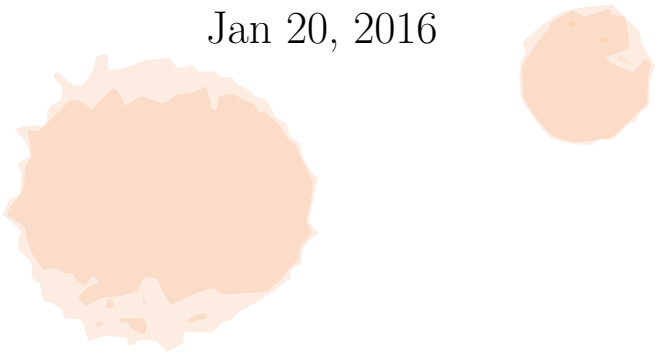
CAFE: Computational Analysis of gene Family Evolution

Tutorial

Jan 20, 2016



Contents

1 This tutorial

This document contains a tutorial[1] that should help you get started with CAFE. The tutorial is divided into two parts:

1. ***Preparing an input dataset that CAFE understands***: this is most of the work, and makes use of auxiliary Python scripts (which we provide) and a few other programs;

2. ***Running CAFE***: performing basic evolutionary inferences about gene family evolution.

1.1 Dependencies

The tutorial assumes you are running a *Unix-based operating system*. It also assumes you have a *local working version of CAFE* (please see CAFE's manual for instructions on how to install it), but also of a few other programs that are necessary for the first part of the tutorial:

- Python 2.7.x;
- BLAST;
- mcl;
- r8s.

1.2 Commands

Copying and pasting commands from this `.pdf` document can add whitespaces and quotes that might lead to errors. So please feel free to copy and paste from the file we provide, `all_cafe_commands.txt`.

[1] If you have any comments or suggestions, please email `fkmendes@indiana.edu`.

2 Preparing the input

2.1 Downloading the data

This tutorial and the scripts we provide assume you will use sequences in FASTA format (`.fa`) downloaded from Ensembl using the Biomart tool. To download the protein sequences from, say, cat (*Felis catus*), you must navigate Biomart:

1. *CHOOSE DATABASE* → *Ensembl Genes 87* → *CHOOSE DATASET* → *Cat genes*;
2. Then click *Attributes* → *Sequences: Peptide* → *Header information: Gene ID + CDS length* (uncheck *Transcript ID*);
3. Finally, click *Results*. If you have a good internet connection, choose *Compressed file (.gz)*, otherwise choose *Compressed web file* and provide your email address.

We are going to analyze data from 12 species: mouse, rat, cow, horse, cat, marmoset, macaque, gibbon, baboon, orangutan, chimpanzee, and human. It should not take too long to download the 12 files, but we are providing you with a tarball (`twelve_spp_proteins.tar.gz`) containing all of them.

In order to decompress `twelve_spp_proteins.tar.gz` (it will create its own folder), open your shell, move to the folder into which you downloaded the file, and enter:

```
$ tar -zxvf twelve_spp_proteins.tar.gz
$ for i in `ls -1 twelve_spp_proteins/*.tar.gz`; do tar -zxvf $i -C
    twelve_spp_proteins/; done
```

2.2 Identifying gene families

Identifying gene families within and among species requires a few steps. First, we need to deal with alternative splicing and redundant gene entries by removing all but the

longest isoform for each gene. After this is done for all 12 species, we shall move all sequences to a single file and prepare a database for BLAST. BLAST will allow us to find the most similar sequence for each sequence in the database (all-by-all blastp). Then we employ a clustering program, mcl, to find groups of sequences (gene families) that are more similar among themselves than with the rest of the dataset. Finally, we parse mcl's output to use as input for CAFE.

2.2.1 Moving all longest isoforms into a single file

In order to keep all but the longest isoforms, and place all sequences from all species into a single `.fa` file for the next tutorial step, run the following commands on your shell from the tutorial folder:

```
$ python python_scripts/cafetutorial_longest_iso.py -d
   twelve_spp_proteins/
$ cat twelve_spp_proteins/longest*.fa > makeblastdb_input.fa
```

2.2.2 All-by-all BLAST

With `makeblastdb_input.fa` in hand, we can now prepare a database for BLAST, and then run `blastp` on it, all sequences against all sequences. This step gives us, for each sequence, the most similar sequence (in addition to itself) in the dataset. We can then find clusters of similar sequences from these similarity scores (see next step). To prepare the database, run the following command on your shell:

```
$ makeblastdb -in makeblastdb_input.fa -dbtype prot -out blastdb
```

This command should create a few files. From here, we can actually run `blastp` (on, say, four threads) with:

```
$ blastp -num_threads 4 -db blast.db -query makeblastdb_input.fa
-outfmt 7 -seg yes > blast_output.txt
```

The `-seg` parameter filters low complexity regions (amino acids coded as X) from sequences. Because this job can take many hours, we provide you with its output (a tarball named `blast_output.tar.gz`) so you can keep on doing the tutorial.

2.2.3 Clustering sequences with mcl

Now we must use the output of BLAST to find clusters of similar sequences. These clusters will essentially be the gene families we will analyse with CAFE. Clustering is done with a program called mcl, running a few commands:

```
$ grep -v "#" blast_output.txt | cut -f 1,2,11 > blast_output.abc
```

In the command above, we convert the blast output into ABC format, which mcl understands. Then with the following commands, we have mcl create a network and a dictionary file (`.mci` and `.tab`, respectively), and perform the clustering:

```
$ mcxload -abc blast_output.abc --stream-mirror --stream-neg-log10 -
    stream-tf 'ceil(200)' -o blast_output.mci -write-tab blast_output.
    tab
$ mcl blast_output.mci -I 3
$ mcxdump -icl out.blast_output.mci.I30 -tabr blast_output.tab -o dump.
    blast_output.mci.I30
```

The `-I` (inflation) parameter determines how granular the clustering will be. Lower numbers means denser clusters, but again, this is an arbitrary choice. A value of 3 usually works, but you can try different values and compare results. Ideally, one wants to be able maximize the number of clusters containing the same number of genes as there are species, as these are likely to represent correct one-to-one ortholog identifications (in our case, we want clusters with 12 species). Furthermore, we want to minimize the number of clusters with just a single sequence.

2.2.4 Final parsing of mcl's output

The file obtained in the last section (the dump file from mcl) is still not ready to be read by CAFE: we need to parse it and filter it. Parsing is quite simple and just involves tabulating the number of gene copies found in each species for each gene family. We provide a script that does it for you. From the directory where the dump file was written, run the following command:

```
$ python python_scripts/cafetutorial_mcl2rawcafe.py -i dump.
    blast_output.mci.I30 -o unfiltered_cafe_input.txt -sp "ENSG00
```

```
ENSPTR ENSPPY ENSPAN ENSNLE ENSMMU ENSCJA ENSRNO ENSMUS ENSFCA
ENSECA ENSBTA"
```

Then there is one final filtering step we must perform. Gene families that have large gene copy number variance can cause parameter estimates to be non-informative. You can remove gene families with large variance from your dataset, but we found that putting aside the gene families in which one or more species have ≥ 100 gene copies does the trick. You can do this filtering step with another script we provide:

```
$ python python_scripts/cafetutorial_clade_and_size_filter.py -i
  unfiltered_cafe_input.txt -o filtered_cafe_input.txt -s
```

As you will see, the script will have created two files: `filtered_cafe_input.txt` and `large_filtered_cafe_input.txt`. The latter contains gene families where one or more species had ≥ 100 gene copies. We can now run CAFE on `filtered_cafe_input.txt`, and use the estimated parameter values to analyse the large gene families that were set apart in `large_filtered_cafe_input.txt`.

For the sake of clarity, we renamed the species ID with the corresponding informal species names, so "ENSG00" would read "human", "ENSPTR" "chimp"", and so on. So be sure you do:

```
$ mv filtered_cafe_input_names.txt filtered_cafe_input.txt
$ mv large_filtered_cafe_input_names.txt large_filtered_cafe_input.txt
```

2.3 Estimating a species tree

Estimating a species tree takes a number of steps. If genome data is available for all species of interest, one will need sequence alignments (with one sequence per species, from hopefully many genes) and then choose one among the many available species-tree estimation methods. Obtaining alignments usually requires finding one-to-one ortholog clusters with mcl (see previous section), but other procedures exist. Alternatively, pre-aligned one-to-one ortholog data from Ensembl or UCSC Genome Browser can sometimes be found and readily used.

With alignments in hand, one could concatenate all alignments and infer a non-ultrametric species tree with a maximum-likelihood (e.g., RAxML or PhyML) or Bayesian

phylogenetic program (e.g., MrBayes). Alternatively, coalescent-based methods can be used (e.g., fast methods such as MP-EST and Astral-II, or full coalescent methods such as BPP and *BEAST).

The species tree from this tutorial should not be problematic, but estimating it from genome scale data would take time that we do not have. So we provide you with a maximum-likelihood tree from concatenated data (in NEWICK format) below:

```
(((cow:0.09289,(cat:0.07151,horse:0.05727):0.00398):0.02355,((((orang
   :0.01034,(chimp:0.00440,human:0.00396):0.00587):0.00184,gibbon
   :0.01331):0.00573,(macaque:0.00443,baboon:0.00422):0.01431)
   :0.01097,marmoset:0.03886):0.04239):0.03383,(rat:0.04110,mouse
   :0.03854):0.10918);
```

We must now make this tree ultrametric, which can be done using the program r8s.

2.3.1 Making the species tree ultrametric

There are many ways to obtain ultrametric trees (also known as timetrees, these are phylogenetic trees scaled to time, where all paths from root to tips have the same length). Here, we use a fast program called r8s. You will need to know the number of sites in the alignment used to estimate the species tree (the one you want to make ultrametric), and then you can specify one or more calibration points (ideally, the age or age window of a documented fossil) to scale branch lengths into time units. We provide you with a script that prepares the control file for running r8s on the species tree above (the number of sites is 35157236, and the calibration point for cats and humans is 94). In your shell, type:

```
$ python python_scripts/cafetutorial_prep_r8s.py -i
   twelve_spp_raxml_cat_tree_midpoint_rooted.txt -o r8s_ctl_file.txt -
   s 35157236 -p 'human,cat' -c '94'
```

Then you can finally run r8s, and parse its output with:

```
$ r8s -b -f r8s_ctl_file.txt > r8s_tmp.txt
$ tail -n 1 r8s_tmp.txt | cut -c 16- > twelve_spp_r8s_ultrametric.txt
```

3 Running CAFE

In order to run CAFE analyses, you can either enter the desired commands directly into CAFE's shell (that is, if you type `cafe` on your Terminal and then start entering commands after '`#`'), or you can list the commands in a shell (`.sh`) script and then just run `cafe shell_script.sh`.

Some of the steps in this tutorial can take a while to finish, so we provide you with all the outputs – we will inform you of which analyses take longer, so you do not accidentally overwrite the output files we provide. Please be sure you finish reading a section before executing commands. Finally, all commands we list below should be run from the tutorial folder.

3.1 Estimating the birth-death parameter λ

The main goal of CAFE is to estimate one or more birth-death (λ) parameters for the provided tree and gene family counts. The λ parameter describes the probability that any gene will be gained or lost.

3.1.1 Estimating a single λ for the whole tree

Estimating λ can be achieved if one types the following commands on CAFE's shell:

```
# #!cafe
# load -i filtered_cafe_input.txt -t 4 -l reports/log_run1.txt
# tree ((((cat:68.710507,horse:68.710507):4.566782,cow:73.277289)
    :20.722711,(((((chimp:4.444172,human:4.444172):6.682678,orang
    :11.126850):2.285855,gibbon:13.412706):7.211527,(macaque:4.567240,
    baboon:4.567240):16.056992):16.060702,marmoset:36.684935)
    :57.315065):38.738021,(rat:36.302445,mouse:36.302445):96.435575)
# lambda -s -t ((((1,1)1,1)1,(((((1,1)1,1)1,1)1,1)1,(1,1)1)1,1)1)1,(1,1)1)
# report reports/report_run1
```

We provide you with file `cafetutorial_run1.sh` so you do not have to type all those commands. Simply run CAFE with:

```
$ cafe cafe_shell_scripts/cafetutorial_run1.sh
```

Understanding the output

After CAFE finishes estimating λ, you will want open file `reports/report_run1.cafe`. It should look like this (some numbers might differ, of course):

```
Tree:((((cat:68.7105,horse:68.7105):4.56678,cow:73.2773) (...)  :96.4356)
((((cat:68.7105,horse:68.7105):4.56678,cow:73.2773) (...)
Lambda:  0.00265952
Lambda tree:  ((((1,1)1,1)1,(((((1,1)1,1)1,1)1,(1,1)1)1,1)1)1,(1,1)1)
# IDs of nodes:((((cat<0>,horse<2>)<1>,cow<4>) (...)  (rat<20>,mouse<22>)<21>)<19>
# Output format for:  (...)  = (node ID, node ID): (0,2) (1,4) (...)
(...)
'ID' 'Newick' 'Family-wide P-value' 'Viterbi P-values'
8 ((((cat_59:68.7105 (...)  mouse_61:36.3024)_62:96.4356)_62 0.438 ((-,-),(-,-) (...)
10 ((((cat_57:68.7105 (...)  mouse_52:36.3024)_53:96.4356)_55 0.916 ((-,-),(-,-) (...)
11 ((((cat_37:68.7105 (...)  mouse_79:36.3024)_69:96.4356)_52 0 ((0.0699637,0.487686),(0.758569,0.202491)
(...)
```

Some of this output is self-explanatory, but let us point out what is important:

- On the second line, you will find the estimated value of λ for the whole tree, which for this run of CAFE was `0.00265952`.
- The line that starts with '`# IDs of nodes`' gives us the number that will represent each species and internal node. So for this run of CAFE, for example, '`0`' represents cat, '`2`' represents horse, and so on.
- The line that starts with '`# Output format for:`' shows the pairs of species or internal branches for which results will be presented later. Whenever you see pairs of values enclosed in parenthesis, one after the other, they shall follow the order shown on this line. In the example above, the first pair of values enclosed in parenthesis you see will refer to cat and horse ('`(0,2)`'), the second pair of values will refer to the internal branch subtending (cat,horse) and then cow ('`(1,4)`'), and so on.

- Then finally you have the results for each gene family, one gene family per line. In our example above, we are showing the first three gene families, identified by their numbers (the first column, '`ID`'), 8, 10 and 11 (see `filtered_cafe_input.txt`). The number that appears after a species name in the tree given under '`Newick`' (e.g., 59 in '`cat_59`' from gene family 8) is the gene count for that species and that gene family. The third column ('`Family-wide P-value`') tells us for each gene family whether it has a significantly greater rate of evolution. When this value is < 0.01, then the fourth column ('`Viterbi P-value`') allows the identification of which branches the shift in λ was significant. Because the family-wide p-value of gene families 8 and 10 were not significant in our example, then no results are presented under the Viterbi p-value. However, gene family 11 had a significant family-wide p-value (0), and so we can now identify which branches underwent significant contractions or expansions. For this CAFE run, branches 0, 1, 2 and 4 have not undergone significant shifts in λ, but branch 8 (leading to humans) have (not shown above, but see `report_run4.cafe`).

Summarizing the output

We provide a convenient script that summarizes the output described above into tables. In order to run it, just enter the following command on your shell:

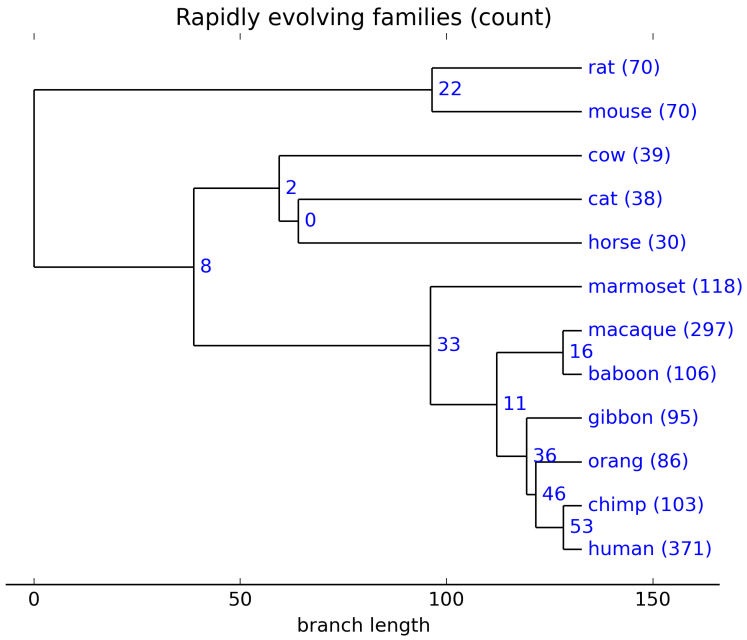
```
$ python python_scripts/cafetutorial_report_analysis.py -i reports/
    report_run1.cafe -o reports/summary_run1
```

This script should produce a few files that will help you visualize the results. For example, if you open file `reports/summary_run1_node.txt`, you will see, for each branch, how many families underwent expansions, contractions, and how many are rapidly evolving. In fact, we provide yet another script that allows you to plot these numbers on a phylogenetic tree. Just run the following command:

```
$ python python_scripts/cafetutorial_draw_tree.py -i reports/
    summary_run1_node.txt -t '((((cat:68.7105,horse:68.7105):4.56678,
    cow:73.2773):20.7227,(((((chimp:4.44417,human:4.44417):6.68268,
    orang:11.1268):2.28586,gibbon:13.4127):7.21153,(macaque:4.56724,
    baboon:4.56724):16.057):16.0607,marmoset:36.6849):57.3151):38.738,(
    rat:36.3024,mouse:36.3024):96.4356)' -d '((((cat<0>,horse<2>)<1>,
    cow<4>)<3>,(((((chimp<6>,human<8>)<7>,orang<10>)<9>,gibbon<12>)
```

```
<11>,(macaque<14>,baboon<16>)<15>)<13>,marmoset<18>)<17>)<5>,(rat
<20>,mouse<22>)<21>)<19>' -o reports/summary_run1_tree_rapid.png -y
 Rapid
```

You can then find the tree in the `reports/summary_run1_tree_rapid.png` file that should have been created.



We can see that the internal branch with the largest numbers of rapidly evolving gene families corresponds to the most recent common ancestor of humans and chimpanzees. The terminal branch with the most rapidly evolving gene families is the one leading to humans. Then if you wish to look at the number of gene families expanding or contracting (but not necessarily with statistical significance), replace '`Rapid`' with '`Expansions`' or '`Contractions`', and rename the output file names accordingly.

Finally, you might also be interested in having a look at `reports/summary_run1_fams.txt`, which will show how many **rapidly evolving** families (and which families) were found for each species and internal branch.

3.1.2 Setting λ to a previously estimated value to deal with families with large numbers of gene copies

As described in section 2.2.4, families with high variance in gene copy number can lead to non-informative parameter estimates, so we had to set them aside. We can now analyse them with the λ estimate obtained from the other gene families by running CAFE with:

```
# #!cafe
# load -i large_filtered_cafe_input.txt -t 4 -l reports/log_run2.txt
# tree ((((cat:68.710507,horse:68.710507):4.566782,cow:73.277289)
   :20.722711,(((((chimp:4.444172,human:4.444172):6.682678,orang
   :11.126850):2.285855,gibbon:13.412706):7.211527,(macaque:4.567240,
   baboon:4.567240):16.056992):16.060702,marmoset:36.684935)
   :57.315065):38.738021,(rat:36.302445,mouse:36.302445):96.435575)
# lambda -l 0.00265952 -t ((((1,1)1,1)1,(((((1,1)1,1)1,1)1,(1,1)1)1,1)
   1)1,(1,1)1)
# report reports/report_run2
```

For the sake of completion, we provide you with a file (`cafe_shell_scripts/cafetutorial_run2.sh`) containing all these commands. However, running this analysis can take a long time – so we suggest you just have a quick look on the output that we provide (`reports/report_run2.cafe`).

3.1.3 Estimating multiple λ for different parts of the tree

If you suspect different species or clades have different rates of gene family evolution, you can ask CAFE to estimate them. In this case, you must tell CAFE how many different λs there are, and which species or clades share these different λs. This can be done by entering the following commands on CAFE's shell:

```
# #!cafe
# load -i filtered_cafe_input.txt -t 4 -l reports/log_run3.txt
# tree ((((cat:68.710507,horse:68.710507):4.566782,cow:73.277289)
   :20.722711,(((((chimp:4.444172,human:4.444172):6.682678,orang
   :11.126850):2.285855,gibbon:13.412706):7.211527,(macaque:4.567240,
   baboon:4.567240):16.056992):16.060702,marmoset:36.684935)
   :57.315065):38.738021,(rat:36.302445,mouse:36.302445):96.435575)
# lambda -s -t ((((3,3)3,3)3,(((((1,1)1,2)2,2)2,(2,2)2)2,3)3)3,(3,3)3)
```

```
# report reports/report_run3
```

or, alternatively, you can call CAFE with:

```
$ cafe cafe_shell_scripts/cafetutorial_run3.sh
```

The most important line here is the argument for the `-t` parameter, '`((((3,3)3,3)3,(((((1,1)1,2)2,2)2,(2,2)2)2,3)3)3,(3,3)3)`'. This tree structure specifies which species are to share the same λ values. In our example, humans, chimpanzees and their immediate ancestor share λ_1; then all the remaining primates (except for marmoset) share λ_2; and finally marmoset and the other species share the λ_3 value.

After CAFE finishes running, you should have obtained values somewhat similar to these: $\lambda_1 = 0.0182972$, $\lambda_2 = 0.00634377$ and $\lambda_3 = 0.00140705$ (see `reports/report_run3.cafe`). This tells us that the lineage leading to (and including) humans and chimpanzees have higher gene family evolution rates, followed by the remaining primates (except for marmosets), and then by the remaining species.

3.2 Comparing models with one *vs* multiple λ

Which model best describes the data: the first one in which the whole tree shares the same global λ, or the second one, with three distinct λs?

CAFE can compare a global-λ model and one with multiple-λs by performing a likelihood ratio test. In order to do this test, you must first have CAFE simulate many datasets with a global λ. For each of these simulations, one can then estimate the likelihoods of a global-λ and of a multi-λ models, and obtain their likelihood ratio (more precisely, twice the difference of their log-likelihoods, $2 \times (lnL_{global} - lnL_{multi})$). Finally, the likelihood ratios from all simulations will comprise a null distribution that can be used to compare the observed (real) likelihood ratio against. For example, if 100 simulations are performed, the p-value will be the number of simulated likelihood ratios that are lower than the observed one, divided by 100.

One can do these steps by entering these commands in the CAFE shell:

```
# # !cafe
# load -i filtered_cafe_input.txt -t 4 -l reports/log_run4.txt
# tree ((((cat:68.710507,horse:68.710507):4.566782,cow:73.277289)
   :20.722711,(((((chimp:4.444172,human:4.444172):6.682678,orang
```

```
    :11.126850):2.285855,gibbon:13.412706):7.211527,(macaque:4.567240,
    baboon:4.567240):16.056992):16.060702,marmos et:36.684935)
    :57.315065):38.738021,(rat:36.302445,mouse:36.302445):96.435575)
# lambda -l 0.00265952
# genfamily tutorial_genfamily/rnd -t 100
# lhtest -d tutorial_genfamily -t ((((3,3)3,3)3,(((((1,1)1,2)2,2)
    2,(2,2)2)2,3)3)3,(3,3)3) -l 0.00265952 -o reports/lhtest_result.txt
```

or running the shell script we provide:

```
$ cafe cafe_shell_scripts/cafetutorial_run4.sh
```

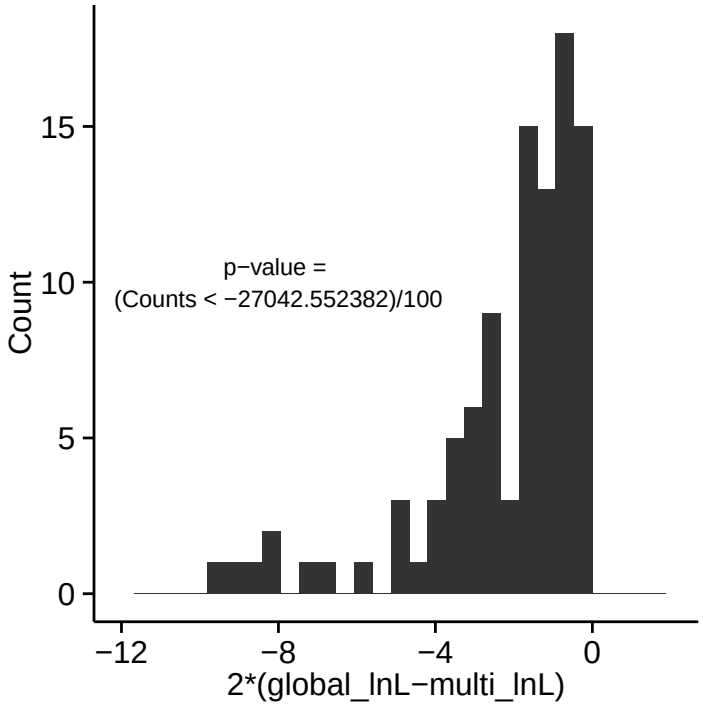
Here, the `genfamily` command simulates the datasets (in the example above, we are asking for 100 simulations with `-t 100`). It estimates λ from the observed data to simulate gene families. Then the likelihoods of the two competing models are calculated with the `lhtest` function, which takes the multi-λ tree structure, and the estimated λ value using the global-λ model.

This command will take a long time for 100 simulations, so go ahead and have a look at the output file that we provide (`reports/lhtest_result.txt`). We can further parse the result of these commands, and plot the null distribution with:

```
$ cut -f 2,4 reports/lhtest_result.txt > reports/run4_lk_diffs.txt
$ Rscript other/lhtest.R reports/run4_lk_diffs.txt -162576.606204
    -149055.330013
```

The numbers -162576.606204 and -149055.330013 are the log-likelihoods of the global-λ and multi-λ models from `reports/log_run1.txt` and `reports/log_run3.txt`, respectively (the negative log-likelihoods are given in these two files). Running the command above creates a histogram with the null distribution from the simulations (`reports/lk_null.pdf`).

Note that the observed likelihood ratio ($2 \times (lnL_{global} - lnL_{multi})$) would fall on the far left tail of the null distribution, yielding a very small p-value, and meaning that the the probability of a multi-λ model fitting better than a global-λ model by chance is very small.



3.3 Estimating separate birth (λ) and death (μ) parameters

The assumption that $\lambda = \mu$ can be relaxed by asking CAFE to estimate them separately. Below you will find the commands to do this analysis assuming the whole tree shares the same λ and the same μ, but allowing these two parameters to be estimated (you can also define species and clades that should share the same λs and μs with the `-t` parameter):

```
# # !cafe
# load -i filtered_cafe_input.txt -t 4 -l reports/log_run5.txt
# tree ((((cat:68.710507,horse:68.710507):4.566782,cow:73.277289)
   :20.722711,(((((chimp:4.444172,human:4.444172):6.682678,orang
   :11.126850):2.285855,gibbon:13.412706):7.211527,(macaque:4.567240,
   baboon:4.567240):16.056992):16.060702,marmoset:36.684935)
   :57.315065):38.738021,(rat:36.302445,mouse:36.302445):96.435575)
# lambdamu -s -t ((((1,1)1,1)1,(((((1,1)1,1)1,1)1,(1,1)1)1,1)1)1,(1,1)
   1)
# report reports/report_run5
```

Or simply run CAFE on the shell script we provide with:

```
$ cafe cafe_shell_scripts/cafetutorial_run5.sh
```

Here, you will want to open file `reports/log_run5.txt` (the `.cafe` file will have the same structure as when a single birth-death parameter was estimated), as it will give you the separate λ and μ values. In our run, we obtained $\lambda = 0.00265363$ and $\mu = 0.00266528$.

3.4 Estimating an error model to account for genome assembly error

Errors in the assembly of a genome (and its annotation) can cause the observed number of gene copies in gene families to deviate from the true ones, possibly leading to a downstream overestimation of λ. In order to account for assembly errors, the latest version of CAFE can estimate the error distribution of a dataset without any external data, which can then be used in λ analyses. We provide a Python script (`caferror.py`) that iteratively searches error distributions defined *a priori*, and returns the one that maximizes the probability of observing the data. In order to run it, enter the following command on your terminal:

```
$ python caferror.py -i cafe_shell_scripts/cafetutorial_run6.sh -d
   reports/run6_caferror_files -v 0 -f 1
```

`caferror.py` will estimate the error model always making the assumption that all branches in the tree share the same unique λ (see `cafe_shell_scripts/cafetutorial_run6.sh`; NOTE: this shell script must specify the full path to the cafe binary on its first line, regardless of whether you added this path to your `$PATH` shell variable). Then the estimated error model can be used on different CAFE analyses. In the case above, we are telling `caferror.py` to run CAFE one initial time without any error model ('`-f 1`').

Because `caferror.py` essentially runs CAFE several times, it can take a while to estimate the error model, and so we provide you with the output of the command above in the folder `reports/run6_caferror_files`. Inside the folder, the file `caferror_default_output.txt` will have the scores for the different error models searched; here,

you want to observe the values in the second column (the lower, the better). Our file looks like this:

ErrorModel	Score
0.4	162576.628503
0.4	166462.116642
0.2	148487.316800
0.1	145776.419850
0.05	147792.113653
0.125	145844.837758
0.0875	145960.422682
0.10625	145743.199198
0.115625	145758.495205
0.053125	147541.731451
0.084375	146033.486212
0.0921875	145871.878264
0.10390625	145751.384434
0.109765625	145740.147917
0.1126953125	145745.745599
0.0548828125	147410.677977
0.0837890625	146048.477903
0.0982421875	145792.494210
0.10546875	145745.371527
0.10908203125	145739.899086
0.110888671875	145741.431870
0.054541015625	147435.615313
0.08271484375	146077.074291

The best error model in our example above is 0.10908203125, and its respective distribution can be found in file `reports/run6_caferror_files/cafe_errormodel_0.10908203125.txt`. With this file in hand, you can now estimate λ values once again. Let us estimate three different λ values, now with an error model:

```
# #!cafe
# load -i filtered_cafe_input.txt -t 4 -l reports/log_run7.txt
# tree ((((cat:68.710507,horse:68.710507):4.566782,cow:73.277289)
   :20.722711,(((((chimp:4.444172,human:4.444172):6.682678,orang
   :11.126850):2.285855,gibbon:13.412706):7.211527,(macaque:4.567240,
   baboon:4.567240):16.056992):16.060702,marmoset:36.684935)
   :57.315065):38.738021,(rat:36.302445,mouse:36.302445):96.435575)
# errormodel -model run6_caferror_files/cafe_errormodel_0.10908203125.
   txt -all
# lambda -s -t ((((3,3)3,3)3,(((((1,1)1,2)2,2)2,(2,2)2)2,3)3)3,(3,3)3)
# report report_run7
```

or by simply running the shell script we provide:

```
$ cafe cafe_shell_scripts/cafetutorial_run7.sh
```

After CAFE finishes running, you can check the three λ estimates using the specified error model (`reports/run7_report.cafe`): 0.01162034494707, 0.00258542894348, and 0.00106942303815. As you can see, the three values are lower than when no error model was employed. This is accordance with the fact that error in genome assembly and gene annotation should artefactually inflate the rate of gene family evolution, and therefore controlling for it should lead to smaller estimates of λ. Nevertheless, the estimate of λ for human, chimpanzee, and their ancestor is still larger than the other two λs – and so the acceleration of gene family evolution in these species is still a result despite correction for error.