
#test2 <- lapply(1:1000, simulate_fake_dataset2)
# Argument is missing for the two inputs, what happens if we add them?
#test3 <- lapply(1:1000, simulate_fake_dataset2(n_strata = 5000, n_obs = 10000))
# Also does not work, what if we use correct syntax, this works
test4 <- lapply(1:1000, simulate_fake_dataset2, n_obs = 10000)
# Why, well because the first argument in the lapply is the first input from the function
# After the call of function, we use a comma and then supply the others.
head(test4)
```

```
## [[1]]
##        X     i      p_Y0      p_Y1       p_D Y1 Y0 D Y
##     1: 1     1 0.1601116 0.4806176 0.6549822  1  0 1 1
##     2: 1     2 0.1601116 0.4806176 0.6549822  1  0 0 0
##     3: 1     3 0.1601116 0.4806176 0.6549822  0  0 1 0
##     4: 1     4 0.1601116 0.4806176 0.6549822  1  0 1 1
##     5: 1     5 0.1601116 0.4806176 0.6549822  0  0 1 0
##    ---
##  9996: 1  9996 0.1601116 0.4806176 0.6549822  1  0 1 1
##  9997: 1  9997 0.1601116 0.4806176 0.6549822  1  0 0 0
##  9998: 1  9998 0.1601116 0.4806176 0.6549822  0  1 0 1
##  9999: 1  9999 0.1601116 0.4806176 0.6549822  0  0 1 0
## 10000: 1 10000 0.1601116 0.4806176 0.6549822  0  0 1 0
```

```
##
## [[2]]
##        X     i      p_Y0      p_Y1       p_D Y1 Y0 D Y
##     1: 2     1 0.2294713 0.5233381 0.6428449  0  0 1 0
##     2: 2     2 0.2294713 0.5233381 0.6428449  1  0 1 1
##     3: 2     3 0.2294713 0.5233381 0.6428449  1  0 1 1
##     4: 2     4 0.2294713 0.5233381 0.6428449  0  0 1 0
##     5: 2     5 0.2294713 0.5233381 0.6428449  1  0 1 1
##    ---
##  9996: 2  9996 0.2294713 0.5233381 0.6428449  1  0 1 1
##  9997: 2  9997 0.2294713 0.5233381 0.6428449  1  1 0 1
##  9998: 2  9998 0.2294713 0.5233381 0.6428449  0  0 1 0
##  9999: 2  9999 0.2294713 0.5233381 0.6428449  1  0 0 0
## 10000: 2 10000 0.2294713 0.5233381 0.6428449  1  0 1 1
##
## [[3]]
##        X     i       p_Y0      p_Y1       p_D Y1 Y0 D Y
##     1: 3     1 0.37140272 0.4333323 0.5309253  0  0 1 0
##     2: 3     2 0.37140272 0.4333323 0.5309253  0  1 0 1
##     3: 3     3 0.37140272 0.4333323 0.5309253  1  1 1 1
##     4: 3     4 0.37140272 0.4333323 0.5309253  1  0 0 0
##     5: 3     5 0.37140272 0.4333323 0.5309253  0  0 0 0
##    ---
##  9996: 2  9996 0.05774274 0.7437182 0.7976952  1  0 1 1
##  9997: 1  9997 0.20063711 0.5545447 0.6699182  1  1 1 1
##  9998: 3  9998 0.37140272 0.4333323 0.5309253  1  1 0 1
##  9999: 3  9999 0.37140272 0.4333323 0.5309253  1  0 0 0
## 10000: 3 10000 0.37140272 0.4333323 0.5309253  0  0 0 0
##
## [[4]]
##        X     i      p_Y0      p_Y1       p_D Y1 Y0 D Y
##     1: 1     1 0.4585954 0.6135739 0.5768748  1  0 1 1
##     2: 1     2 0.4585954 0.6135739 0.5768748  0  1 1 0
##     3: 1     3 0.4585954 0.6135739 0.5768748  1  0 1 1
##     4: 1     4 0.4585954 0.6135739 0.5768748  1  1 0 1
##     5: 1     5 0.4585954 0.6135739 0.5768748  0  1 0 1
##    ---
##  9996: 1  9996 0.4585954 0.6135739 0.5768748  1  1 1 1
##  9997: 1  9997 0.4585954 0.6135739 0.5768748  0  0 1 0
##  9998: 1  9998 0.4585954 0.6135739 0.5768748  1  0 1 1
##  9999: 1  9999 0.4585954 0.6135739 0.5768748  0  0 0 0
## 10000: 1 10000 0.4585954 0.6135739 0.5768748  1  1 1 1
##
## [[5]]
##        X     i      p_Y0      p_Y1       p_D Y1 Y0 D Y
##     1: 3     1 0.1740690 0.6827736 0.7344676  0  0 0 0
##     2: 1     2 0.1282173 0.9791237 0.8457713  1  0 1 1
##     3: 4     3 0.1928313 0.7424246 0.7501077  1  0 1 1
##     4: 3     4 0.1740690 0.6827736 0.7344676  1  1 0 1
##     5: 4     5 0.1928313 0.7424246 0.7501077  0  0 0 0
##    ---
##  9996: 4  9996 0.1928313 0.7424246 0.7501077  1  0 1 1
##  9997: 4  9997 0.1928313 0.7424246 0.7501077  0  0 1 0
##  9998: 4  9998 0.1928313 0.7424246 0.7501077  1  1 1 1
```

```
##  9999: 1  9999 0.1282173 0.9791237 0.8457713  1  0 0 0
## 10000: 3 10000 0.1740690 0.6827736 0.7344676  1  0 0 0
##
## [[6]]
##        X     i       p_Y0      p_Y1       p_D Y1 Y0 D Y
##     1: 2     1 0.32669819 0.5598414 0.6145044  0  0 1 0
##     2: 2     2 0.32669819 0.5598414 0.6145044  0  0 1 0
##     3: 1     3 0.48365547 0.6967752 0.6049753  1  0 1 1
##     4: 2     4 0.32669819 0.5598414 0.6145044  0  0 0 0
##     5: 2     5 0.32669819 0.5598414 0.6145044  1  0 0 0
##    ---
##  9996: 3  9996 0.03592978 0.8252722 0.8290182  0  0 0 0
##  9997: 2  9997 0.32669819 0.5598414 0.6145044  1  1 1 1
##  9998: 1  9998 0.48365547 0.6967752 0.6049753  1  1 1 1
##  9999: 5  9999 0.47609858 0.6847515 0.6028384  0  1 1 0
## 10000: 2 10000 0.32669819 0.5598414 0.6145044  0  0 1 0
```

We will go over the code from the regression lab from that document, rather than repeating it here. That code is what will be most useful for the upcoming lab.