Post 02 - Essential R Packages for the Aspiring Data Scientist

Matthew Sit
November 30, 2017

Introduction and Motivation

I recently came across a post on LinkedIn from David Langer, an experienced data science professional. In the post, there is a list of some R packages that any aspiring data scientist should know about simply because of how frequently they are used in practice and in industry. While I recognized some of the packages mentioned, I was unfamiliar with quite a few. Since we have been learning R as a tool for data science this semester in Statistics 133, I decided to write this post in order to better acquaint myself and my peer readers with these essential packages so that we have an improved understanding of what R can be used to accomplish and how such tasks can be done most effectively.

In this post, I focus specifically on a few packages that Langer mentioned that we haven't explored in the course yet. These are most relevant to us as students right now because they offer utilities that allow us to interact better with the tools we already know. Langer also mentions other packages which are useful for interfacing with SQL servers and for performing machine learning; since these concepts are quite out of scope for this course, I chose to only describe one machine learning package. I also do not discuss packages that we have already seen, such as ggplot2, stringr, dplyr, and readr.

tidyverse

The tidyverse is a suite of packages that is meant to be useful for everyday data analysis. Packages within tidyverse are meant to be easy to use in conjunction with one another. Some tidyverse packages we've already seen in the course include ggplot2, dplyr, and readr. Here, I discuss and illustrate two more packages under tidyverse that Langer recommends.

tidyr

As the name implies, tidyr is used for cleaning your data set from a formatting standpoint (not necessarily the same as complete pre-processing). Once data is tidy, then it will be easier to use the other packages in the tidyverse. A data set is considered tidy if each observation or data point is represented as a row, if each feature or variable is represented as a column, and if each value is represented as a single cell.

To use this package, install and/or import it as necessary:

```
# install.packages('tidyr')
library(tidyr)
```

```
## Warning: package 'tidyr' was built under R version 3.4.2
```

For this example, we will use a built-in R data set:

```
data('mtcars')
# Convert rownames to a column (added as last column).
mtcars$Make <- rownames(mtcars)
# Sort rows
mtcars <- mtcars[order(mtcars$mpg),]
head(mtcars)
```

```
##                     mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Cadillac Fleetwood  10.4   8  472 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8  460 215 3.00 5.424 17.82  0  0    3    4
## Camaro Z28          13.3   8  350 245 3.73 3.840 15.41  0  0    3    4
## Duster 360          14.3   8  360 245 3.21 3.570 15.84  0  0    3    4
## Chrysler Imperial   14.7   8  440 230 3.23 5.345 17.42  0  0    3    4
## Maserati Bora       15.0   8  301 335 3.54 3.570 14.60  0  1    5    8
##                                  Make
## Cadillac Fleetwood   Cadillac Fleetwood
## Lincoln Continental Lincoln Continental
## Camaro Z28                   Camaro Z28
## Duster 360                   Duster 360
## Chrysler Imperial     Chrysler Imperial
## Maserati Bora             Maserati Bora
```

From the preview above, we can see that the mtcars data set has a column dedicated to each particular attribute. If we wanted to reduce the number of columns and move that information to extra rows instead, we can use `gather()`. What this does is before, we had one car per row, and that row had information on all of the car's attributes, but now we have one car taking up multiple rows and each row only has information on one of the car's attributes.

```
mtcars <- gather(data=mtcars, key=Attribute, value=Value, -Make)
head(mtcars)
```

```
##                  Make Attribute Value
## 1  Cadillac Fleetwood       mpg  10.4
## 2 Lincoln Continental       mpg  10.4
## 3          Camaro Z28       mpg  13.3
## 4          Duster 360       mpg  14.3
## 5   Chrysler Imperial       mpg  14.7
## 6       Maserati Bora       mpg  15.0
```

```
tail(mtcars)
```

```
##               Make Attribute Value
## 347  Porsche 914-2      carb     2
## 348      Fiat X1-9      carb     1
## 349    Honda Civic      carb     2
## 350   Lotus Europa      carb     2
## 351       Fiat 128      carb     1
## 352 Toyota Corolla      carb     1
```

If we wanted to do the inverse operation to our data, we can do so in a similar fashion using `spread()`.

```
mtcars <- spread(data=mtcars, key=Attribute, value=Value)
# Sort rows
mtcars <- mtcars[order(mtcars$mpg),]
# Restore row names
row.names(mtcars) <- mtcars$Make
mtcars$Make <- NULL
head(mtcars)
```

```
##                    am carb cyl disp drat gear  hp  mpg  qsec vs    wt
## Cadillac Fleetwood  0    4   8  472 2.93    3 205 10.4 17.98  0 5.250
## Lincoln Continental 0    4   8  460 3.00    3 215 10.4 17.82  0 5.424
## Camaro Z28          0    4   8  350 3.73    3 245 13.3 15.41  0 3.840
## Duster 360          0    4   8  360 3.21    3 245 14.3 15.84  0 3.570
## Chrysler Imperial   0    4   8  440 3.23    3 230 14.7 17.42  0 5.345
## Maserati Bora       1    8   8  301 3.54    5 335 15.0 14.60  0 3.570
```

lubridate

Another useful tidyverse package we haven't used much is lubridate, which allows for better ease of use when working with times and dates in R.

To use this package, install and/or import it as necessary:

```
# install.packages('lubridate')
library(lubridate)
```

```
## Warning: package 'lubridate' was built under R version 3.4.2
```

```
##
## Attaching package: 'lubridate'
```

```
## The following object is masked from 'package:base':
##
##     date
```

To extract out the month, day, or year out of any date format, you can use the function that models the order of the numbers. These include `dmy()`, `myd()`, `ymd()`, etc.

Here, I have two dates in two formats and I can extract the month, day, and year from both.

```
thanksgiving <- dmy('23/11/2017')
thanksgiving2 <- mdy('11-23-2017')
month(thanksgiving) == month(thanksgiving2)
```

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

```
day(thanksgiving) == day(thanksgiving2)
```

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

```
year(thanksgiving) == year(thanksgiving2)
```

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

We can also work with arithematic involving time, for example.

```
currTime <- now()

# Current time.
currTime
```

```
## [1] "2017-11-24 11:50:11 PST"
```

```
# Adding two hours to current time.
currTime + 2*60
```

```
## [1] "2017-11-24 11:52:11 PST"
```

```
# Convert current time (Pacific) to New York time.
with_tz(currTime, "America/New_York")
```

```
## [1] "2017-11-24 14:50:11 EST"
```

randomForest

Finally, I will explore the randomForest package. In general, a random forest is a machine learning model used to classify incoming data points as one of several classes. It is called a forest because it is composed of several decision tree models. Decision trees originate from the early days of machine learning because it is a very intuitive approach to classification. At each node of the tree, we decide what feature we want to examine and at what threshold do we want to split our data. If a

data point falls below the threshold, then we then proceed by examining the decisions on the left side of the tree. Otherwise, we look at the right side of the tree. This continues until we reach the bottom of the tree, where we hopefully have made enough decisions to classify the data point in question with reasonable confidence.

To use this package, install and/or import it as necessary:

```
# install.packages('randomForest')
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.4.2

## randomForest 4.6-12

## Type rfNews() to see new features/changes/bug fixes.
```

For this example, we will use a built-in R data set:

```
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
```

This dataset seems like something we can use machine learning on. We have several data points on the length and widths of the sepals and petals as well as the species the data point belongs to. With a random forest, we can attempt to classify a new data point by providing our best guess for its species, given all of its measurements.

For reproducibility purposes, I set a random seed:

```
set.seed(0)
```

First, I divide my data set into a training set, which will be used to calibrate my predictive model, and a testing set, which will be used to see how well my model actually works. I also divide the sets by inputs (the measurements, `X`) and output classes (the species, `Y`).

```
# First shuffle our data set
iris <- iris[sample(nrow(iris)),]
# Then divide it up
iris_training_X <- iris[1:100,1:4]
iris_training_Y <- iris[1:100,5]
iris_testing_X <- iris[101:150,1:4]
iris_testing_Y <- iris[101:150,5]
```

Now we use the package to train our model and then see how it does on the testing set:

```
model <- randomForest(x=iris_training_X, y=iris_training_Y, xtest=iris_testing_X, ytest=iris_testing_Y)
model$test$confusion
```

```
##            setosa versicolor virginica class.error
## setosa         18          0         0  0.00000000
## versicolor      0         15         0  0.00000000
## virginica       0          1        16  0.05882353
```

From the confusion matrix output shown, we see that out of the 50 testing points that we used, most of the data points were classified correctly (the counts along the diagonal, i.e. setosas that were classified as setosas, versicolors classified as versicolors, virginicas classified as virginicas). The number of points that were misclassified is very low (the counts off of the diagonal).

Conclusion

Throughout this course, we have learned about the fundamentals of R and also several popular packages. In this post, I filled in some of our knowledge gaps by explaining some remaining packages that we have not seen yet but our widely used in practice. Thus, the take-home message is two-fold: first, R has a lot of interesting packages that are useful for data science and second, this course has given us a strong enough foundation of the language and as a result, we are now able to learn about and take advantage of packages we've never seen before as we discover and need them.

References

- Post on LinkedIn, David Langer, https://www.linkedin.com/feed/update/urn:li:activity:6337309248229380096.
- R Built-in Data Sets, Statistics tools for high-throughput data analysis, http://www.sthda.com/english/wiki/r-built-in-data-sets.
- Tidyr, http://tidyr.tidyverse.org/.
- Lubridate, http://lubridate.tidyverse.org/.
- Decision Trees and Random Forests, CS 189 Fall 2017, UC Berkeley, http://www.eecs189.org/static/notes/n23.pdf.
- Titanic: Getting Started With R - Part 5: Random Forests, Trevor Stephens, http://trevorstephens.com/kaggle-titanic-tutorial/r-part-5-random-forests/.
- Random Forests in R, R-bloggers, Anish Singh Walia, https://www.r-bloggers.com/random-forests-in-r/.