Ancestral State Reconstruction for Binary and Categorical Traits Walkthrough

November 26, 2022

Contents

This walkthrough describes the phenotype inference step of an RERconverge analysis in more detail. The method of ancestral state reconstruction described in this walkthrough can be used with binary or categorical traits, but not continuous traits. This walkthrough builds on existing RERconverge objects. First time users should refer to the "RERconverge Analysis Walkthrough" vignette and the "Categorical Trait Analysis Walkthrough" vignette.

Overview

In order to follow along with this walkthrough, you will first need to install RERconverge, read in your gene trees, and read in your phenotype data. For detailed instructions on how to install RERconverge, reference the RERconverge install page. For instructions on reading in trees and phenotype data, reference the "RERconverge Analysis Walkthrough" vignette.

Every RERconverge analysis involves a phenotype inference step, in which the phenotype data for the living species at the tips of the tree is used to infer the phenotypes of ancestral species in the tree. For categorical traits, RERconverge uses maximum likelihood estimation to obtain the likelihoods at each internal node of being in any given state of the phenotype. This method can just as easily be used on binary traits though it works differently than the method currently used by RERconverge for binary traits.

Maximum likelihood estimation requires a model of evolution to describe the probability of transitioning between phenotype states. This model is a continuous time markov model, in which a square transition matrix, denoted Q, describes the instantaneous rate of transitioning from state i to state j at position (i,j). All off diagonal elements of the transition matrix are non-negative, while the diagonal elements are negative such that each row sums to zero. On each branch of the tree, the transition probability matrix, P, is obtained by taking the exponential of Q multiplied by the branch length. One nice property of this formulation is that the rows of P will sum to 1 such that the entries represent probabilities of transitioning between states. The probabilities on the diagonal represent the probability of staying in a given state. The true rates in the transition matrix are hardly ever known for sure. Therefore, they are inferred in order to maximize the probability of the observed data at the tips of the tree. This walkthrough will refer to the process of estimating optimal transition rates as fitting the transition matrix.

RERconverge relies on functions from the castor(Louca and Doebeli 2017) and ape(Paradis and Schliep 2019) packages to fit the transition matrix and obtain ancestral likelihoods. Currently, RERconverge assigns to each node the state with the maximum likelihood.

Thus, to obtain ancestral states in the phenotype tree, an RERconverge user must only supply a phylogenetic tree, phenotype data, and a rate model. The rate model describes constraints on fitting the transition matrix. For instance, an equal rates model forces every transition rate to be the same. On the other hand, an all rates different model allows each transition rate to take on an independent value.

Rate models can be used to incorporate prior biological knowledge. For instance, position (i,j) can be set to zero if you know that transitions between state i and j cannot happen directly. It is important to note that the transition matrix describes instantaneous transition rates. Multiple transitions can occur along a single branch in the tree. Thus, setting position (i,j) to zero does not prevent an i to j transition from occurring along a single branch.

This walkthrough will review the diagnostics provided by RERconverge for choosing the optimal rate model for your analysis.

Getting Started

Data Input Requirements and Formatting

The required inputs are as follows:

1. Phylogenetic trees of the same format described in the "RERconverge Analysis Walkthrough" vignette.

2. Species-labeled phenotype values

- The species labels MUST match the tree tip labels that were used in `getAllResiduals` to calculate the relative evolutionary rates (RERs)
- a named vector of categorical trait values

Reading in the Data

For the purpose of this walkthrough, we will read in the gene trees and phenotype data as shown below. The data used in this walkthrough is on mammal sleep patterns, a categorical trait with the following categories: cathemeral, crepuscular, diurnal and nocturnal.

For more details about reading in the specific data you need for your analysis, refer to the "Categorical Trait Analysis Walkthrough" vignette and the "RERconverge Analysis Walkthrough" vignette.

```
# check that RERconverge was successfully installed
library(RERconverge)

# find where the package is located on your machine
rerpath = find.package('RERconverge')

# read in the trees with the given file name
toytreefile = "subsetMammalGeneTrees.txt"
toyTrees=readTrees(paste(rerpath,"/extdata/",toytreefile,sep=""),
max.read = 200)

# load the phenotype data into your workspace
# This will create a named vector with the name basalRate
data("basalRate")
```

Finding the Optimal Rate Model

The "optimal" rate model depends on a number of factors, and even then there may be no way of knowing the true best rate model. Therefore, the diagnostics that will be described in this section are designed to provide more information and insight about potential rate models in order to help you make an educated selection. These diagnostics do not take prior biological knowledge into account. Thus it is up to the user to interpret the results, possibly rejecting models that do not make sense biologically.

Most of the diagnostics explained below use the likelihood ratio to compare rate models. The likelihood ratio is calculated as $-2 \times log_2(\frac{\text{likelihood of simpler model}}{\text{likelihood of more complex model}})$. When the simpler model is a special case of the more complex model (a.k.a. when the models are nested), then the likelihood ratio is distributed as a chi squared distribution with the degrees of freedom equal to the difference in the number of free parameters between the simpler and more complex models. When they are not nested, the p-value can still be obtained by using simulations to generate the null distribution for the likelihood ratio (Pagel 1994).

Specifying a Rate Model

All rate models can be described by a square matrix with dimensions equal to the number of phenotype states. The diagonal elements must be zero because they are dependent on the value of the off-diagonal elements, thus are not a free parameter. Everywhere else in the matrix, each free parameter should be assigned a unique number and any off-diagonal elements that are set to zero represent transitions that will be set to zero in the transition matrix. The order of unique numbers does not matter as long as the numbers are consecutive and begin at 1. For example, the all rates differnt model can be specified as `"ARD"` for short or, equivalently for a phenotype with three states, `matrix(c(0,1,2,3,0,4,5,6,0),3)`. The equal rates model can be specified as `"ER"` for short, or equivalently for a phenotype with three states,

`matrix(c(0,1,1,1,0,1,1,1,0),3)`. Finally, the rate model abbreviated as `"SYM"` for a symmetric model is equivalent to `matrix(c(0,1,2,1,0,3,2,3,0),3)`in which every rate (i,j) is equal to the rate (j,i).

Performing a Rate Model Search

One guiding principle for selecting a rate model is to choose the simplest model that still fits the data well. However, the number of all possible rate models to search over becomes too large once the number of phenotype categories exceeds two or three. Therefore, the `searchRateModels` function uses an iterative approach to find simpler models that don't sacrifice their ability to fit the data. This approach was inspired by an algorithm described in Jayaswal 2011 (Jayaswal et al. 2011). A brief overview of the method is as follows:

1. A list is created to store the rate models. To begin with, the list will only contain the all rates different (ARD) model which allows every transition to take on a different value and is thus the most complex model. We start our counter at `i = 1` and fit a transition matrix using the `ith` rate model in this list, which to begin with is the ARD model. However, if the `ith` rate model has only one free parameter (the simplest rate model), we do not move on to step 2 since we cannot generate simpler models from this model. Instead we skip to step 4 to move on to the next rate model in the list. If there is no next model, we are done.
2. Based on the transition matrix, we generate two new models that decrease the number of free parameters. In the first new model, we set the two positions with the closest transition rates equal. In the second new model, we set the position with the smallest transition rate to zero. More precisely, this step may generate more than two new models if there are multiple positions in the transition matrix with the same value, but each new model is generated similarly to one of the two ways described above.
3. Fit the new models to the data and calculate the likelihood ratios between the previous model and each new model. If the previous, more complex, model is NOT significantly better at describing the observed data as determined by the likelihood ratio, then we add the new model to the list of rate models. The new model represents a simpler model that is equally capable of describing the observed data.
4. Increment the counter, `i = i + 1`.

Eventually, no new models are added to the list of rate models either because they cannot be simplified further or because any simpler model would not be able to describe the observed data as well as the more complex models.

This function **outputs** the list of rate models that was generated during the search. It also outputs a table containing the likelihood ratios between the new models and previous models that were computed on each iteration of the algorithm. When the models are nested, a p-value for the likelihood ratio is also included in the table. In this table, `prevIndex` and `newIndex` refer to the indices of the previous and new rate models in the list of rate models that is also returned by this function. This table also contains the AIC and loglikelihood of fitting the transition matrix in step 1 above.

To perform a rate model search use `searchRateModels`, which takes the following inputs:

- `treesObj`: the trees object returned by `readTrees` (see the "RERconverge Analysis Walkthrough" vignette for information about `readTrees`)
- `phenvals`: the named phenotype vector
- `pthreshold`: If the likelihood ratio has a p-value that is less than or equal to `pthreshold`, then the more complex model is considered significantly better at fitting the transition matrix to the observed data and the new model is discarded. `pthreshold` is only used when the new model and previous model are nested, thus a p-value can be obtained from the chi squared distribution.
- `lthreshold`: The threshold for the likelihood ratio used when the previous model and new model are not nested, thus a p-value cannot be obtained from the chi squared distribution. The simulation based approach is not used in this case to determine a p-value because it is too time consuming. If the

likelihood ratio is above this threshold, then the more complex model is considered significantly better at fitting the transition matrix to the observed data and the new model is discarded.

- `max_iterations`: If not `NULL`, the algorithm will stop when `i > max_iterations` in order to avoid long run times. The default is `2000`. When an analysis is stopped due to exceeding the `max_iterations`, a message indicating such will be printed to the console.
- `...`: additional parameters for `fit_mk`, the castor function used to fit the transition matrix from the rate model

Increasing `pthreshold` and decreasing `lthreshold` weakens the conditions under which the more complex model is considered significantly better. Equivalently, it creates stricter conditions for considering a simpler rate model equally capable of fitting the transition matrix to the observed data. This *increases* the number of simpler rate models that get discarded, speeding up the analysis and limiting the total number of results that are generated.

```
library(RERconverge)
search_res = searchRateModels(toyTrees, basalRate, pthreshold = 0.25, lthreshold = 1.3)

# view the generated models
tail(search_res$models)

# view the model statistics
tail(search_res$stats)
```

Although `searchRateModels` significantly reduces the search space, it can still generate over one thousand models. In order to filter through these models, use `filterByCriteria`, which takes the following parameters:

- `models`: The list of rate models to filter
- `criteria`: A function specifying the filter criteria. The body of this function must be a boolean expression, the form of which is shown below. This function may be used to filter for models that match prior biological knowledge and/or to filter for models with a certain number of zero transitions and/or number of free parameters.

Use `map_to_state_space` to see which indices in the rate models correspond to the categories in your analysis.

```
# View the category to integer mapping
map_to_state_space(basalRate)$name2index
```

```
## high  low  med
##    1    2    3
```

```
# rm is a rate model in the list returned by searchRateModels
# Example 1: If you know something about transitions between states
criteria1 <- function(rm) {
  # filter out rate models in which the high to med transitions are not zero
  # and the high to low transitions are not zero
  rm[1,3] != 0 && rm[1,2] != 0
}

filtered_res1 = filterByCriteria(search_res$models, criteria1)
tail(filtered_res1)
```

```
## [[1]]
##      [,1] [,2] [,3]
## [1,]    0    1    2
## [2,]    0    0    1
## [3,]    1    2    0
```

```
##
## [[2]]
##      [,1] [,2] [,3]
## [1,]    0    1    2
## [2,]    2    0    1
## [3,]    1    0    0
##
## [[3]]
##      [,1] [,2] [,3]
## [1,]    0    1    2
## [2,]    2    0    1
## [3,]    1    1    0
##
## [[4]]
##      [,1] [,2] [,3]
## [1,]    0    1    1
## [2,]    1    0    0
## [3,]    1    1    0
##
## [[5]]
##      [,1] [,2] [,3]
## [1,]    0    1    2
## [2,]    1    0    0
## [3,]    1    1    0
##
## [[6]]
##      [,1] [,2] [,3]
## [1,]    0    1    1
## [2,]    1    0    2
## [3,]    1    1    0
```

```
# Example 2: filtering by number of zeros and number of free parameters
criteria2 <- function(rm) {
  # no more than 3 transitions between different states set to zero
  # (recall there will be 3 zeroes from the diagonal, so <= 6 zeroes overall)
  # and no more than 4 free parameters
  sum(rm == 0) <= 6 && max(rm) <= 4
}

filtered_res2 = filterByCriteria(search_res$models, criteria2)
tail(filtered_res2)
```

```
## [[1]]
##      [,1] [,2] [,3]
## [1,]    0    1    1
## [2,]    1    0    0
## [3,]    1    1    0
##
## [[2]]
##      [,1] [,2] [,3]
## [1,]    0    1    0
## [2,]    1    0    2
## [3,]    1    1    0
##
## [[3]]
```

```
##      [,1] [,2] [,3]
## [1,]    0    1    2
## [2,]    1    0    0
## [3,]    1    1    0
##
## [[4]]
##      [,1] [,2] [,3]
## [1,]    0    1    1
## [2,]    1    0    2
## [3,]    1    1    0
##
## [[5]]
##      [,1] [,2] [,3]
## [1,]    0    1    0
## [2,]    0    0    1
## [3,]    1    2    0
##
## [[6]]
##      [,1] [,2] [,3]
## [1,]    0    1    0
## [2,]    2    0    1
## [3,]    1    1    0
```

Comparing Rate Models

Once you have a set of candidate rate models, you can use `compareRateModels` to perform pairwise likelihood ratio comparisons and determine which rate model best fits the observed data. For instance, one could select a subset of rate models from the rate model search described previously. The rate model comparison is fastest and most interpretable on a small set of ten or fewer rate models.

The **output** of `compareRateModels` is a square table of either p-values or likelihood ratios for pairwise comparisons between rate models. The diagonal of the table contains only `NA` values since rate models are not compared to themselves. Additionally, each comparison is only performed once with the simpler model in the numerator. Thus if position (i,j) in the table has a p-value or likelihood ratio, you should expect position (j,i) to be `NA`.

The model corresponding to the row will always be the simpler model and the model corresponding to the column will always be the more complex model. Thus, a small p-value or large likelihood ratio is evidence that the more complex model, corresponding to the column, provides a significantly better fit to the data at the tips. On the other hand, a large p-value or small likelihood ratio is evidence that the simpler model, corresponding to the row, is equally capable of providing a good fit to the data as the more complex model.

The code below runs you through how to make a list of rate models and run it on `compareRateModels` which takes the following as input:

- `rate_models`: a named list of rate models to compare. All rate models in the list must be matrices. Use the function `getMatrixFromAbbr` to convert `"ER"`, `"ARD"`, and `"SYM"` to their respective matrices.
- `treesObj`: the trees object returned by `readTrees`.
- `phenvals`: the named phenotype vector
- `nsims`: the number of simulations to use in order generate the null distribution of likelihood ratios for calculating a p-value when two models are not nested. The default value is `100`. Decreasing `nsims` would decrease run time at the expense of statistical validity.
- `nested_only`: a boolean specifying whether to only calculate p-values for nested models. The default is `FALSE`. Setting `nested_only` to `TRUE` speeds up the analysis, but will leave out comparisons between

non-nested models.

- `return_type`: The default value is `"pvals"`, meaning the comparisons between rate models are returned as p-values for the likelihood ratios. Alternatively, setting `return_type` to `"ratios"` will return the likelihood ratios themselves without calculating p-values. This is an alternative way to speed up the analysis while still calculating comparisons between non-nested models. Note: while p-values have traditional thresholds for statistical significance, interpreting likelihood ratios is less well defined. Nonetheless, the larger the likelihood ratio, the more support there is for the more complex model being a significantly better fit.
- `...`: additional parameters for `fit_mk`, the castor function used to fit the transition matrix from the rate model.

```
# the list of rate models includes the three common rate models-- "ER", "ARD", and "SYM"--, two rate mo

custom = matrix(c(0, 0, 1, 0, 0, 2, 0, 3, 0),3)

# getMatrixFromAbbr takes the abbreviation ("ER", "ARD", or "SYM") and the number of phenotype states
rate_models = list(getMatrixFromAbbr("ER", 3), getMatrixFromAbbr("ARD",3),
                   getMatrixFromAbbr("SYM",3), filtered_res1[[23]], filtered_res1[[5]],
                   custom)

names(rate_models) = c("ER", "ARD", "SYM", "search1", "search2", "custom")

comp = compareRateModels(rate_models = rate_models,
                         treesObj = toyTrees,
                         phenvals = basalRate,
                         nsims = 100,
                         nested_only = FALSE,
                         return_type = "pvals")
```

```
# view the comparisons
comp
```

```
##         ER     ARD     SYM search1 search2 custom
## ER      NA 0.70816 0.49896    0.23 0.41327   0.66
## ARD     NA      NA      NA      NA      NA     NA
## SYM     NA 0.66928      NA      NA 0.40000     NA
## search1 NA 0.95055 0.77000      NA 0.52000   0.74
## search2 NA 0.95875      NA      NA      NA     NA
## custom  NA 0.02121 0.00000      NA 0.00000     NA
```

```
# refer back to the rate models
rate_models
```

```
## $ER
##      [,1] [,2] [,3]
## [1,]    0    1    1
## [2,]    1    0    1
## [3,]    1    1    0
##
## $ARD
##      [,1] [,2] [,3]
## [1,]    0    3    5
## [2,]    1    0    6
## [3,]    2    4    0
##
```

```
## $SYM
##      [,1] [,2] [,3]
## [1,]    0    1    2
## [2,]    1    0    3
## [3,]    2    3    0
##
## $search1
##      [,1] [,2] [,3]
## [1,]    0    1    2
## [2,]    1    0    2
## [3,]    1    0    0
##
## $search2
##      [,1] [,2] [,3]
## [1,]    0    1    3
## [2,]    4    0    4
## [3,]    1    2    0
##
## $custom
##      [,1] [,2] [,3]
## [1,]    0    0    0
## [2,]    0    0    3
## [3,]    1    2    0
```

From the results above, we may conclude that the more complex models "ARD", "SYM", search1, and search2 were not significantly better at fitting the data compared to the "ER" model because the p-values in the "ER" row were relatively high. On the other hand, notice that the custom rate model had many transition rates set to 0 which does not seem very plausible in the context of this trait. This is reflected in the bottom row of the table in which the p-values are 0.021 and 0, indicating that the more complex models "ARD", "SYM", and search2 were significantly better at fitting the data compared to this implausible model. Additionally, the custom model was not significantly better at fitting the data compared to the simpler rate models "ER" and search1 (with p-values of 0.81 and 0.73 respectively). This is not to say that the model(s) with the best fit are necessarily the best model(s), however comparing goodness of fit is a useful diagnostic for narrowing down the search space of potential rate models and comparing a few candidate rate models to each other.

Assessing Prediction Accuracy

The third method for comparing rate models aims to measure prediction accuracy rather than comparing goodness of fit to the observed data.

The RERconverge function, `boxPlotTest`, takes a list of rate models and returns a box plot for each rate model in the list. This function works as follows:

1. Get the next rate model in the list
2. Fit a transition matrix using the rate model obtained in step 1
3. Simulate `nsims` times from the transition matrix obtained in step 2 (using the `simulate_mk_model` function provided by the castor package)
4. For each simulation, use the simulated states at the tips of the tree to infer the states of internal nodes under every rate model in the list. Compare the inferred states to the actual states generated by the simulation and calculate the percentage of states that were assigned correctly.
5. Make a box plot showing the percent of correct predictions on the y axis and the rate model on the x axis.
6. Repeat steps 1-5 for every rate model in the list of rate models.

`boxPlotTest` takes the following parameters:

- `treesObj`: the trees object returned by `readTrees`
- `phenvals`: named phenotype vector
- `rate_models`: the list of rate models to test
- `nsims`: The number of simulations to use per transition matrix
- `confidence_threshold`: the default value is `NULL`. However, if a confidence threshold in the range [0,1] is provided, the percent of correct matches will only be calculated for nodes whose maximum likelihood is greater than or equal to the confidence threshold. This can be beneficial to see whether the more confident state assignments tend to be accurate and if accuracy breaks down at nodes with greater uncertainty, or not.
- `...`: additional parameters for `fit_mk`, the castor function used to fit the transition matrix from the rate model

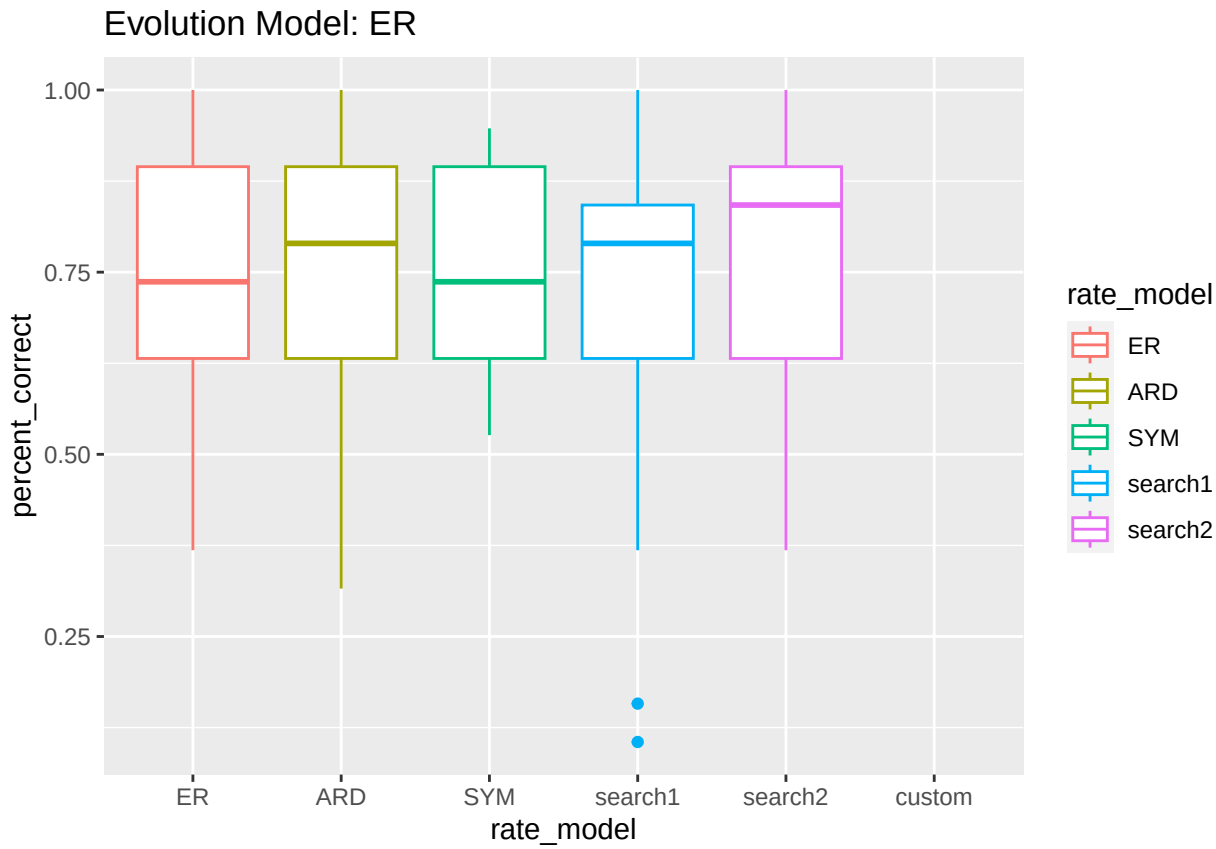
Note that the current implementation of this function may take a few minutes to run on larger trees and will be fastest when all the rate models are symmetrical as long as `use_simmap` is `FALSE`.

```
boxPlots = boxPlotTest(toyTrees, basalRate, rate_models, nsims = 25)
```

When viewing the box plots you may see the following warning: `## Warning: Removed n rows containing non-finite values (stat_boxplot).` This is not a problem, it just means that for some of the simulations, the percent correct (on the y-axis) could not be calculated for one or more of the rate models. This occurs when the rate model is a poor fit to the simulated data thus producing `NaN` values when calculating ancestral likelihoods. (This occurred in the case of our custom model which was purposely chosen to represent an implausible model).

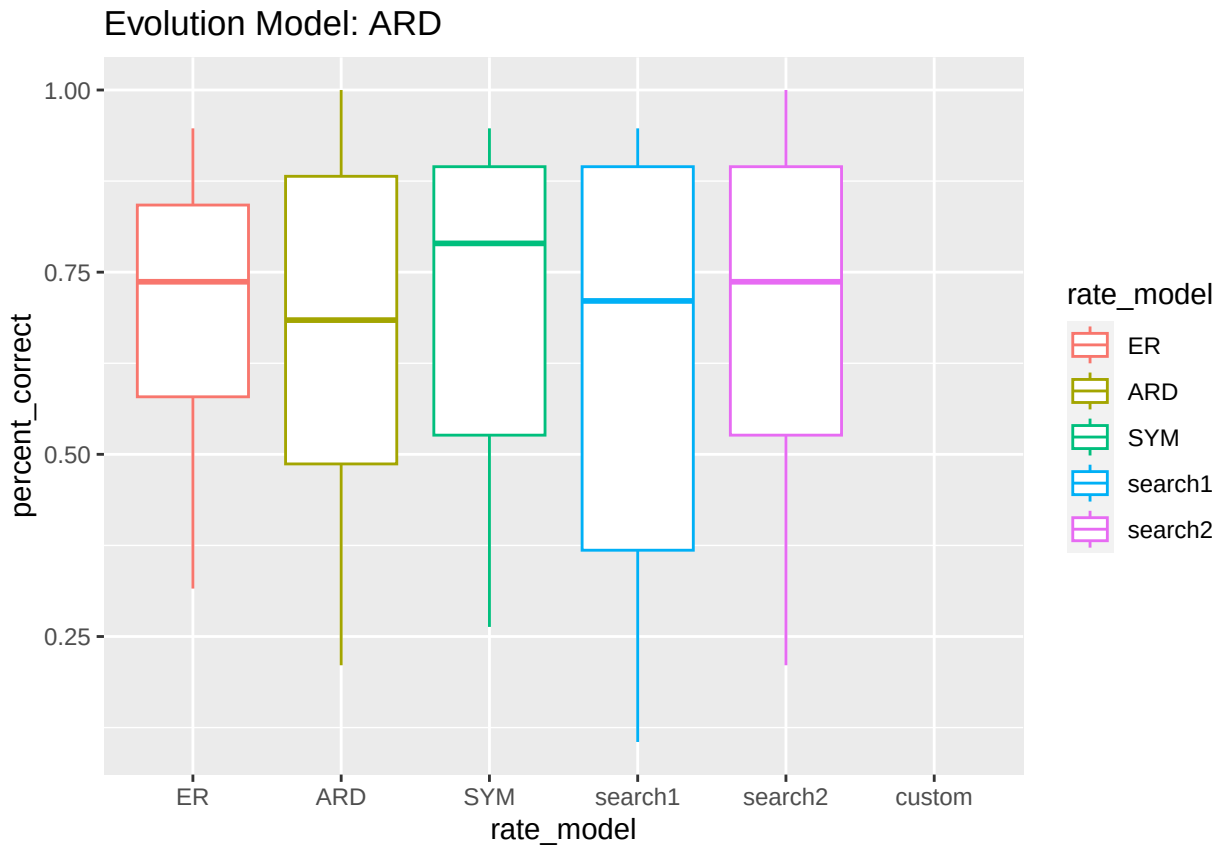
```
# View the box plots
boxPlots$plots[[1]]
```

```
## Warning: Removed 25 rows containing non-finite values (`stat_boxplot()`).
```



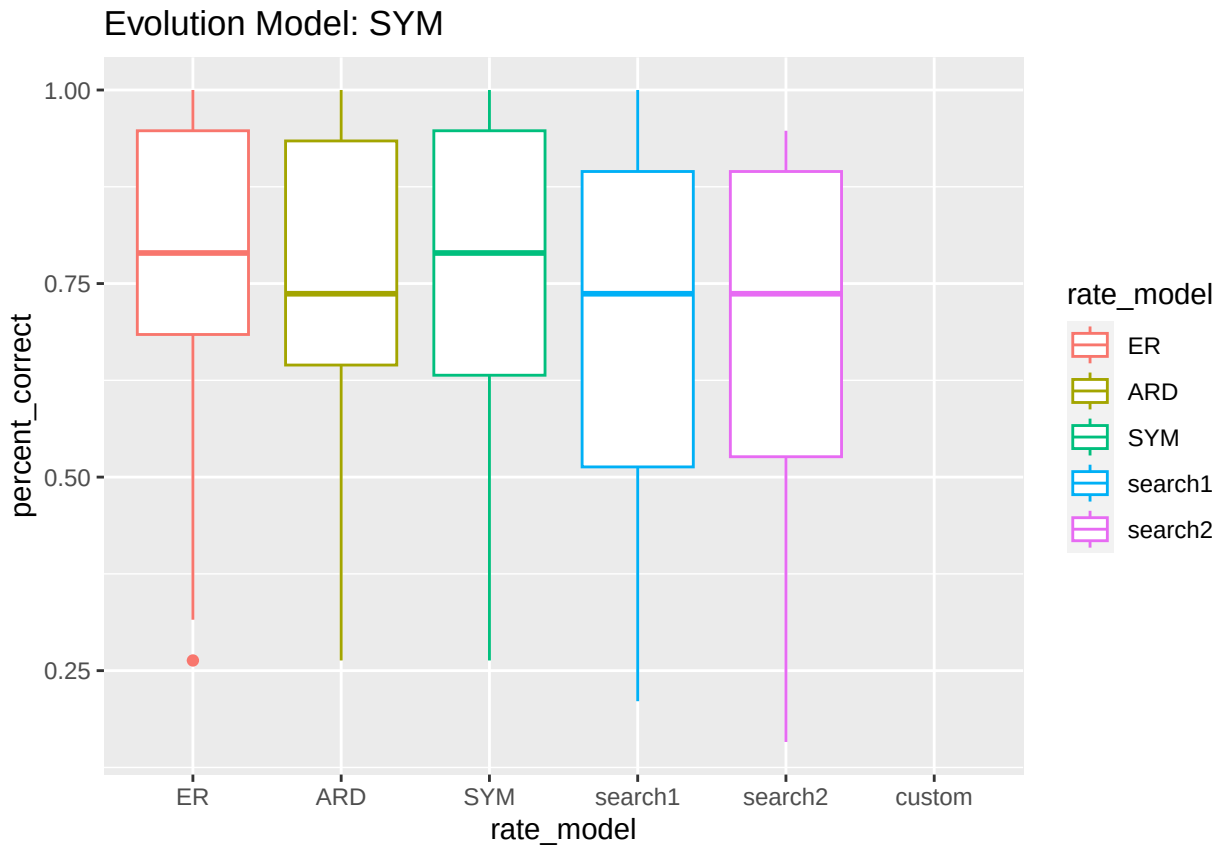

```
boxPlots$plots[[2]]
```

```
## Warning: Removed 29 rows containing non-finite values (`stat_boxplot()`).
```



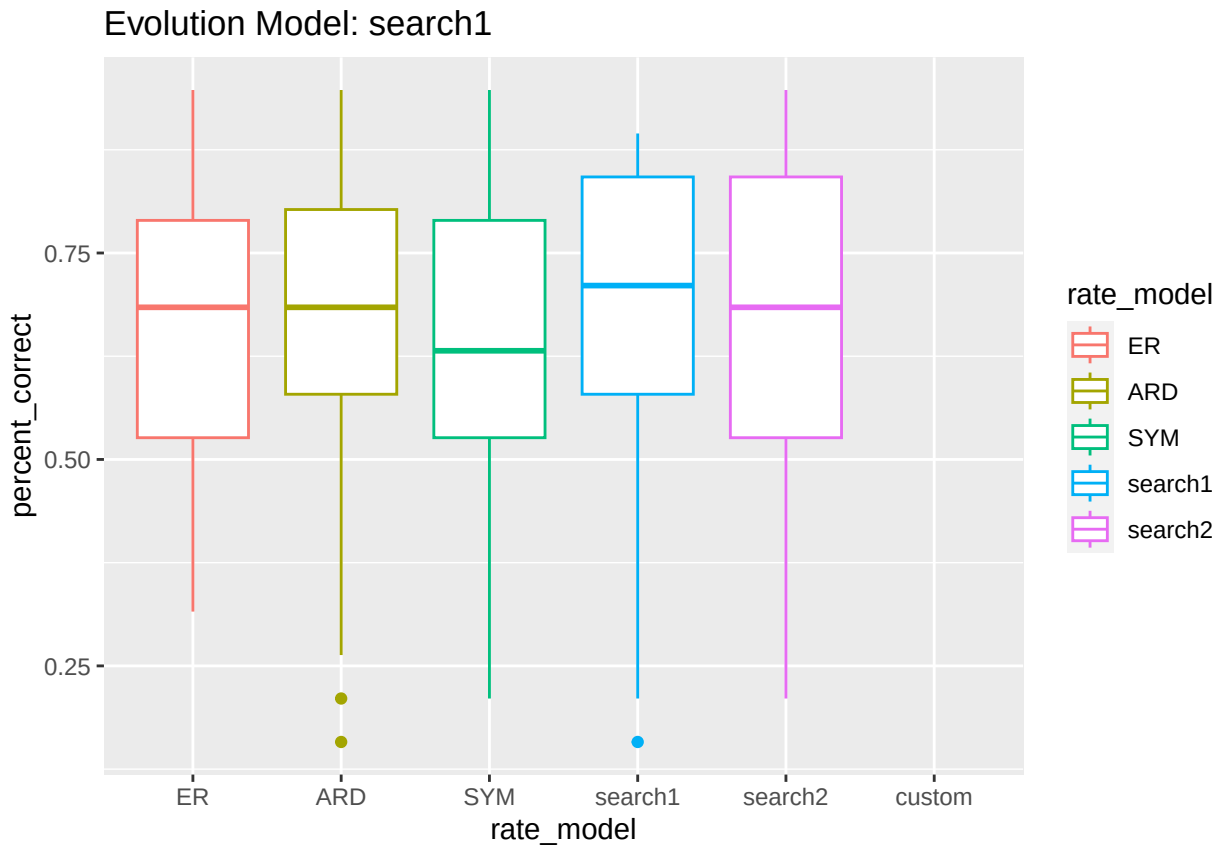

```
boxPlots$plots[[3]]
```

```
## Warning: Removed 29 rows containing non-finite values (`stat_boxplot()`).
```



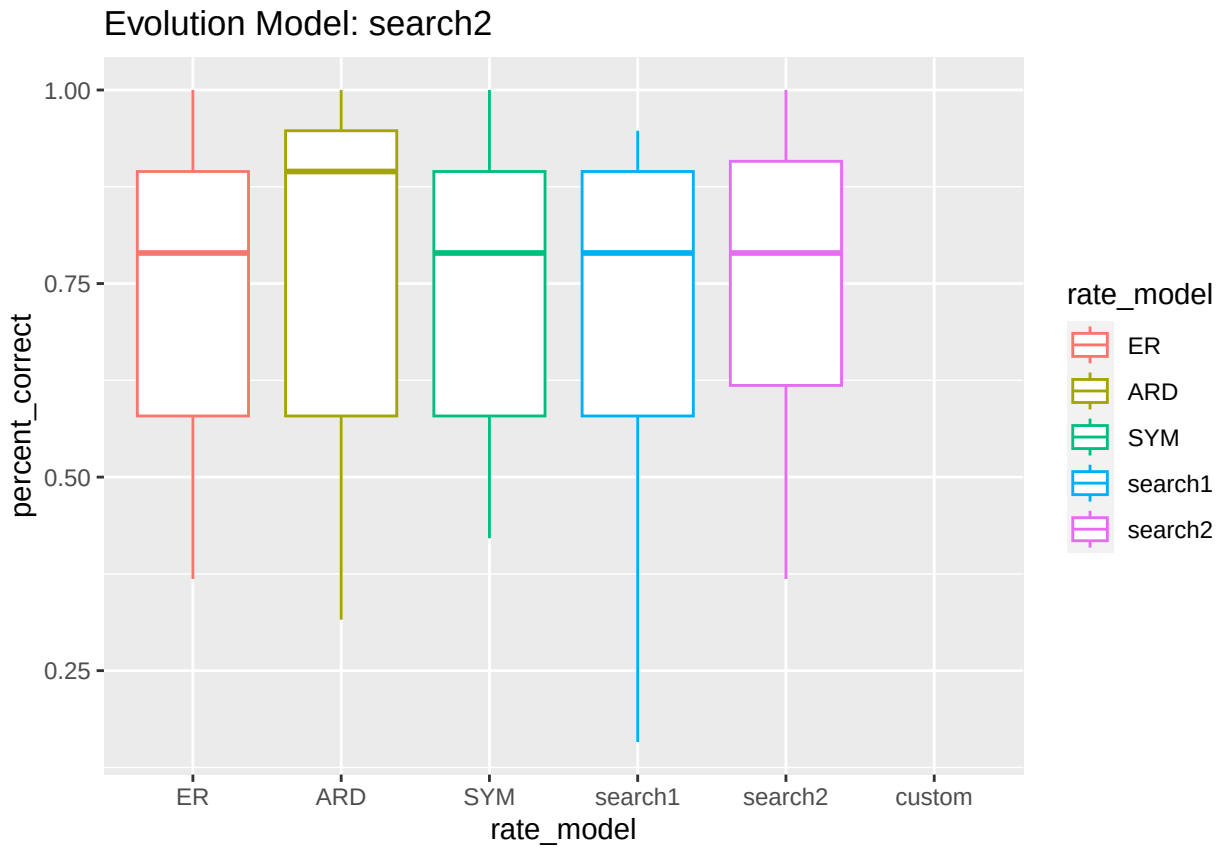

```
boxPlots$plots[[4]]
```

```
## Warning: Removed 27 rows containing non-finite values (`stat_boxplot()`).
```




```
boxPlots$plots[[5]]
```

```
## Warning: Removed 28 rows containing non-finite values (`stat_boxplot()`).
```




```
boxPlots$plots[[6]]
```

Simulations from custom evolution model failed to generate extant species of every cate

```
# View the transition matrices fit under each model
boxPlots$transition_matrices
```

```
## $ER
##           [,1]      [,2]      [,3]
## [1,] -7.190801  3.595401  3.595401
## [2,]  3.595401 -7.190801  3.595401
## [3,]  3.595401  3.595401 -7.190801
##
## $ARD
##           [,1]      [,2]      [,3]
## [1,] -6.314164  6.314164  0.000000
## [2,]  2.302702 -6.164785  3.862083
## [3,]  6.034574  0.000000 -6.034574
##
## $SYM
##           [,1]       [,2]       [,3]
## [1,] -9.457521  7.1365768  2.3209441
## [2,]  7.136577 -8.0353985  0.8988216
## [3,]  2.320944  0.8988216 -3.2197657
##
## $search1
##           [,1]      [,2]      [,3]
## [1,] -8.277957  6.621821  1.656137
## [2,]  6.621821 -8.277957  1.656137
## [3,]  6.621821  0.000000 -6.621821
##
```

```
## $search2
##           [,1]      [,2]      [,3]
## [1,] -6.642621  6.642621  0.000000
## [2,]  3.478894 -6.957788  3.478894
## [3,]  6.642621  0.000000 -6.642621
##
## $custom
##          [,1]      [,2]      [,3]
## [1,]  0.00000   0.00000   0.00000
## [2,]  0.00000 -21.12679  21.12679
## [3,] 17.12491  49.35531 -66.48022
```

Simulating from a transition matrix that was fit on each rate model represents alternative models of evolution that the phenotype may have followed along the tree. Then under each model of evolution, the box plot allows you to evaluate the predication accuracy of each rate model. This can reveal whether one or more rate models perform better over others regardless of the model of evolution.

Visualizing Reconstructions Under Different Rate Models

Finally, you can use `visCompareTwoRateModels` to compare how closely ancestral likelihoods or ancestral state assignments agree between two different rate models. `visCompareTwoRateModels` takes the following as input:

- `A`: the first rate model
- `B`: the second rate model
- `treesObj`: the trees object returned by `readTrees`
- `phenvals`: the named phenotype vector
- `mode`: the default mode is `"entropy"`. Under this mode, the relative entropy between the ancestral likelihoods at each node is calculated and plotted along the branches of the tree. Red corresponds to greater relative entropy, indicating greater disagreement between ancestral likelihoods. Blue corresponds to lower relative entropy, indicating greater agreement between ancestral likelihoods. The other option is `"match"`, which measures whether or not the states with the maximum ancestral likelihood at each node are the same between the different rate models. A red node indicates that the states differ and a blue node indicate that they are the same.
- `cex`: A graphing parameter for setting the font size of node labels and tree tip labels. The default value is `0.5`.
- `...`: additional parameters for `asr_mk_model`, the castor function for calculating ancestral likelihoods.

The function returns either a list of relative entropies or a table of state assignments under each rate model in node order. It also returns the lists of ancestral likelihoods for each rate model. The numbers on the nodes are simply the node indices in the tree.

```
# comparing rate_models$search2 to rate_models$ER
relative_entropies = visCompareTwoRateModels(rate_models$ER,
                                             rate_models$search2,
                                             toyTrees,
                                             basalRate, mode = "entropy")
```


```
head(relative_entropies$relative_entropies)
```

```
##      21      22      23      24      25      26
## 0.01542 0.24476 0.00612 0.08126 0.25210 0.09531
```

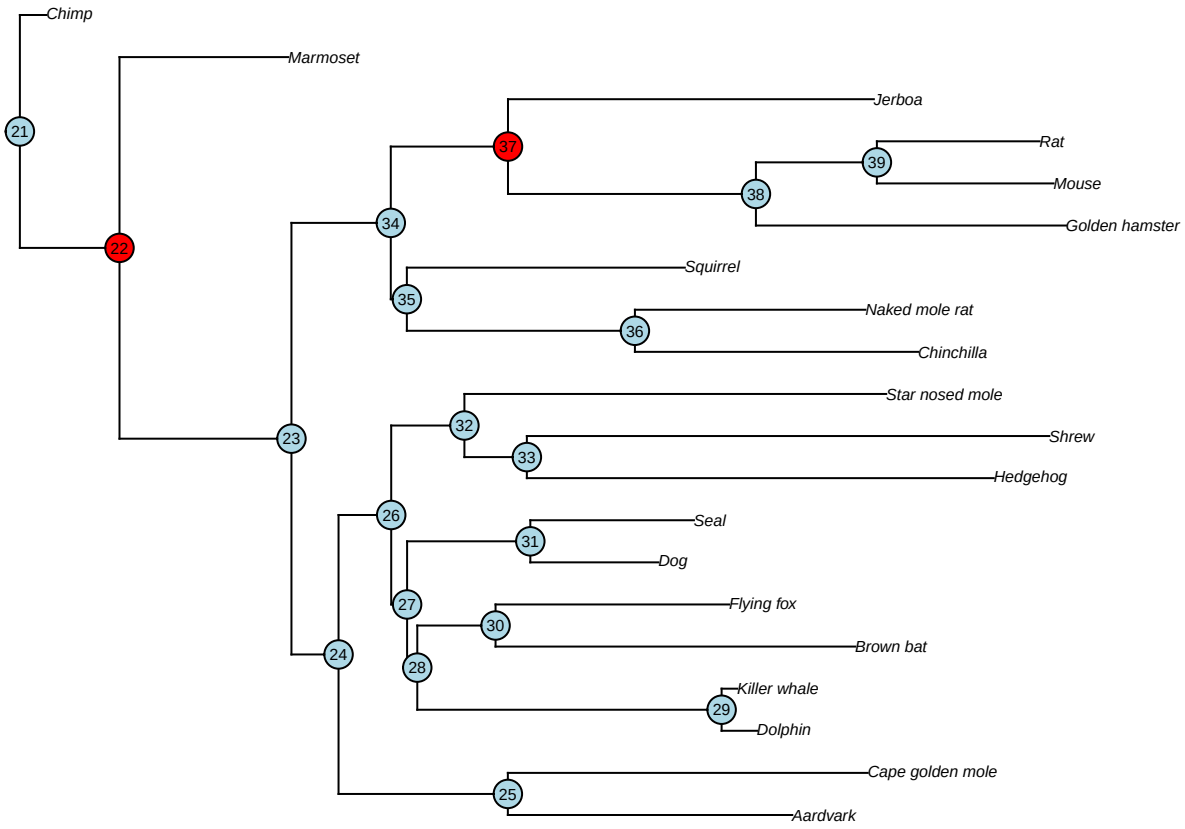
```
head(relative_entropies$anc_liks1) # ancestral likelihoods for the ER model
```

```
##           [,1]       [,2]       [,3]
## [1,] 0.9112745 0.06928253 0.01944302
## [2,] 0.6262402 0.32951222 0.04424757
## [3,] 0.6440816 0.30563231 0.05028610
## [4,] 0.7153664 0.23712253 0.04751107
## [5,] 0.3672484 0.40331141 0.22944017
## [6,] 0.8769956 0.10495617 0.01804819
```

```
head(relative_entropies$anc_liks2) # ancestral likelihoods for the search2 model
```

```
##           [,1]       [,2]       [,3]
## [1,] 0.9466677 0.03994015 0.01339210
## [2,] 0.3544074 0.61749880 0.02809378
## [3,] 0.6136817 0.31539486 0.07092345
## [4,] 0.7366194 0.14180587 0.12157471
## [5,] 0.1351489 0.60372694 0.26112417
## [6,] 0.9117855 0.03337857 0.05483590
```

```
node_states = visCompareTwoRateModels(rate_models$ER,
                                      rate_models$search2,
                                      toyTrees, basalRate,
                                      mode = "match")
```




```
head(node_states$states)
```

```
##    statesA statesB
## 21       1       1
## 22       1       2
## 23       1       1
## 24       1       1
## 25       2       2
## 26       1       1
```

Conclusion

This walkthrough reviewed some diagnostics provided by RERconverge to make an informed decision on which rate model to use in the phylogenetic inference step of a binary or categorical RERconverge analysis. After using one or more of the tools above to select a rate model, run `char2TreeCategorical` and/or `char2PathsCategorical` with that rate model and proceed with the RERconverge analysis as described in the "RERconverge Analysis Walkthrough" vignette or the "Categorical Trait Analysis Walkthrough" vignette.

References

Pagel, Mark. "Detecting Correlated Evolution on Phylogenies: A General Method for the Comparative Analysis of Discrete Characters." *Proceedings: Biological Sciences*, vol. 255, no. 1342, 1994, pp. 37–45. *JSTOR*, http://www.jstor.org/stable/49836. Accessed 26 Nov. 2022.

Jayaswal, V., F. Ababneh, L. S. Jermiin, and J. Robinson. 2011. "Reducing Model Complexity of the General Markov Model of Evolution." *Molecular Biology and Evolution* 28 (11): 3045–59. https://doi.org/10.1093/molbev/msr128.

Louca, Stilianos, and Michael Doebeli. 2017. “Efficient Comparative Phylogenetics on Large Trees.” https://doi.org/10.1093/bioinformatics/btx701.

Paradis, E., and K. Schliep. 2019. “Ape 5.0: An Environment for Modern Phylogenetics and Evolutionary Analyses in {r}” 35: 526–28.