Lab 5. Maximum likelihood

Due Wednesday, November 2, 2022 at 3 pm

This week, we will be using maximum likelihood to infer our pasta phylogenies. We will learn how to implement character ordering in the maximum likelihood setting, how to partition our dataset, and what the different choices associated with partitioning actually mean. We will also look into a problem that is specific to maximum likelihood, namely the need to correct for *ascertainment bias*.

Software

These analyses will be performed in IQ-TREE 2, an actively developed software package that is available for free on all platforms (Windows, macOS, Linux) and can be downloaded from `http://www.iqtree.org`. One of the neat things about IQ-TREE is its very extensive documentation, which is organized into easily digestible mini-tutorials that are further grouped by topic area: `http://www.iqtree.org/doc/`. It also comes with a single detailed manual, if you prefer to familiarize yourself with it that way: `http://www.iqtree.org/doc/iqtree-doc.pdf`.

Data

Throughout this handout, the Tedford et al. canid dataset will be used to demonstrate the features and functionality of IQ-TREE. You will then be asked to repeat the corresponding analyses on your own pasta data in order to answer the numbered questions in **bold font**.

Getting started

To run IQ-TREE in the simplest way possible, we only have to provide it with a single command-line argument, passed to the program via the `-s` flag: this is the filename of the dataset we want to analyze. For example:

```
./iqtree2 -s Tedford_2009-1.nex
```

If you are on Windows rather than macOS or Linux, you'll need to omit the "current directory" symbol (a dot followed by a forward slash) in front of the executable name:

```
iqtree2 -s Tedford_2009-1.nex
```

Note that this will only work if the IQ-TREE executable (called `iqtree2`) is located in the same folder as your dataset. If that's not the case, you will need to provide the absolute path to the executable, the dataset, or both. For example, if I navigate to the directory where my executable is located, but my dataset is somewhere else, I can specify its location as follows:

```
./iqtree2 -s /Users/David/Downloads/tedford/Tedford_2009-1.nex
```

Once again, if you are on a Windows machine, keep in mind that your absolute paths will look a bit different:

```
iqtree2 -s C:\Users\David\Downloads\tedford\Tedford_2009-1.nex
```

Or I can do it the other way around: go to the directory where my dataset is located, and specify the absolute path to my executable:

```
~/Grive/Slater_Lab/iqtree-2.1.3-MacOSX/bin/iqtree2 -s Tedford_2009-1.nex
```

The location of the output files is always determined by where you run the analysis from – if it is the directory where your executable is located, the files are going to end up there; if it is the directory containing your dataset, that is where your output is going to show up. Let's take a look at what this output consists of:

File	Description
`Tedford_2009-1.nex.bionj`	A starting tree for the heuristic search, generated using the "BioNJ" algorithm.
`Tedford_2009-1.nex.ckp.gz`	A compressed tree "checkpoint" file. This is similar to a saved-state file you might know from computer games: it stores the current tree and associated information so that an interrupted or unfinished run can be restarted from the last saved checkpoint.
`Tedford_2009-1.nex.iqtree`	A comprehensive summary of the analysis. It summarizes the dataset properties, tells you what substitution model proved to be the best fit to the data, provides the maximum-likelihood estimates of its parameters, and – most importantly – gives you the maximum-likelihood tree, both in a computer-readable format (as a Newick string) and as a drawing similar to those produced by PAUP*.
`Tedford_2009-1.nex.log`	A log file recording the command used to run the analysis, and everything that was printed to the console over the course of the run. This is very useful for making sure our analyses are reproducible.

`Tedford_2009-1.nex.mldist`	A file providing the maximum-likelihood estimates of pairwise distances among the taxa in your dataset. These distances are used to infer the BioNJ starting tree.
`Tedford_2009-1.nex.model.gz`	A checkpoint file similar to `.ckp.gz`, but for substitution model comparisons rather than the tree search.
`Tedford_2009-1.nex.parstree`	A maximum-parsimony tree used as a starting tree for the heuristic search.
`Tedford_2009-1.nex.treefile`	The maximum-likelihood tree in Newick format – the same one as in the `.iqtree` file, but given without any bells and whistles here. Use this file if you want to open your tree in FigTree.

If this seems a bit overwhelming, don't worry. Most of the time, you will only need the `.iqtree` file; you can safely ignore the rest.

Now that we've gained a better understanding of the output of our initial IQ-TREE run, let's try running a more sophisticated analysis:

```
./iqtree2 -s /Users/David/Downloads/tedford/Tedford_2009-1.nex -st MORPH -m
MK+G -pre my_ML_analysis -o Hesperocyoninae -b 1000 -T 4 --seed 12345
```

Let's take a look at what all these extra options mean:

Command-line option	**Meaning**
`-st MORPH`	Sequence type. This argument tells IQ-TREE what kind of data we will be using. It is similar to the `DATATYPE=` command we used in our Nexus files. We don't actually need it here; IQ-TREE is smart enough to figure out we gave it morphological data on its own.
`-m MK+G`	Substitution model. Here, the M*k* (Markov *k*-state) model of Lewis (2001) is used, with an additional "rate heterogeneity" model to account for the fact that different characters may evolve at different rates. Specifically, the `+G` flag tells IQ-TREE that we are willing to assume that these rates follow a gamma distribution (`https://en.wikipedia.org/wiki/Gamma_distribution`), and that we want this distribution to be approximated by 4 discrete categories.
`-pre my_ML_analysis`	What prefix we want the output files to have. By default, the full name of the dataset file is used.

`-o Hesperocyoninae`	Which taxon to select as the outgroup. By default, IQ-TREE uses the first taxon from our data file. In the canid dataset, that already happens to be Hesperocyoninae, so we could actually do without specifying this option.
`-b 1000`	Run 1000 boostrap replicates. IQ-TREE actually has two bootstrap options: the "ultrafast bootstrap approximation", which the developers are very proud of and which is activated using the upper-case `-B` flag, and the slower but more thorough classical bootstrap, activated using the lower-case `-b` flag. Our datasets are small enough that we can afford to stick with the slower option.
`-T 4`	Run the analysis on 4 processor threads. By default, IQ-TREE only uses one. You can also use the `-T AUTO` option, which detects the number of CPU cores available on your machine and uses as many of them as it sees fit.
`--seed 12345`	Seed used for generating (pseudo)random numbers. When not specified, it is automatically created by IQ-TREE based on your computer's clock. As a result, if you run the same analysis twice, you'll get slightly different results. To avoid this, you can seed your random number generator with a specific number, in which case you'll get *exactly* the same results every time.

The output files are slightly different this time around. Since we specified a particular model instead of letting IQ-TREE choose one for us, we no longer get any `.model.gz` file. However, we get a new file with the `.boottrees` extension, which stores all of our 1000 bootstrapped trees, and a `.contree` file with a consensus tree constructed out of all these 1000 replicates. Note, however, that the consensus of bootstrap trees is not the same thing as the maximum-likelihood tree, which is still found in the `.iqtree` and `.treefile` files. If you open them, you'll find that the maximum-likelihood tree is now annotated not just with branch lengths but also with bootstrap support values.

That's it! You're now ready to use IQ-TREE to perform a maximum-likelihood analysis on your pasta dataset.

---

1) Create a figure of your maximum-likelihood tree using either FigTree or R, and paste it into your lab report. Make sure the tip labels are legible, and provide the full command you used to run your IQ-TREE analysis.

2) How does your maximum-likelihood tree differ from the majority-rule consensus of most parsimonious trees that you generated in Lab 3 using PAUP*?

3) As a reminder, what do the branch lengths of your maximum-likelihood tree mean?

Correcting for ascertainment bias

In our lectures, we will talk about a problem called the *ascertainment* or *acquisition bias*, which didn't affect maximum parsimony, but which we have to account for in model-based methods such as maximum likelihood. Briefly, maximum likelihood assumes that our dataset contains a *random sample of characters* – specifically, random with respect to their informativeness about the phylogenetic relationships under consideration.

This assumption makes a great deal of sense when we are analyzing DNA sequences, for which maximum-likelihood phylogenetic inference was originally developed. With DNA, we may sequence a certain gene from different taxa and find that some characters are parsimony-informative (have at least two states, each of which is present in at least two taxa), some are autapomorphic (have a derived state in just one taxon), and many more are constant (have the same state in all taxa). Maximum parsimony only cares about the first category of characters, since autapomorphies and constant characters have the same length on all possible trees, and as a result, cannot discriminate between them. However, maximum likelihood needs all three of these types of characters to estimate a tree. This is because constant characters and autapomorphies inform our branch length estimates: if a taxon has a lot of autapomorphies, that tells us there is a high probability of change along the branch subtending that taxon. That, in turn, means that we'll be less likely to interpret the character states shared between this taxon and other taxa as synapomorphies, and more likely to interpret them as homoplasies.

Unfortunately, researchers who assemble morphological datasets never treat them as random samples of characters. On the contrary, they specifically search for characters that are informative about the relationships among the taxa they want to analyze, just like you did during the pasta lab. Autapomorphies are occasionally included, but constant characters almost never are. Moreover, even if you decided to come up with a few – e.g., "feathers: no (0), yes (1)" would be a good constant character for a phylogenetic analysis of birds – the real problem is to make sure you've included just the right number of them for your characters to approximate a random sample. Since no one has figured out how to do that, the only solution we are left with is to give up on sampling constant characters altogether and correct our analyses for their absence. Fortunately, Paul Lewis introduced such a correction in the very first paper that showed how to infer phylogenies from morphological data using maximum likelihood (*Systematic Biology*, 50(6): 913–925, 2001). You can get IQ-TREE to implement Lewis's correction by tacking `+ASC` onto whatever model you are using: e.g., instead of typing `-m MK+G` as on page 3, we could specify `-m MK+G+ASC`.

---

4) Refer to lectures (once available) or to Lewis's 2001 paper: in what ways would we expect our analyses to go wrong if we *don't* correct for ascertainment bias?

5) Re-run your analysis with the ascertainment bias correction. Did it change anything? Make sure to consider not just the topology of your tree but also its branch lengths.

Modeling multistate and ordered characters by partitioning

Until now, we have assumed that a single substitution model – be it `MK+G` or `MK+G+ASC` – can be applied to the whole character matrix. However, that can't be right. In fact, the very name of Lewis's morphological model – "Markov k-state" – hints at the fact that the model for binary characters, where $k = 2$, is going to be quite different from the model for five-state characters, where $k = 5$. The former model can be described by a 2-by-2 rate matrix; the latter model needs a 5-by-5 matrix. Based on its name, you might perhaps expect IQ-TREE to be smart enough to figure out which character requires which matrix.

Unfortunately, this is not so. What IQ-TREE actually does is take the highest-numbered state in the dataset (for the Tedford et al. canid data, that would be state 5), construct the appropriate rate matrix (since there are six character states ranging from 0 to 5, it will be a 6-by-6 matrix), and mindlessly apply it to every character in the dataset. If it sees a character for which every taxon is coded either "0" or "1", it will still treat it as a 6-state character; it will just assume that states 2, 3, 4, and 5 happened not to be observed in the taxa under consideration. Since the Mk model has a uniform stationary distribution of character states (which is just a mathy way of saying that after a long enough time, all character states are equally likely to be observed), this outcome is extremely improbable, and as a result, our tree likelihoods will sink like a stone.

But wait, it gets worse! We know that some multistate characters need to be ordered. While IQ-TREE can handle this, it doesn't like us to mix ordered and unordered characters the way we were able to do it in PAUP*.

Fortunately, there is a way out of this quandary, and that is *partitioning*. Partitioning refers to breaking up your dataset into chunks, or "partitions", in such a way that all characters in the same partition can be accurately described using the same substitution model. In most cases, it's not necessary to literally break up your data file into multiple smaller files; you just provide the program with an extra file that specifies which characters are supposed to go into which partition. But we're all out of luck today, since for morphological data, IQ-TREE actually does require us to generate a separate data file for each partition. To make this step as painless as possible, I wrote an R script called `partition.R` and made it available on Canvas. You provide it with (1) the absolute path to your Nexus file and (2) a list of characters to be ordered, and it will create a separate Phylip file for each partition. Here is how you run it:

```
Rscript partition.R -p "/Users/David/Downloads/Tedford_2009-1.nex" -o "14,
23, 32, 38, 41, 57"
```

Or on Windows:

```
Rscript.exe partition.R -p "C:\\Users\\David\\Downloads\\Tedford_2009-1.nex"
-o "14, 23, 32, 38, 41, 57"
```

Please note that while I tried to make this script as general as possible, it is ridiculously easy to make it come crashing down. The path argument, specified using the `-p` flag, has to be enclosed in quotation marks, and if you're on Windows, you have to use double backslashes to make it parsable. The indices of ordered characters, specified using the `-o` flag, have to be separated by commas and enclosed in quotation marks as well.

After I've run the script using the command above, I now have the following files in the directory where I stored the original dataset:

- `Tedford_2009-1_MK2.phy`
- `Tedford_2009-1_MK3.phy`
- `Tedford_2009-1_MK4.phy`
- `Tedford_2009-1_MK6.phy`
- `Tedford_2009-1_ORDERED3.phy`

The next step, unfortunately, cannot be automated quite so easily: you will have to create your own plain-text file that assigns these partitions appropriate substitution models. For me, that file – which I'm going to uncreatively call `partitions.nex` – looks like this:

```
#nexus
begin sets;
   charset part1 = Tedford_2009-1_MK2.phy;
   charset part2 = Tedford_2009-1_MK3.phy;
   charset part3 = Tedford_2009-1_MK4.phy;
   charset part4 = Tedford_2009-1_MK6.phy;
   charset part5 = Tedford_2009-1_ORDERED3.phy;
   charpartition phylogeneticsrules =
      JC2+ASC+G: part1,
      MK+ASC+G: part2,
      MK+ASC+G: part3,
      MK+ASC: part4,
      ORDERED+ASC+G: part5;
end;
```

First, I assigned the `JC2` model to binary characters, the `MK` model to unordered multistate characters, and the `ORDERED` model to ordered multistate characters. Then, I tacked the ascertainment bias correction (`+ASC`) onto all of these models, since none of my partitions contained any constant characters. Finally, I added the discrete-gamma rate heterogeneity model (`+G`) to all partitions except the 6-state one (`part4`). My reasoning was that the 6-state partition turned out to contain just a single character, so by definition, there wasn't any among-character rate variation to account for.

Since the `partitions.nex` file already fully specifies the data files, data types, and substitution models to be used in the analysis, I can simplify my IQ-TREE command a little bit:

```
./iqtree2 -p /Users/David/Downloads/partitions.nex -pre canids_partitioned
-o Hesperocyoninae -b 1000 -T AUTO
```

Now we can apply what we've learned:

---

6) What partitions did you get by running the `partition.R` script on your pasta data?

7) What models did you assign to them? Justify your choices.

8) Run a partitioned analysis on your pasta dataset. Compare the results to those from questions 1, 2, and 5.

Partitioning, continued: analyzing real data

For the last few steps in today's exercise, we may need something a bit more substantive than the pasta data we've been playing with so far:

---

9) Run a partitioned maximum-likelihood analysis either on the dataset you will be using for your project, or the dataset you found for Lab 1. What settings did you use? Provide the full IQ-TREE command and justify your choices.

10) Describe the tree – is it different from the tree shown in the original paper? If so, how?

11) Create a figure of your maximum-likelihood tree using either FigTree or R, and paste it into your lab report. Make sure the tip labels and bootstrap support values are legible.

One last thing to notice: when passing the partitioning file to IQ-TREE, we used the `-p` flag. This means we wanted the branch lengths of each partition to be *proportional*: the length of the branch subtending taxon x doesn't have to be the same in partition 1 and partition 2, but if its branch is twice as long as that of taxon y in partition 1, it will also have to be twice as long in partition 2. If we think of this in terms of the number of branch lengths we have to estimate, we find that the number is equal to $(2n-3)+(k-1)$, where n is the number of taxa and k is the number of partitions: $(2n-3)$ is simply the number of branches in an unrooted n-taxon tree, and every additional partition that we add gets a "multiplier" that scales its branch lengths relative to the first partition.

However, we could also require that the length of each branch be exactly the same for every partition. In this case, we could do with just $(2n-3)$ branch lengths. We can make IQ-TREE do this using the following command – note that we use `-q` instead of `-p`:

```
./iqtree2 -q /Users/David/Downloads/partitions.nex -pre equal_brlens -o
Hesperocyoninae -b 1000 -T AUTO
```

Finally, the opposite extreme would be to give each partition its own independent set of branch lengths. This would allow us to deal with some scenarios that the previous two approaches were too inflexible to accommodate: e.g., x could have a longer branch than y in one partition, and y could have a longer branch than x in another partition. However, in terms of the number of branch lengths to estimate, this approach is very costly: we need $(2n - 3) \times k$ of them. If we (perhaps unwisely) decide to go ahead with it anyway, we need to preface the name of our partition-specifying file with the `-Q` flag:

```
./iqtree2 -Q /Users/David/Downloads/partitions.nex -pre independent_brlens
-o Hesperocyoninae -b 1000 -T AUTO
```

---

12) In real-world analyses, we almost always use proportional branch lengths, which represent a nice middle ground between biological realism (since we shouldn't expect all partitions to evolve at exactly the same rate) and model complexity (since having too many branch lengths in the model makes it hard to estimate them all with accuracy). However, it can be interesting to explore the impact of these choices. Re-run the analysis from questions 9–11 while treating branch lengths as equal (`-q`) or completely independent (`-Q`) across partitions. Did this make any difference?
