Using KDETrees

Grady Weyenberg

August 21, 2013

1 Introduction

KDETrees is a tool for finding discordant phylogenetic trees. It takes as input an `ape::multiPhylo` object, which contains a set of trees, and produces a score for each tree. (A list of `phylo` objects is also accepted.) High scores mean the tree is relatively similar to other trees in the sample, while low scores indicate that the tree in question may be discordant with the others. Low scoring trees are identified as putative "outliers" with the cutoff controlled by a tuning parameter k, and their contribution to the calculation is removed.

2 Using `kdetrees`

The simplest method of using the software is the `kdetrees.complete` function. This is a convienence function which will do all the steps of the analysis at once. The simplest use is to simply pass it the filename of a Newick file containing the trees to be analyzed. It will write several result files to the R working directory (`getwd`).

This call assumes there is a file containing newick formatted trees called `trees.tre` in the current working directory. It will write out 4 files: `outliers.tre` a newick file containing only the trees identified as outliers; `results.csv` a csv files with the density estimates; `plot.png` and `hist.png` are diagnostic images.

```
> kdetrees.complete("trees.tre")
```

The `kdetrees.complete` function also accepts any of the parameters accepted by the `kdetrees` function, as described in Sections 2.2 and 2.4.

2.1 Importing Trees

Trees may be imported using any of the methods provided by `ape`. (See `?read.tree` and `?read.nexus` for examples.) In the following examples, many functions are a part of the `ape` package, and it is recommended that you import it. For example, to load the `apicomplexa` dataset, I placed the Newick tree strings into the `apicompexa.tre` file and used the following command:

```
> blibrary(ape)
> apicomplexa <- read.tree("apicomplexa.tre")
```

2.2 Running kdetrees

The simplest way to run `kdetrees` is to call the function of the same name, with the list of trees as the first argument.

```
> result <- kdetrees(apicomplexa)

Called from: kdetrees(apicomplexa)

> result

Call: kdetrees(trees = apicomplexa)
Density estimates:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
  0.9963  65.8700  82.6400  77.3000  94.4900 114.5000
Cutoff:  22.92261

Outliers detected:
 [1] 488.tre 497.tre 515.tre 546.tre 547.tre 641.tre 660.tre
 [8] 662.tre 728.tre 747.tre 773.tre 780.tre
```

There are 3 main settings which control the method used in the analysis: the outlier detection tuning parameter, the distance computation method, and whether or not to include branch length information in the distance calculation. These are controlled by the parameters `k`, `distance`, and `topo.only`, respectively. By default the geodesic distance with branch lengths is used, with a tuning parameter of $k = 1.5$.

For example, this call uses topology-based dissimilarity map distance.

```
> kdetrees(apicomplexa, k=1.25, distance="diss", topo.only=TRUE)
```

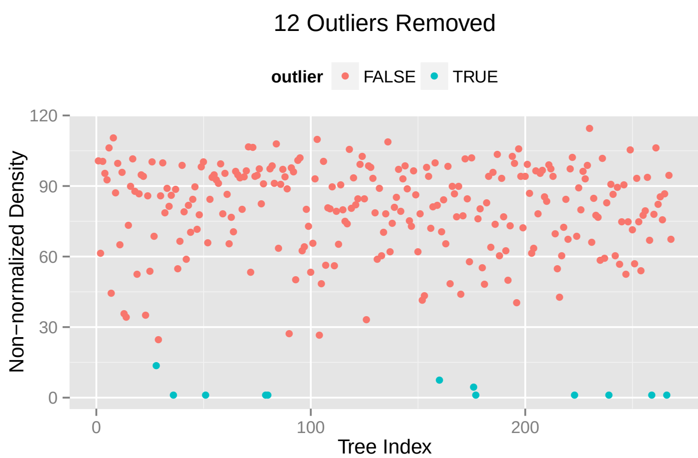
One can `plot` or `hist` the result object to create diagnostic plots. The `plot` and `hist` methods use the `ggplot2` package, not base graphics, thus you can modify them as you see fit. See Figure 1 for example plots.

2.3 Results

The result object is a list with three components, as well as several attributes that are used internally. The first element, `density`, has the computed score for each tree in the input list. This is the variable displayed in the diagnostic plots. The second elemen,t `i`, contains the indices of the low scoring trees which were not included in the calculations. Finally, the `outliers` element contains the trees which were identified as outliers.

One might then wish to look at a plot of the putative outlier trees. Here I plot the lowest scoring tree in the apicomplexa dataset. It appears that something

```
> plot(result)
```




```
> hist(result)
```

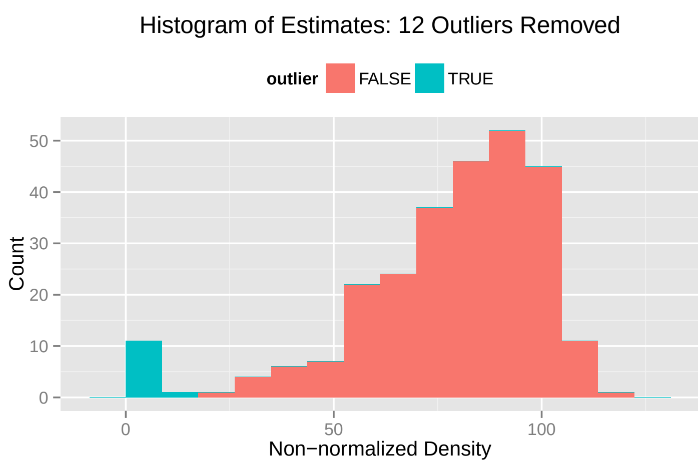


Figure 1: Diagnostic plots can be created with `plot` and `hist`.

```
> plot(result$outliers[[1]])
```

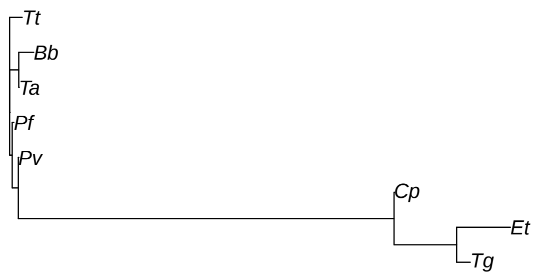


Figure 2: A plot of an outlying tree.

bad happened during the reconstruction of this tree, causing one branch to be much longer than the others.

If you would like to export the outlier trees to a file, you may use the `write.tree` function in the `ape` package. Data frames can also be created from the result object.

```
> write.tree(result$outliers,file="outliers.tre")
> as.data.frame(result)
```

2.4 Advanced Options

Currently, kdetrees uses an adaptive bandwidth method based on a nearest-neighbor calculation by default. It is possible to control the number of trees used to define the neighborhood, or disable the adaptive method entirely and provide a constant bandwidth, using the `bw` parameter. If `bw` is passed as a list, the list is used as a set of parameters for a call to `bw.nn`. For example, to change the neighborhood to include 50% of the sample we pass the following option.

```
> kdetrees(apicomplexa, bw=list(prop=0.5))
```

If `bw` is set to a single number, a constant bandwidth is used.

```
> kdetrees(apicomplexa, bw=6)
```

2.5 CLI

CLI use can be achieved by using the `Rscript` executable included with R. For example, this CLI command replicates the first example call in Section 2.

```
$ Rscript -e 'library(kdetrees); kdetrees.complete("trees.tre")'
```