Tutorial vignette for package 'specificity'

John L Darcy

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 this 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 further strength 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.

Software Requirements

- `R` v. 3.5.0 or higher (untested on lower versions, but may still work?)
- Other dependencies will be automatically installed with the package:
  - ape
  - Rcpp
  - fields

Installation

There are two different ways to install `specificity`. The first way is to compile the package by installing from source, and is the best way to guarentee you're getting the most recent version. This requires a functioning installation of `Rcpp` though, which may be difficult for Mac OS users. The second way to install the package is with precompiled binaries, available at the specificity_builds repo. Instructions for installing precompiled builds can be found there, but note those builds get updated much less frequently than the source.

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

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

You'll also need `Rcpp`, which compiles C++ functions and makes specificity 10 to 20 times faster. It can be challenging to install on Mac OS, but it's easy to install on Linux systems. See the Rcpp install notes here: https://teuder.github.io/rcpp4everyone_en/020_install.html.

Now that we have `remotes`, we can install `specificity`.

```
  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))
# the version of specificity with which this vignette was built:
packageVersion("specificity")
```

```
## [1] '0.1.10.9000'
```

```
# check what objects are within endophyte
names(endophyte)
```

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

```
# attach "endophyte" data set from specificity
attach(endophyte)
```

As you can see, there are 4 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.
- `taxonomy`: taxonomic information for fungal OTUs.

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)
```

The HPC used to process these data originally had 24 threads and 128 gigabytes of RAM. Your machine may not be so impressive. This is just a random downsampling of the data, done only for demonstration purposes. Real data in a real analysis should not be downsampled.

```
  set.seed(12345)
  keep <- sample(c(rep(TRUE, 300), rep(FALSE, nrow(otutable) - 300)))
  otutable <- otutable[keep,]
  metadata <- metadata[keep,]
```

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] 168
```

```
# (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. 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). To get more accurate P-values with few simulations, we use `p_method = "gamma_fit"`. This 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. For typical hardware, use 4 or so. If you're using a cluster, know that more cores = more faster. On typical hardware, each of the below analyses should take a minute or two.

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

```
# set number of CPU cores to use in commands below - change for your system.
spec_ncores <- 4
# 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=500, n_cores=spec_ncores)

# specificity for rainfall
specs_list$"Rainfall" <- phy_or_env_spec(otutable_ovr10, env=metadata$Rainfall,
  n_sim=500, n_cores=spec_ncores)

# specificity for evapotranspiration
```

```
specs_list$"Evapotranspiration" <- phy_or_env_spec(otutable_ovr10,
  env=metadata$Evapotranspiration, n_sim=500, n_cores=spec_ncores )

# 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=500,
  n_cores=spec_ncores)
```

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

```
# 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=500, n_cores=spec_ncores)
```

Visualization - Built-in

Our built-in visualization function takes a list of results from `phy_or_env_spec()` as input, and produces a violin plot colored by statistical significance (dark = sig, light = nonsig). The names of items within `specs_list` will be the labels used in the plot (Figure 1).

```
plot_specs_violin(specs_list, label_cex=0.6, cols=c("red", "blue", "gold", "green", "black"))
```

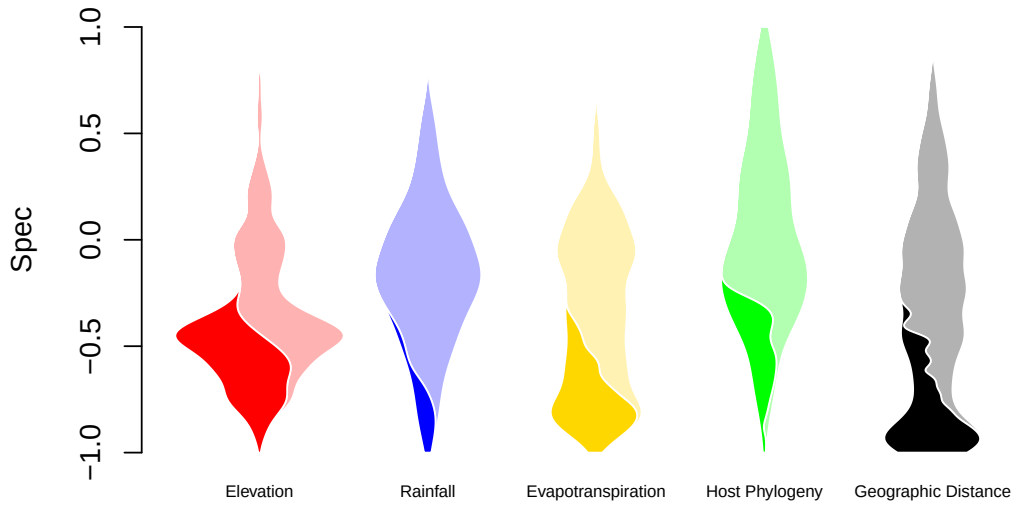

Figure 1: Specificity violin plot

We can also see how species specificity correlates between variables. From this, it looks like for most species,

specificity to geographic distance strongly correlates with specificity to evapotranspiration (Figure 2)

```
plot_pairwise_spec(specs_list)
```

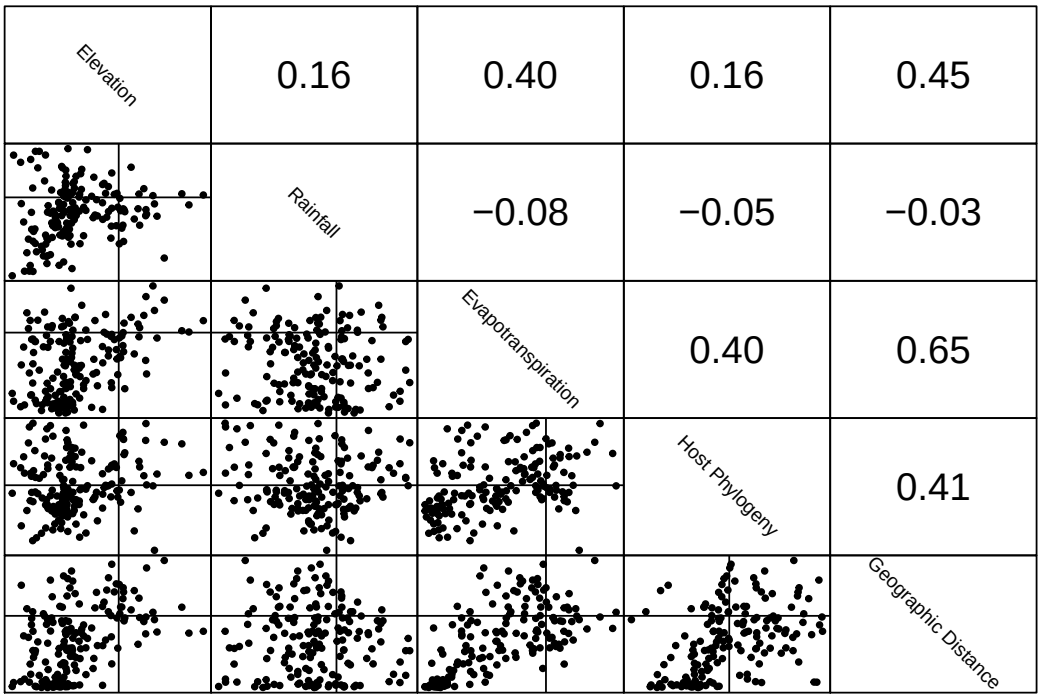

Figure 2: Pairwise specificity plot

Visualization - ggplot

The popular `ggplot2` package can also be used to visualize data generated with `specificity` (Figure 3). To do this, first data must be transformed from our `specs_list` object to a long-format `data.frame`. Other species information (e.g. taxonomy) can be optionally combined into this object as well. In this example, we'll include (but not use) fungal taxonomic information.

```
specs_df <- aggregate_specs_list(specs_list, byFeature=FALSE, fd=taxonomy, fd_id=1)
```

Now we'll load up ggplot and make a plot similar to the plot made above:

```
library(ggplot2)
ggplot(specs_df, aes(x=Variable, y=Spec, fill=Variable)) +
  geom_violin() +
  geom_jitter(width=0.2, size=0.2)
```

Interactive Visualization with specificity.shiny

`specificity`'s companion package, `specificity.shiny`, creates interactive visualizations where individual species can be selected and compared between variables. The following requires `specificity.shiny`, which can be installed as follows:

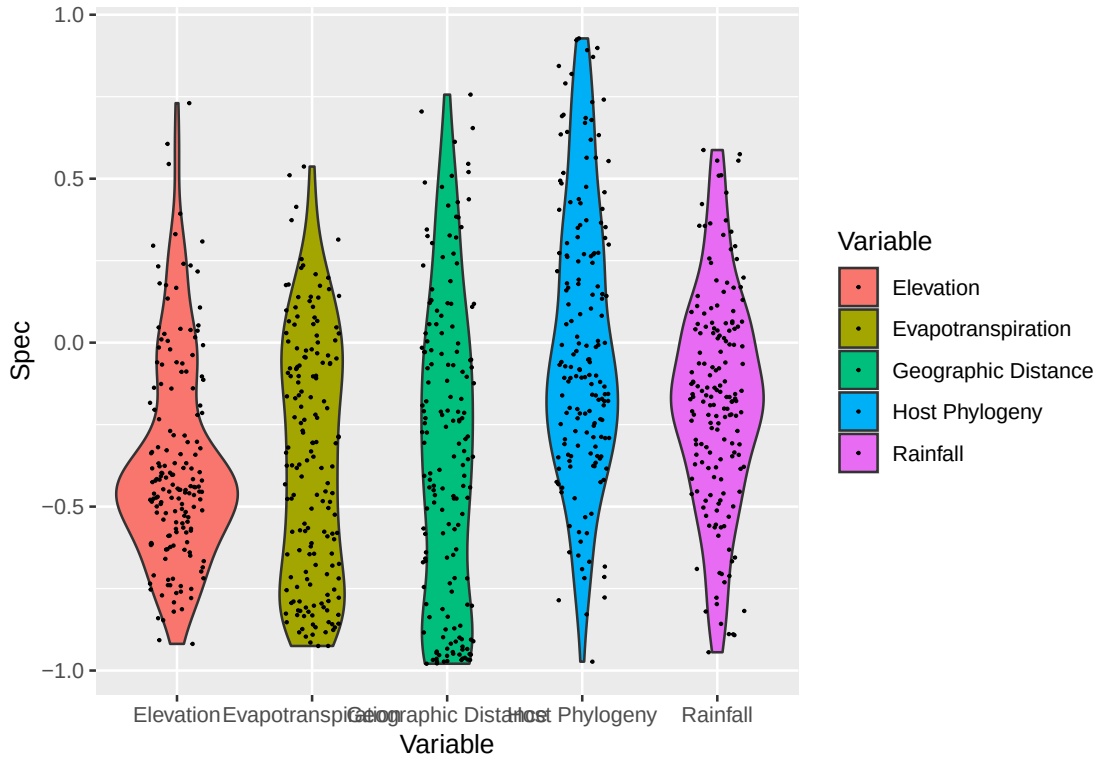

Figure 3: Specificity violins using ggplot

```
remotes::install_github("darcyj/specificity.shiny")
```

Now we can use the functions from `shiny` and from `specificity.shiny` to run an interactive visualization. Feature data (`fd`) are supplied as a data.frame object that has some column of feature IDs. The default operation of this function is to assume that the first column contains those IDs, which must match with the IDs in your specs_list (`sl`) object.

```
specificity.shiny::plot_specs_shiny(sl=specs_list, fd=endophyte$taxonomy)
```

That command creates and opens upp the visualization in your browser (Figure 4). In the visualization, points can be selected by dragging a rectangle around them. When you do so, those same features are hilighted across other variables. See the example screenshot below, where species with strong specificity to rainfall are selected. Feature data for those species, in this case including taxonomic information, are shown in the table at bottom. That table, or the raw data, can be downloaded using the "Download" button.

The search feature can be used to only show features that match a certain string (Figure 5). By setting the search field to a column of the feature data supplied when making the visualization, plotted points can be subset to only show features that match your search string. Note that background violins will not be altered, only the points shown. This is especially useful when searching for species that match a certain taxonomic query, as in the example below where only fungi in the Basidiomycota are displayed. Those features can then be hilighted as above, to see their details.

Hosting Your Interactive Visualization on shinyapps.io

The visualization can also be made portable, so that it can be run on servers such as `shinyapps.io`, or simply run again locally without re-running anything else. Any visualization that is successful using the

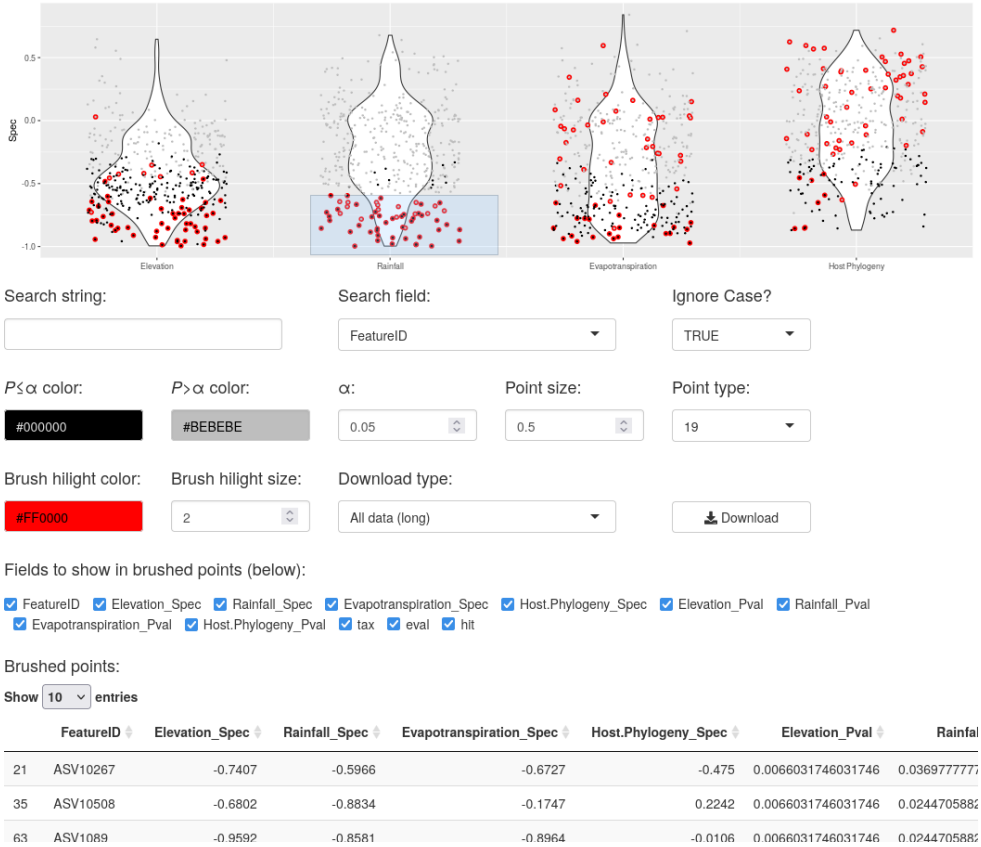

Figure 4: Selecting points in specificity.shiny

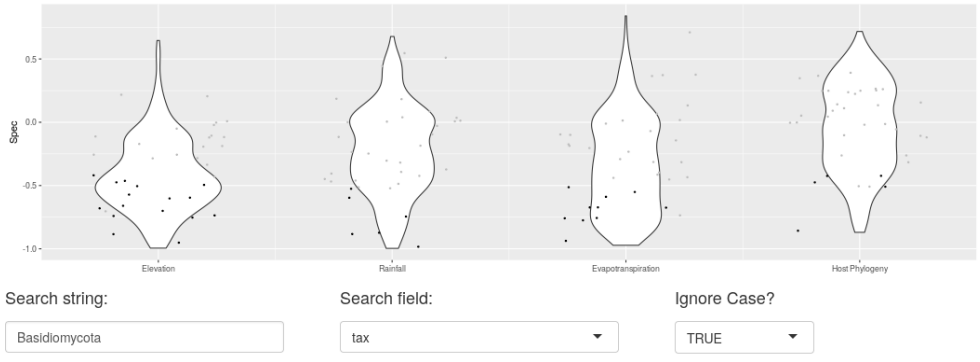

Figure 5: Searching by string match in specificity.shiny

above approach can be made into a portable app. The only additional argument required is the name of the folder that should contain the app's data (we refer to this folder as the app). The output of the following command can be found with this vignette.

```
specificity.shiny::make_specs_app(sl=specs_list, fd=endophyte$taxonomy, app_fp="endophytes_specs_app")
```

You can run the app locally using shiny:

```
shiny::runApp("endophytes_specs_app")
```

Or upload it to `shinyapps.io` so others can use your visualization online. See their guide on uploading here. Basically, set up your account on that website, install the `rsconnect` package, and then set your account info in R using:

```
rsconnect::setAccountInfo(
  name="your account name here",
  token="your token here",
  secret="your secret here"
)
```

Then, simply uppload your app using `rsconnect::deployApp`. This will automatically open up your app in a web browser, so you can copy the URL and send it to collaborators. Note that unlike before, your app will be running on a remote server, not locally as before.

```
rsconnect::deployApp("endophytes_specs_app")
```