Comparative Methods in R

Brian C. O'Meara

May 19, 2015

Contents

Chapter 1: First steps

First, understand the question you want to answer. There are a wide variety of methods, and they wax and wane in popularity, but the key to doing good science is addressing compelling questions, not using the latest method. Once you have that question, find the appropriate methods (and, depending on how early it is in the study design, the right taxa and data) to address it. Understand how those methods work, including the various ways they can fail (as all can).

Many methods are now implemented in R: the phylogenetics task view has a brief overview. You can also install the relevant packages that are on CRAN and R-Forge using the task view itself:

```
install.packages("ctv")
library(ctv)
install.views("Phylogenetics")
```

Note that this will not install packages that are on github or authors' individual websites. The `devtools` package can be useful for installing packages directly from github.

An important resource for learning about phylogenetics in R is Emmanuel Paradis' book, Analysis of Phylogenetics and Evolution with R. This is written by the same person who is the lead author of the essential `ape` package for phylogenetics in R.

Chapter 2: Getting data and trees into R

Data and tree object types

In R, there are many kinds of objects. These kinds are called "classes". For example, take the text string "Darwin".

```
class("Darwin")
```

```
## [1] "character"
```

It is a `character` class. `pi` is a defined constant in R:

```
print(pi)
```

```
## [1] 3.141593
```

```
class(pi)
```

```
## [1] "numeric"
```

Its class is `numeric` [and note that its value is stored with more precision than is printed on screen].

Objects can sometimes be converted from one class to another, often using an `as.*` function:

```
example.1 <- "6"
print(example.1)
```

```
## [1] "6"
```

```
class(example.1)
```

```
## [1] "character"
```

```
example.2 <- as.numeric(example.1)
print(example.2)
```

```
## [1] 6
```

```
class(example.2)
```

```
## [1] "numeric"
```

```
example.2 * 7
```

```
## [1] 42
```

Trying to multiply example.1 by seven results in an error: you are trying to multiply a character string by a number, and R does not automatically convert classes. Classes have many uses in R; for example, one can write a different `plot()` function for each class, so that a tree is plotted one way, while a result from a regression model is plotted a different way, but users just have to call `plot()` on each and R knows what to do.

In phylogenetics, we mostly care about classes for trees, for data, and for things to hold trees and data.

Tree classes

The main tree class in R is `phylo` and is defined in the `ape` package. Let's look at one in the wild:

```
library(ape)
phy <- rcoal(5) #to make a random five taxon tree
print(phy)
```

```
##
## Phylogenetic tree with 5 tips and 4 internal nodes.
##
## Tip labels:
## [1] "t5" "t4" "t2" "t3" "t1"
##
## Rooted; includes branch lengths.
```

```
str(phy)
```

```
## List of 4
##  $ edge       : int [1:8, 1:2] 6 7 7 9 9 6 8 8 7 1 ...
##  $ edge.length: num [1:8] 1.0721 0.3127 0.2163 0.0964 0.0964 ...
##  $ tip.label  : chr [1:5] "t5" "t4" "t2" "t3" ...
##  $ Nnode      : int 4
##  - attr(*, "class")= chr "phylo"
##  - attr(*, "order")= chr "cladewise"
```

This is the one used in most packages. However, it has some technical disadvantages (sensitivity to internal structure, no checking of objects) that has led to the `phylo4` format for trees and `phylo4d` for trees plus data in the `phylobase` package. Other packages add on to the `phylobase` format (i.e., `phytool`'s `simmap` format) but these are typically not shared across packages.

Sequence data

Other character data

Phylogenies

The most common way to load trees is to use `ape`'s functions:

```
phy <- read.tree(file='treefile.phy')
```

To get a tree in Newick format (sometimes called Phylip format): essentially a series of parenthetical statements. An example (from `ape`'s documentation) is `((Strix_aluco:4.2,Asio_otus:4.2):3.1,Athene_noctua:7.3);`. The format name comes from the name of the lobster house where several major phylogenetic software developers met to agree on a tree format.

You can use the same function to enter tree strings directly, changing the argument from the `file` containing the tree to `text` containing the tree string:

```
phy <- read.tree(text = '((Strix_aluco:4.2,Asio_otus:4.2):3.1,Athene_noctua:7.3);')
```

Note the trailing semicolon.

One thing that can trip users up with `read.tree()` (and the `read.nexus()` function, below) is that the output class depends on the input. If you read from a file with one tree, the returned output is a single tree

object with class `phylo`. You can then use `plot()` on this object to draw the tree, pass this object into a comparative methods package to estimate rates, and so forth. If the file has more than one tree, the returned class is `multiphylo`: `plot()` will automatically cycle through plots as you type return, most comparative method implementations will fail (they are written to expect one tree of class `phylo`, not a vector of trees in a different class). `read.tree()` has an optional `keep.multi` function: if set to TRUE, the class is always `multiphylo`, and you can always get the first tree by getting the first element in the returned object:

```
phy.multi <- read.tree(file='treefile.phy', keep.multi = TRUE)
phy <- phy.multi[[1]]
```

For NEXUS formatted files (Maddison et al., 2007), `ape`'s `read.nexus()` function can pull in the trees (and its `read.nexus.data()` function can pull in data from a NEXUS file). NEXUS is a very flexible format, and there are valid NEXUS files that still cause errors with `ape`'s function. A more robust function to read in NEXUS trees is the package `phylobase`'s `readNexus()` function (note the lack of a period and different capitalization of Nexus from `ape`'s similar function). `phylobase` uses a different structure to store trees than `ape` does.

Reconciling datasets

Chapter 3: Visualizing data before use

A key step in any analysis is looking at the data. If you have loaded protein coding DNA sequences, are they aligned correctly? Are the codon positions specified correctly? For trait data, is everything measured in the same units, or are some oddly a thousand-fold higher than others? Are you dealing with an older dataset format that uses -1 or 19 for missing data, and have you incorrectly treated those as observations? Is your tree ultrametric?

It is easy to overlook this step, but you can draw the wrong conclusions based on errors at this stage. Most peer reviewers will not notice this, either, so your error could slip into the literature and mislead others. Take the time to get to know your data.