Corr function

Ahmed Ezzat Afifi

17-10-2014

Here I want to introduce a more sophisticated concepts of R that are beyond the homework but will help you much understanding and working with R.

I've used R markdown to make this document, hope it would be a better presentation of the code.

To make similar documents, will go to File -> New File -> R markdown.

r-studio will probably ask you to download some packages. click ok to download them and you are set to go.

Here we write some R code in code chunks, R compiles them then prints the output below the code chunks.

The corr function requirements

We want to compute the correlation between the two pollutants using R standard cor() function according to a some threshold indicating the number of complete cases inside the whole file. If our file pass that certain threshold then we compute the cor() between them

In other words, say we want to discard every file that has less than 100 complete cases. or any other threshold.

The corr function

now suppose you don't know what the cor function does, the first thing we do is ?cor to get an idea of what arguments it takes, what it computes and what does its output looks like. so its time to do.

```
? cor # Remember, R documentation is your best friend
```

```
## starting httpd help server ... done
```

Alright, skipping statistics details, the cor function two numeric or integer arguments x and y and computes the correlation between them.

Building the algorithm idea

1- we will have to read all files

2- create a variables to save our results in

3- start the for loop

4- check if the file passes the complete.cases threshold or not

5- save our sulfate complete.cases values in a variable, lets call it sulfateComplete

6- save our nitrate complete.cases values in a variable, lets call it nitrateComplete

7- call our corr function with corr(sulfateComplete, nitrateComplete)

Now Let’s check our data on my disk

since they are all identical in structure lets just check the first of them

```
path = c("C:\\Users\\is7yX\\Documents\\specdata\\001.csv") #change to your path when running the code
#Remember in windows use double \\ not one
#Thats because \ is an escape character in R and most languages
#So paths in windows are a little awkward
data <- read.csv(path)
summary(data)
```

```
##          Date         sulfate          nitrate             ID
##  2003-01-01:   1   Min.   : 0.613   Min.   :0.1180   Min.   :1
##  2003-01-02:   1   1st Qu.: 2.210   1st Qu.:0.2835   1st Qu.:1
##  2003-01-03:   1   Median : 2.870   Median :0.4530   Median :1
##  2003-01-04:   1   Mean   : 3.881   Mean   :0.5499   Mean   :1
##  2003-01-05:   1   3rd Qu.: 4.730   3rd Qu.:0.6635   3rd Qu.:1
##  2003-01-06:   1   Max.   :19.100   Max.   :1.8300   Max.   :1
##  (Other)   :1455   NA's   :1344     NA's   :1339
```

Quick hint for later, there are Date and ID variables that we have no use for in our algorithm

The function

Now our function will look like this.

```
corr <- function(directory, threshold = 0) {
        ## 'directory' is a character vector of length 1 indicating
        ## the location of the CSV files

        ## 'threshold' is a numeric vector of length 1 indicating the
        ## number of completely observed observations (on all
        ## variables) required to compute the correlation between
        ## nitrate and sulfate; the default is 0

        ## Return a numeric vector of correlations

        # 1- Read all files in directory with full paths

        files <- list.files(directory, pattern=".csv", full.names=TRUE)

        corResult <- c() #Variable to save our results in

        #Now unlike complete and pollutantsmean our function here
        #didn't give us an id argument. so we will need to figure out
        #a method to know how many files in our directory.
        #Aaaaand that's the length of our files vector :)
        #since every element inside it is a full path of a file in directory

        for(i in 1:length(files)) {
                #as usual the first step is to read the data into a variable
                data <- read.csv(files[i])

                #Append to the corrolation vector
                #if threshold of complete data is met

                if(sum(complete.cases(data)) > threshold)
                        {
                        #get the matched complete cases of 2nd and 3rd column of the data
                        #ie: sulfate  and nitrate
                        #Remember complete.cases() returns the number
                        #of rows that are complete across all data
                        #so we are asking are for those exact rows
                        #in second column
                        sulfateComplete <- data[complete.cases(data),2]

                        nitrateComplete <- data[complete.cases(data), 3]

                        #now we have to compute the correlation according to
                        #our saved variables

                        correlation <- cor(sulfateComplete, nitrateComplete)

                        #append the result vector
                        #with the new data of the every file
                        corResult <- append(corResult, correlation)

                        }

        }

        return(corResult)

}
```

Now that actually works, you can submit this code and it will pass correctly.

But we can make it faster and less memory consuming!

That’s big data

1- every iteration of the for loop we load 2 unneeded columns, over thousands of rows. that's actually big waste of memory. 2- we may get away with it now, but suppose you want to run more sophisticated algorithm over much larger set of data

we have 2 options to discard this waste

1- inside the for loop after reading the data

```
path <- c("C:\\Users\\is7yX\\Documents\\specdata\\001.csv")  #change to your path when running the co
de

data <- read.csv(path)

#we build a new skimmed data frame consists of only variables we will be dealing with
skimmedData <- data.frame(sulfate = data$sulfate, nitrate = data$nitrate)

#and later on we work with our skimmedData
#and remove our original data frame from memory: rm(data)

summary(skimmedData)
```

```
##     sulfate          nitrate
##  Min.   : 0.613   Min.   :0.1180
##  1st Qu.: 2.210   1st Qu.:0.2835
##  Median : 2.870   Median :0.4530
##  Mean   : 3.881   Mean   :0.5499
##  3rd Qu.: 4.730   3rd Qu.:0.6635
##  Max.   :19.100   Max.   :1.8300
##  NA's   :1344     NA's   :1339
```

The problem with this method is that fact that it is naive and really does nothing but load full data then skim it.

A better solution

now R provides a good option while reading csv files. its called colClasses.

colClasses tells R the type "class() of" to expect in every column it reads.

Think about, how did R figure it out on its own that every column in our data is either integer, numerical or string, date..?

it has an authomatic functions that detects the class of every column, in huge data sets of millions of rows, this automatic type detection slows down the data reading process. and if your data was not clean enough it could screw up the data types.

like with the Date variable. did you notice that R reads it as a factor not date type?

```
path <- c("C:\\Users\\is7yX\\Documents\\specdata\\001.csv")
data <- read.csv(path)
class(data$Date)
```

```
## [1] "factor"
```

If we want to use date functions over our data after loading it, we would be using as.Date()

```
path <- c("C:\\Users\\is7yX\\Documents\\specdata\\001.csv")
data <- read.csv(path)
data$Date <- as.Date(data$Date)
#we are telling are to take the Date column and and put it back after applying as.Date() over it
#Aaaand it works!
class(data$Date)
```

```
## [1] "Date"
```

Great!

now if we want to load our data with everythig correct from first time?

```
path <- c("C:\\Users\\is7yX\\Documents\\specdata\\001.csv")
#We are telling R to expect the class of every column
#Remember order is important
data <- read.csv(path, colClasses = c("Date", "numeric", "numeric", "numeric"))
class(data$Date)
```

```
## [1] "Date"
```

Now what if we want to discard columns? easy! use “NULL” as data type to expect!

```
path <- c("C:\\Users\\is7yX\\Documents\\specdata\\001.csv")
#We are telling R to discard the first and 4th columns, don't load them
#Remember again order is important
data <- read.csv(path, colClasses = c("NULL", "numeric", "numeric", "NULL"))
#See? only two variables
summary(data)
```

```
##     sulfate          nitrate
##  Min.   : 0.613   Min.   :0.1180
##  1st Qu.: 2.210   1st Qu.:0.2835
##  Median : 2.870   Median :0.4530
##  Mean   : 3.881   Mean   :0.5499
##  3rd Qu.: 4.730   3rd Qu.:0.6635
##  Max.   :19.100   Max.   :1.8300
##  NA's   :1344     NA's   :1339
```

Using colClasses makes are skip a lot of calculations. you can see here, in R-Bloggers (http://www.r-bloggers.com/using-colclasses-to-load-data-more-quickly-in-r/) how faster this could be in larger datasets.

And that's it, now we can modify our original function to be like this

```
#I've callded here corr2. if you want to submit this one, rename it to corr instead
corr2 <- function(directory, threshold = 0) {

        files <- list.files(directory, pattern=".csv", full.names=TRUE)
        corResult <- numeric()

        for (i in 1:length(files)) {
                data <- read.csv(files[i], header=TRUE, colClasses=c("NULL", "numeric", "numeric", "N
ULL"))
                if (sum(complete.cases(data)) > threshold) {
                        #See? here we used first and second columns
                        #Instead of 2nd and 3rd
                        sulfateComplete <- data[complete.cases(data),1]
                        nitrateComplete <- data[complete.cases(data), 2]
                        corResult <- c(corResult, cor(sulfateComplete, nitrateComplete))
                }
        }

        corResult
}
```

Let's check if we did everything correctly

Let's check if both outputs are identical!

```
#Set path
path <- c("C:\\Users\\is7yX\\Documents\\specdata")

## our first function
corr <- function(directory, threshold = 0) {

        files <- list.files(directory, pattern=".csv", full.names=TRUE)
        corResult <- c() #Variable to save our results in

        for(i in 1:length(files)) {
                data <- read.csv(files[i])
                if(sum(complete.cases(data)) > threshold)
                        {
                        sulfateComplete <- data[complete.cases(data),2]
                        nitrateComplete <- data[complete.cases(data), 3]
                        correlation <- cor(sulfateComplete, nitrateComplete)
                        corResult <- append(corResult, correlation)
                        }
        }
        return(corResult)
}

### AND second function

corr2 <- function(directory, threshold = 0) {

        files <- list.files(directory, pattern=".csv", full.names=TRUE)
        corResult <- numeric()

        for (i in 1:length(files)) {
                data <- read.csv(files[i], header=TRUE, colClasses=c("NULL", "numeric", "numeric", "N
ULL"))
                if (sum(complete.cases(data)) > threshold) {
                        #See? here we used first and second columns
                        #Instead of 2nd and 3rd
                        sulfateComplete <- data[complete.cases(data),1]
                        nitrateComplete <- data[complete.cases(data), 2]
                        corResult <- c(corResult, cor(sulfateComplete, nitrateComplete))
                }
        }

        corResult
}

#NOW call the first function with
corr1Result <- corr(path)
corr2Result <- corr2(path)

#Check if both solutions are identical

identical(corr1Result, corr2Result)
```

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

And that’s it. Hope you find it useful and this document is of much help :)