Introduction to Programming in R: phylogenetic trees

Erik Spence

SciNet HPC Consortium

26 April 2023

Today's slides

Today's slides can be found here. Go to the "Introduction to programming in R" page, under Lectures, "Phylogenetic trees".

https://scinet.courses/1277

Today's Lecture

Today's class will explore building phylogenetic trees in R.

- Introduce phylogenetic trees.
- Distance-based methods.
- Maximum parsimony methods.
- Likelihood-based methods.

We'll review many of the ways to build such trees.

So what are phylogenetic trees?

Phylogenetic trees are diagrams that show evolutionary relationships between organisms.

- They can be built of many different types of heritable traits,
  - morphological data,
  - structural features,
  - DNA or protien sequences.
- They are used to
  - understand biodiversity,
  - estimate common ancestors,
  - estimate times of divergence.
- There are many different styles of trees.
- Trees can be rooted or unrooted.

There are several different approaches to building such trees. We will use genetic data to build our trees.

Phylogenetic trees require distance

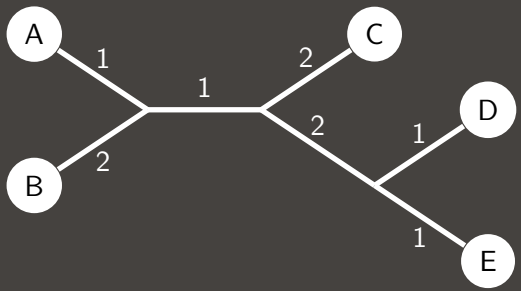
Almost all phylogenetic methods start with some sort of 'distance' metric.

- Why? Because we need some sort of measure of how different two sequences are from one another.
- The 'distance' is often just the sum of the discrepencies between nucleotides.
- These distances are often expressed as a distance matrix, containing the distance between each sequence.
- Methods based solely on distance are the simplest possible trees that you can build.
- More-sophisticated techniques combine distance with other criteria to refine the trees.

A distance-based method forms the starting point of all the tree-construction methods.

How do we perform phylogenetics?

	A	B	C	D	E
A	0	3	4	5	5
B	3	0	5	6	6
C	4	5	0	5	5
D	5	6	5	0	2
E	5	6	5	2	0



Families of trees

There are several major families of phylogenetic trees.

- Distance-based methods.
  - We use the distance matrix to cluster the species. The greater the distance the farther apart the species are in the tree.
  - Trees can be either rooted or unrooted, depending on the algorithm.
  - Can be inaccurate, since the model changes depending on the algorithm (the approach is not robust).
- Maximum parsimony methods.
  - These algorithms attempt minimize the number of character changes along the tree.
  - These trees tend to be simpler, with fewer branches.
- Likelihood-based methods.
  - The algorithms use Bayesian methods to maximize the likelihood of the model being correct, given the data ($P(M|D)$).
  - Tends to minimize the number of mutations in the tree.

Distance-based methods

There are two commonly encounted distance-based algorithms for building phylogenetic trees.

- The UPGMA (Unweighted Pair Group Method using Arithmetic mean):
  - Produces unrooted trees.
  - assumes a constant rate of evolution (nucleotide substitution).
- Neighbour joining:
  - Takes the two closest nodes of the tree and defines them as neighbours.
  - Produces unrooted trees.
  - Does not assume a constant rate of evolution.

All the trees we'll build today start with a neighbour-joining model. These are among the simplest trees that you can build.

Example data

Let us use the seasonal flu data 'adegenet' package. We will download it using the fasta2DNAbin function. The DNAbin format is a somewhat more efficient way to store sequence data.

Because influenza evolves rapidly we can build trees that show its evolution in a relatively short time frame.

```
>
> install.packages('adegenet')
>
> library(adegenet)
>
> flu.dna <- fasta2DNAbin(file = "http://adegenet.r-
forge.r-project.org/files/usflu.fasta")
>
> str(flu.dna)
'DNAbin' raw [1:80, 1:1701] a a a a ...
- attr(*, "dimnames")=List of 2
..$:  chr [1:80] "CY013200" "CY013781" "CY012128" ...
..$:  NULL
>
```

Example, annotations

We now download the annotations. Annotations consist of additional information about the sequence, such as identified genes, locations of protien sequences, and other information.

In this case the data merely consists of the sequence name, the year it was identified, and miscellaneous information.

```
>
> flu.annot <- read.csv("http://adegenet.r-forge.r-
project.org/files/usflu.annot.csv",
+         header = TRUE, row.names = 1)
>
> str(flu.annot)
'data.frame':  80 obs.  of 3 variables:
$accession:  chr "CY013200" "CY013781" "CY012128" ...
$year :  int 1993 1993 1993 1993 1993 1994 ...
$misc :  chr "(A/New York/783/1993(H3N2))" "(A/New
York/802/1993(H3N2))" "(A/New York/758/1993(H3N2))" ...
>
```

Distance-based trees, calculating distance

The first step to building a distance-based tree is to calculate the distances between the sequences.

We use the 'dist.dna' function, from the 'ape' package, to compute the distances.

There are 3160 unique distances between all the sequences.

```
>
> library(ape)
>
> flu.dist <- dist.dna(flu.dna)
>
> str(flu.dist)
'dist' num [1:3160] 0.00251 0.0082 0.00188 ...
- attr(*, "Size")= int 80
- attr(*, "Labels")= chr [1:80] "CY013200" ...
- attr(*, "Upper")= logi FALSE
- attr(*, "Diag")= logi FALSE
- attr(*, "call")= language dist.dna(x = dna)
- attr(*, "method")= chr "K80"
>
```

Distance-based tree, building the tree

The 'nj' function will build a neighbour-joining tree.

By default the tree will be 'unrooted'. If we want to root the tree we need to tell 'nj' where the root is. Let's pick one of the earliest years as the root. The 'root' function roots the tree.

The 'hclust' function does classical hierarchical clustering.

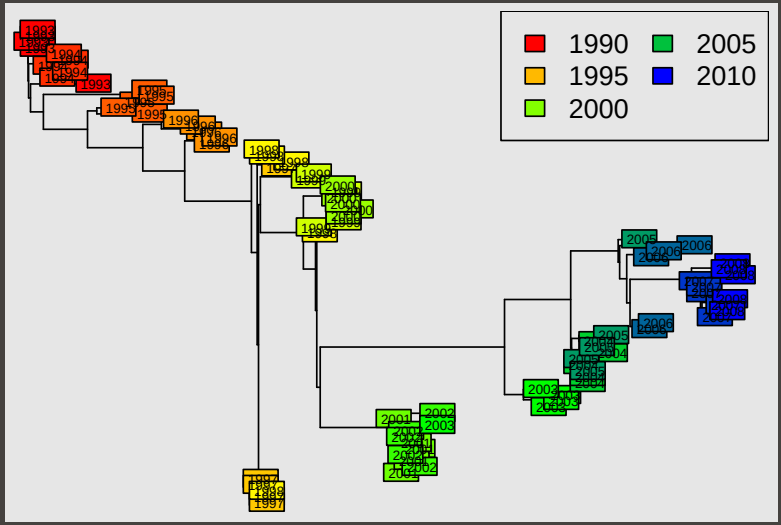
```
> nj.flu.tree <- nj(flu.dist)
>
> head(flu.annot)
     accession   year                              misc
 1    CY013200   1993   (A/New York/783/1993(H3N2))
 2    CY013781   1993   (A/New York/802/1993(H3N2))
 3    CY012128   1993   (A/New York/758/1993(H3N2))
 4    CY013613   1993   (A/New York/766/1993(H3N2))
 5    CY012160   1993   (A/New York/762/1993(H3N2))
 6    CY012272   1994   (A/New York/729/1994(H3N2))
>
> nj.flu.tree <- root(nj.flu.tree, out = 1)
>
```

Distance-based tree, plotting the tree

Rather than plotting the tree with the names of the sequences, instead we'll just plot it with the years.

```
>
> my.palette <- colorRampPalette(c("red", "yellow",
+                                  "green", "blue"))
>
> plot(nj.flu.tree, show.tip = FALSE)
>
> tiplabels(flu.annot$year, cex = 0.5,
+           bg = num2col(flu.annot$year,
+                        col.pal = my.palette))
>
> years <- pretty(1993:2008, 5)
>
> legend("topright", leg = years, ncol = 2,
+        fill = num2col(years, col.pal = my.palette))
>
```

Distance-based tree, plotting the tree, continued



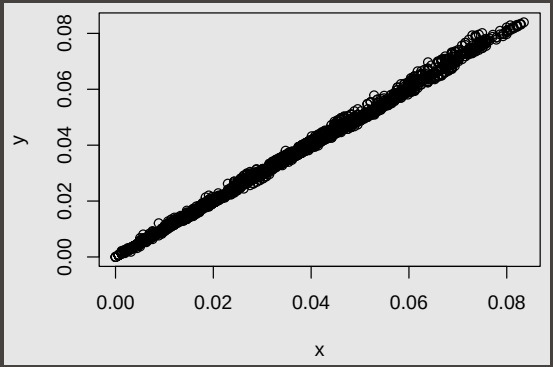
Analyzing the tree

But is this tree any good?

We can calculate the distance between the tips of the tree, using the 'cophenetic' function.

We can then compare this to the distance between the different sequences.

```
> x <- as.vector(flu.dist)
>
> y <- as.vector(as.dist(cophenetic(nj.flu.tree)))
>
> plot(x, y)
> cor(x, y)
[1] 0.9987531
```



There's a strong correlation. This is a good tree.

Maximum parsimony trees

What are maximum parsimony trees?

- parsimony: simplicity.
- This approach builds trees in an attempt to minimize the number of changes between tree branches.
- This requires building all possible trees, and then picking the tree with the fewest changes.
- This approach is also known as 'parsimony analysis'.

This family of approaches results in tree which have fewer branches.

Maximum parsimony trees, continued

We need the data to be in the 'phyDat' format, to use the phanghorn functionality.

To reconstruct the tree using maximum parsimony methods we need the tree to be in 'ape' format (which it already is).

```
>
> library(phangorn)
>
> max.pars.flu.dna <- as.phyDat(flu.dna)
>
> class(max.pars.flu.dna)
[1] "phyDat"
>
```

Maximum parsimony trees, continued more

We first calculate our previous tree's parsimony.

The function to optimize our existing tree, based on maximum parsimony is 'optim.parsimony'. In this case there is minimal improvement.

```
>
> parsimony(nj.flu.tree, max.pars.flu.dna)
[1] 422
>
> max.pars.flu.tree <- optim.parsimony(nj.flu.tree,
+                                      max.pars.flu.dna)
Final p-score 420 after 2 nni operations
>
> parsimony(max.pars.flu.tree, max.pars.flu.dna)
[1] 420
>
```

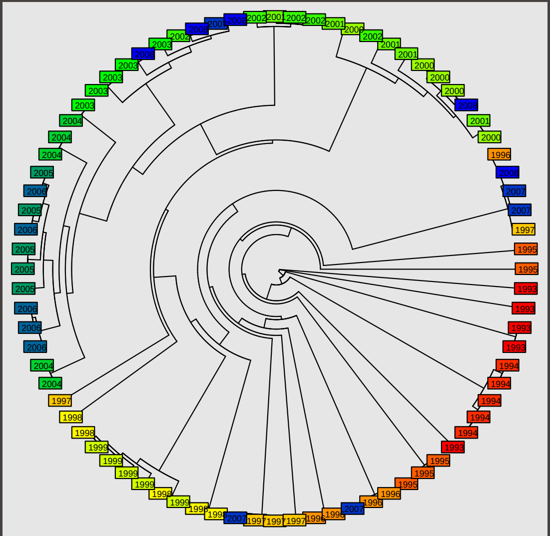
Maximum parsimony trees, plotting the tree

The commands to plot the tree are the same as before.

This time we specify a 'fan' type of tree. This makes the tree into a circle.

```
>
> plot(max.pars.flu.tree, show.tip = FALSE,
+       type = 'fan')
>
> tiplabels(flu.annot$year, cex = 0.5,
+           bg = num2col(flu.annot$year,
+                        col.pal = my.palette))
>
> legend("topright", leg = years, ncol = 2,
+        fill = num2col(years,
+                       col.pal = my.palette))
>
```

Maximum parsimony trees, plotting the tree, continued



Maximum likelihood-based trees

Maximum likelihood-based approaches are based on probability theory.

- Uses an outside model of how the evolution might be proceeding.
- For example, genetic mutations that change A to G are much more common than mutations from A to T or C.
- Maximum likelihood takes this restriction into account.
- The most-probable model is the model chosen.

These are considered one of the better techniques for building trees.

Maximum likelihood-based trees

We specify that the distances between the flu strains should be calculated using the model of Tamura and Nei (1993).

The 'pml' function calculates the likelihood, which in Bayesian terms means the probability of the model (in this case, the tree) given the data.

```
> max.like.tree <- nj(dist.dna(flu.dna,
+                              model = 'TN93'))
>
> pml(max.like.tree, max.pars.flu.dna)
model:  JC
loglikelihood:  -5687.342
unconstrained loglikelihood:  -4736.539

Rate matrix:
a c g t
a 0 1 1 1
c 1 0 1 1
g 1 1 0 1
t 1 1 1 0

Base frequencies:
a c g t
0.25 0.25 0.25 0.25
```

Maximum likelihood-based trees, continued

We can now optimize the tree. There are many options, choosing what to optimize:

- optNni: tree topology,
- optBf: base frequencies,
- optQ: substitution rates.

The goal, of course, is to maximize the likelihood.

```
>
> max.like.pml <- pml(max.like.tree,
+                     max.pars.flu.dna)
>
> fit.pml <- optim.pml(max.like.pml,
+                      optNni = TRUE, OptBf = TRUE,
+                      optQ = TRUE)
optimize edge weights:  -5687.342 --> -5636.745
optimize base frequencies:  -5636.745 --> -5592.613
optimize rate matrix:  -5592.613 --> -5397.233
optimize edge weights:  -5397.233 --> -5396.995
⋮
optimize edge weights:  -5385.428 --> -5385.428
optimize base frequencies:  -5385.428 --> -5385.428
optimize rate matrix:  -5385.428 --> -5385.428
optimize edge weights:  -5385.428 --> -5385.428
>
```

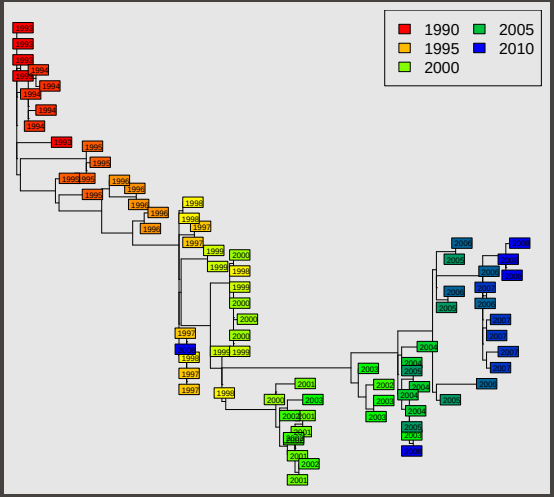
Maximum likelihood-based trees, continued more

Examining the values of log likelihood, we can see that the fit model is an improvement.

The goal, of course, is to maximize the likelihood.

```
> pml(max.like.tree, max.pars.flu.dna)$logLik
[1]  -5641.785
>
> fit.pml$logLik
[1] -5385.428
>
> pml.tree <- root(fit.pml$tree, 1)
>
> plot(max.pars.flu.tree, show.tip = FALSE,
+       type = 'tidy')
> tiplabels(flu.annot$year, cex = 0.5,
+           bg = num2col(flu.annot$year,
+                        col.pal = my.palette))
> legend("topright", leg = years, ncol = 2,
+        fill = num2col(years,
+                       col.pal = my.palette))
>
```

Maximum likelihood-based trees, plotting the tree



Summary

Today we covered several techniques you can use to build phylogenetic trees.

- A measure of 'distance' is needed to start building all trees.
- Simple distance-based measurements can build the simplest trees. We used neighbour-joining to start our trees today.
- Maximum parsimony trees build simplified versions of the distance-based trees.
- Maximum-likelihood methods use probabilistic methods to determine the most-probable tree, given the data.

And that's the class! Thanks to everybody!