Tutorial vignette for package 'specificity'

John L Darcy

April 07, 2020

Introduction

In this example analysis, our goal is to analyze the extent to which microbes have specificity to different aspects of their environment. We will use the 'endophyte' data set (included with the specificity R package), which contains data from a survey of foliar endophytic fungi living within the leaves of native Hawaiian plants. In this context, a fungus with strong specificity would be one that occupies a narrow range of sampled environment. For example, a fungus that preferentially associates with a narrow clade of host plants (e.g. asteraceae), or prefers a narrow range of elevation (e.g. between 700 and 1200 m.a.s.l). The previous two examples were for unimodal specificity, but specificity as calculated with our software can be multimodal as well (e.g. phylogenetic specificity to both asteraceae and malvaceae). It is worth noting that a strength of this approach is that specificity is agnostic to the ideal habitat of a species: a species with strong specificity to high elevations may be just as specific to elevation as a species specific to low elevations. Indeed, a strength of this approach is that the ideal habitat does NOT need to be modeled in order to calculate specificity. Using default options, we calculate specificity as an index that ranges from -1 to 1, with -1 indicating perfect specificity, 0 indicating no difference from the null model, and 1 indicating perfect cosmopolitanism (`denom_type = "index"`).

Software Requirements

- `R` v. 3.5.0 or higher
- Other dependencies will be automatically installed with the package:
  - ape
  - geiger (reason for 3.5.0 requirement - may remove this dependency)
  - fitdistrplus
  - Rcpp
  - fields

Installation

To install `specificity`, first we need the `remotes` package, which is part of `devtools`. You may already have it installed. Open up R and get it with:

```r
install.packages("remotes")
```

Now, activate `remotes` and install `specificity`.

```r
library("remotes")
install_github("darcyj/specificity")
```

Load 'endophyte' example data set

The endophyte dataset consists of foliar endophytic fungi sampled from leaves of native Hawaiian plants across the Hawaiian archipelago. These data can be loaded into R with:

```
# load specificity R package
suppressMessages(library(specificity))
# load endophyte data set from specificity
data(endophyte)
# check what objects are within endophyte
names(endophyte)
```

```
## [1] "otutable"  "metadata"  "supertree"
```

```
# add those objects to R namespace so we aren't typing "endophyte$" every time
attach(endophyte)
```

As you can see, there are 3 objects inside 'endophyte'.

- `metadata`: table of data where each row corresponds to a sample, and each column is a different metadata category (e.g. PlantGenus, Elevation, Lon=longitude, Lat=latitude)
- `otutable`: table of OTU (species) observation data, where each row is a sample, and each column is a different fungal OTU (really DADA2 ASVs, but those are a type of OTU).
- `supertree`: a phylogenetic tree of host plants.

Pre-processing of 'endophyte' data

The species 'abundance' values we feed to `specificity` are really just proxies for how confident we are that a species appeared in a given sample. Object `otutable` is a site-by-species matrix containing counts, but each site could have different total counts (in these example data, that isn't the case, but in your data it may be important). Here, we'll just transform the data to proportional abundance:

```
otutable <- prop_abund(otutable)
```

Like every tool, specificity is useless with a low sample size. In the case of specificity, this 'sample size' is the occupancy of a species, or how many samples that species was observed in. Here, we will create a new table containing only species that observed in at least 10 samples. NOTE: It's important we do this AFTER our proportional abundance calculation (above)! Otherwise the proportion of rare taxa in each sample would bias the proportional abundances.

```
# apply occupancy threshold to remove low-occupancy species
otutable_ovr10 <- occ_threshold(otutable, threshold=10)
# how many species are we left with?
ncol(otutable)
```

```
## [1] 3479
```

```
ncol(otutable_ovr10)
```

```
## [1] 416
```

```
# (there are MANY species in this dataset that are extremely rare...)
# (that's common for microbiome data.)
```

Note that we did not modify the rows (samples) of otutable! This is important, since they're paired with the rows (samples) in `metadata`. Just to make sure, let's check:

```
all(rownames(otutable_ovr10) == rownames(metadata))
```

```
## [1] TRUE
```

Specificity analysis

Calculating specificity for various data types is fairly easy with this package. In this example, we will analyze specificity to elevation, rainfall, host plant phylogeny, and geographic distance. In the function `phy_or_env_spec()`, the `n_sim` argument determines the number of simulations (i.e. permutations) to do. Each analysis below should take a few seconds to complete (maybe several minutes on laptop hardware). When you run them, you will see status updates that are omitted in this vignette for the sake of brevity. To make all of this run faster, we are only using 100 simulations, but recommend 1000 sims for a real analysis (default). `p_method = "gamma_fit"` means we fit a gamma distribution to permuted specificity values in order to calculate P-values for each species. The default is `p_method = "raw"`, which requires a much higher `n_sim` to get reasonable P-values. Funky variable distributions may break `gamma_fit` (you'll get an error message to that effect), but using `raw` with lots of permutations will always work.

`phy_or_env_spec()` is parallelized, and speeds up very nicely using multiple CPU cores. In the example below we'll use 10 CPU cores because that way this document gets compiled more quickly. For typical hardware, you should change it to 4 or so. If you're using a cluster, know that more cores = more faster.

One more thing to note is that here, we're placing each specificity result into a list object. That's because `plot_specificities` (used below) expects them to all be in a list. It's also a convenient way to organize our results.

```r
# set number of CPU cores to use in commands below - change for your system.
spec_ncores <- 10
# make empty list for specificity values
specs_list <- list()
# specificity for elevation:
specs_list$"Elevation" <- phy_or_env_spec(otutable_ovr10, env=metadata$Elevation,
  n_sim=100, n_cores=spec_ncores, p_method = "gamma_fit")
# specificity for rainfall
specs_list$"Rainfall" <- phy_or_env_spec(otutable_ovr10, env=metadata$Rainfall,
  n_sim=100, n_cores=spec_ncores, p_method = "gamma_fit")
# specificity for evapotranspiration
specs_list$"Evapotranspiration" <- phy_or_env_spec(otutable_ovr10, env=metadata$Evapotranspiration,
  n_sim=100, n_cores=spec_ncores, p_method = "gamma_fit")
# specificity for host phylogeny - converting tree to dist will take a while!
specs_list$"Host Phylogeny" <- phy_or_env_spec(otutable_ovr10, hosts=metadata$PlantGenus,
    hosts_phylo=supertree, n_sim=100, n_cores=spec_ncores, p_method = "gamma_fit")
```

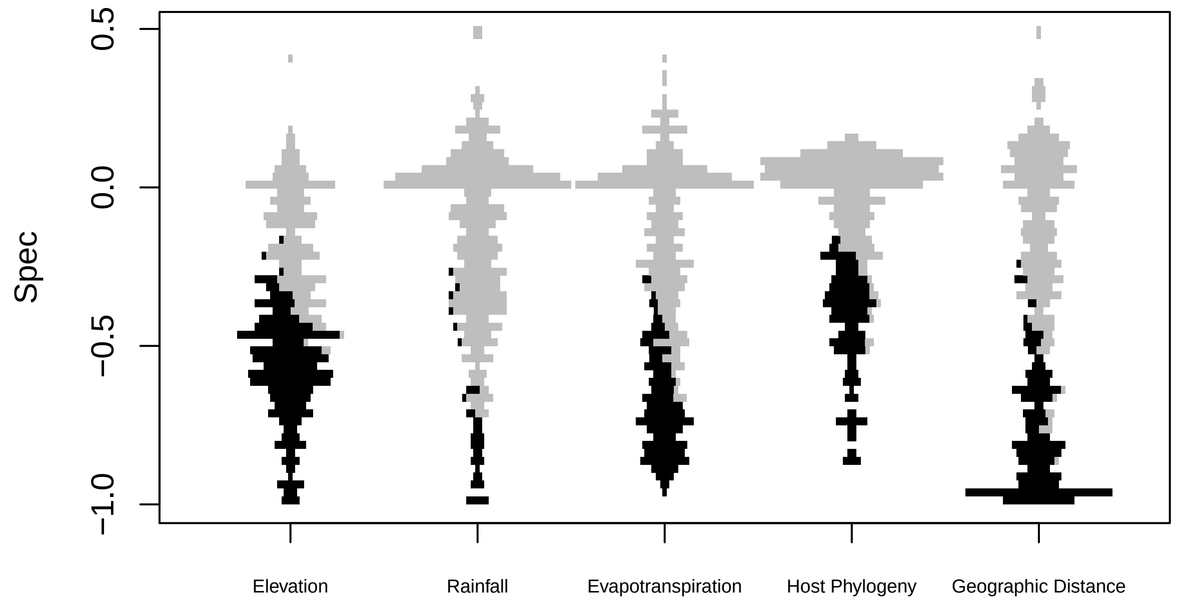
For geographic distance, we'll use specificity's `distcalc` function to calculate pairwise geographic distances between samples, using latitude and longitude data.

```r
# calculate geographic distances
geo_distmat <- distcalc(lat=metadata$Lat, lng=metadata$Lon, sampIDs=metadata$SampleID)
# specificity for geographic distance
specs_list$"Geographic Distance" <- phy_or_env_spec(otutable_ovr10, env=geo_distmat, n_sim=100,
    n_cores=spec_ncores, p_method = "gamma_fit")
```

Visualization

Our built-in visualization function takes a list of results from `phy_or_env_spec()` as input, and produces a violin plot composed of bars colored by statistical significance (dark = sig, light = nonsig). The names of items within `specs_list` will be the labels used in the plot.

```r
plot_specificities(specs_list, n_bins = 60)
```


Spec
Elevation
Rainfall
Evapotranspiration
Host Phylogeny
Geographic Distance