Introduction to Data Processing with R

Jon Clayden <j.clayden@ucl.ac.uk>

DIBS Teaching Seminar, 11 Dec 2015

Photo by José Martín Ramírez Carrasco
https://www.behance.net/martini_rc
R: Background and status

- A free and open-source implementation of S
- Appeared 1993; current version is 3.2.3
- Core strength is statistics, but very good at handling and manipulating data
- Increasingly used by Google, Microsoft, Oracle, etc., for data science applications
- Runs on Windows, Mac OS X, Linux, etc.

- Main contributed code repository (CRAN) contains 7000+ packages; growing superlinearly
- Huge array of statistical methods available
- Annual useR! conference
- About 25 packages currently in the medical imaging “task view”
The language

- **High-level**: comparable to MATLAB
- **Vectorised**: you can operate on multiple data elements at once
- A matrix or higher-dimensional array is represented as a vector with a dimension attribute

```r
> 1:4
[1] 1 2 3 4
> x <- matrix(1:4, ncol=2)
> x
 [,1] [,2]
[1,] 1 3
[2,] 2 4
> attributes(x)
$dim
[1] 2 2
```
Lists and data frames

- A list can contain (named or unnamed) variables of different types.
- Elements are accessed using `[[` or `$` syntax.
- A data frame is similar, but elements must be vectors (and will be "recycled").
- Data frames are typically used to store tabular data, like in a spreadsheet.

```r
> x <- list(2:3, a="text", b=1)
> x
[[1]]
[1] 2 3
$a
[1] "text"
$b
[1] 1
> x$b
[1] 1

> y <- data.frame(2:3, a="text", b=1)
> y
   X2.3 a  b
1  2   text 1
2  3   text 1
> y$b
[1] 1 1
```
Factors and formulas

- A **factor** is a vector whose elements can only take certain values (**levels**)

```r
> factor(c(1,2,1,3,1,4))
[1] 1 2 1 3 1 4
Levels: 1 2 3 4
```

```r
> factor(c(1,2,1,3,1,4), levels=1:3)
[1] 1 2 1 3 1 <NA>
Levels: 1 2 3
```

- Note that the element which is not a valid level is set to **NA**, which is used by R to denote **missing values**

- Because of R’s statistical heritage, **formulas** describing relationships between variables are important

```r
> y ~ x
y ~ x
> class(y ~ x)
[1] "formula"
```

- More on this later
Data manipulation

- As in most vectorised languages, widespread use of `for` loops is inefficient and unnecessary.
- The `apply` function allows another function to be applied along one or more dimensions of an array.

```r
> y <- readImageFile("genu.nii")
> image(y[, , 35], col = grey(0:100/100))
```

- `lapply` is used for applying a function to elements of a list, and returning a list containing the results.

```r
> x <- matrix(1:4, ncol = 2)
> apply(x, 1, mean)
[1] 2 3
```

```r
> z <- apply(y, 1:2, max)
> image(z, col = grey(0:100/100))
```
tapply

• **tapply** lets you apply a function to subsets of a vector defined by the levels of a factor

```r
> gender <- factor(c("male","female","male","male","female"))
> age <- c(28,31,30,29,32)
> tapply(age, gender, mean)
female   male
         31.5   29.0
> tapply(age, gender, sd)
female   male
         0.7071068 1.0000000
```
Simple statistics

```r
> a <- rnorm(10); b <- rnorm(10)  # Generate random data
> t.test(a,b)  # Do the means of “a” and “b” differ?

Welch Two Sample t-test

data:  a and b
t = 0.5343, df = 16.344, p-value = 0.6003
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-0.6769035  1.1341339
sample estimates:
 mean of x  mean of y
 0.15810667 -0.07050854

> cor.test(a,b)  # Are “a” and “b” correlated?
(output removed)
```
Using a data frame and formula

- A **formula** is used to define a simple (ANCOVA) model

```
> data(Seatbelts)
> s <- as.data.frame(Seatbelts)
> head(s)

DriversKilled      drivers    front  rear    kms  PetrolPrice  VanKilled law
1           107    1687   867  269  9059   0.1029718        12   0
2            97    1508   825  265  7685   0.1023630         6   0
3           102    1507   806  319  9963   0.1020625        12   0
4            87    1385   814  407 10955   0.1008733         8   0
5           119    1632   991  454 11823   0.1010197        10   0
6           106    1511   945  427 12391   0.1005812        13   0
```

- We are assuming that the **response** (*DriversKilled*) may be modelled using a linear combination of **drivers** and **law**

```
> anova(lm(DriversKilled ~ drivers * law, data=s))
```

```
Analysis of Variance Table

Response: DriversKilled

                   Df Sum Sq Mean Sq       F value   Pr(>F)
drivers            1 97196  97196 734.2697 <2e-16 ***
law                1   693   693  5.2387  0.0232 *
drivers:law        1   256   256  1.9324  0.1661
Residuals         188 24886   132
```

Graphics

- **plot** creates a standard scatter plot; additions can be made with lines or points

```r
> plot(scale(s$DriversKilled), type="l", lwd=2, xlab="month", ylab="")
> lines(scale(s$drivers), col="red", lwd=2)
```

- Other useful plots include histograms (**hist**), box-and-whisker plots (**boxplot**) and 3D surface plots (**persp**)

- Also many more specialised ones
The “Hadleyverse”

- The packages of one very productive R contributor: Hadley Wickham
- Getting data into R: readr, haven, readxl, rvest
- Data manipulation: plyr, dplyr, tidyr
- Working with particular data types: httr, stringr, lubridate
- Visualisation: ggplot2, ggvis, rggobi
- Tools for package developers: devtools, testthat, roxygen2
The ggplot2 package

- Highly recommended; provides a neat mechanism for mapping graphical aesthetics to variables

```r
> library(ggplot2)
> qplot(drivers, DriversKilled, colour=factor(law), data=s) + geom_smooth(method="lm")
```
The dplyr package

- Provides a set of simple, chainable operations which can be applied to data frames

```r
library(dplyr)

# How many drivers were killed on average with and without the seatbelt law?
s %>% group_by(law) %>% summarise(AverageDriversKilled = mean(DriversKilled))

Source: local data frame [2 x 2]

<table>
<thead>
<tr>
<th>law</th>
<th>AverageDriversKilled</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>125.8698</td>
</tr>
<tr>
<td>1</td>
<td>100.2609</td>
</tr>
</tbody>
</table>

# Was the law in place during the worst months?
s %>% filter(DriversKilled > 180) %>% select(law)

<table>
<thead>
<tr>
<th>law</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>
```
The `mmand` and `RNiftyReg` packages

- Standalone packages which are also used by TractoR
- `mmand` is for mathematical morphology and resampling
- `RNiftyReg` is for registration; also has fast functions for reading and writing NIfTI files

- Affine (linear) and nonlinear registration
- 2D or 3D (target may also be 4D)
- Control over cost function, resampling scheme
- Can apply transformations to other images or points, construct affine matrices from scratch
Mathematical morphology

- Basis of morphological image processing
- **Erosion/dilation**: region growing/shrinking
- **Opening/closing**: e.g., removing “holes”
- Additional composite processes
- A kernel, or “structuring element”, acts like a brush
- The mmand package can work in any number of dimensions, with arbitrary kernels

- Bioconductor package **EBImage** also offers (2D) morphology

Wikipedia/Renato Keshet
Binary morphology in 1D

```r
library(mmand)
x <- c(0,0,1,0,0,0,1,1,1,0,0)
kernel <- c(1,1,1)
erode(x, kernel)
dilate(x, kernel)
```
Two dimensions

```r
library(png); library(mmand)
fan <- readPNG(system.file("images", "fan.png", package="mmand"))
display(fan)
```
Greyscale morphology in 2D

```
kernel <- shapeKernel(c(3, 3), type="diamond")
display(erode(fan, kernel))
```
Morphological gradient

```
kernel <- shapeKernel(c(3,3), type="diamond")
display(dilate(fan,kernel) - erode(fan,kernel))
```
Resampling

- **Indexing** between elements

  ```r
  x <- c(0,0,1,0,0)
x[2.5]
  [1] 0
  ```

- R truncates 2.5 to 2 and returns the second element

- In some cases there is conceptually a value at location 2.5 but we don’t know it

- Best guess is probably that it’s 0, or 1, something in between

- Using `mmand` we can *interpolate* using different sampling kernels

  ```r
  # "Nearest neighbour"
  resample(x, 2.5, boxKernel())
  [1] 1
  
  # Linear interpolation
  resample(x, 2.5, triangleKernel())
  [1] 0.5
  
  # Mitchell-Netravali cubic spline
  resample(x, 2.5, mitchellNetravaliKernel(1/3, 1/3))
  [1] 0.5347222
  ```

- An entire image of any dimensionality can be resampled similarly

- Allows *regridding, upsampling* and *downsampling*
Upsampling a smaller image

```r
fan_small <- readPNG(system.file("images", "fan-small.png", package="mmand"))
display(rescale(fan_small, 4, mnKernel()))
```
Image registration

- Aligning two related images
- Contrasts may be similar or different
- Pixel information may be combined
- Optimisation over a space of transformations (global/linear or local/nonlinear)
- Resampling to match the target image

Courtesy of Jiří Borovec, Czech Technical University, Prague
library(RNiftyReg)
source <- readNifti(system.file("extdata","epi_t2.nii.gz",package="RNiftyReg"))
target <- readNifti(system.file("extdata","mni_brain.nii.gz",package="RNiftyReg"))

linear <- niftyreg(source, target, scope="affine")
nonlinear <- niftyreg(source, target, scope="nonlinear", init=forward(linear))
Combining the packages: checking registration

```r
library(jpeg)
library(mmand)
library(RNiftyReg)

# Read images and convert to greyscale
source <- readJPEG("source.jpg")
target <- readJPEG("target.jpg")
source <- apply(source, 1:2, mean)
target <- apply(target, 1:2, mean)

# Register images
result <- niftyreg(source, target)

# Calculate morphological gradient
kernel <- shapeKernel(c(3,3), type="diamond")
gradient <- dilate(result$image, kernel) - erode(result$image, kernel)

# Display the results
display(target)
display(threshold(gradient, method="kmeans"), add=TRUE, col="red")
```
Combining the packages: checking registration
Other tools I have found useful

- The multiplatform **RStudio** IDE
- **car** for better AN(C)OVA
- **lme4** for random/mixed effects models
- **igraph** for graph theory
- **pcaMethods** for PCA (duh!)
- **png** and **jpeg** for working with image formats
- **R.matlab** for reading .mat files
- **soma** for nonlinear optimisation
- **ore** for text processing
- **TractoR**, of course!
- See also [http://www.statmethods.net](http://www.statmethods.net) for other applications
Why use R?

- Very strong on stats
- Scripted analyses for reproducibility; explicit tests
- Quick development due to high-level code
- Good performance in vectorised code
- Very easy to link in bits of C/C++/FORTRAN code for improved speed where needed
- Pretty, publication-ready graphics

- Free and open source: install it wherever you like
- Even modify it if you want!
- A strong platform for (image) data analysis
- Usage and awareness in imaging groups is growing (cf. Tabelow, Clayden et al., *NeuroImage*, 2011)