Using spectra to predict biodiversity

Anna K. Schweiger and Jesús N. Pinto-Ledezma

In this lab, we will explore some aspects of **Spectral data** and how it can be used to predict biodiversity. More specifically, we will use spectra to predict chemical components in some plant species. To do so, we will use the R packages **spectrolab** (Meireles et al. 2018) and **pls** (Mevik and Wehrens 2007). **spectrolab** is a R package that process and visualize data from portable spectrometers and establishes a common interface to spectra information and **pls** is a package designed to perform some cousins of the linear regression, such as, Principal Component Regression (**PCR**) and Partial Least Squares Regression (**PLSR**). Note that Anna give us an explanation of how PLS models works. As you will see, working with spectral information/data is similar to work with any other information (e.g., species) and consequently it can be used to calculate any metric of diversity. Thus **Spectral diversity** can be considered a dimension of biodiversity.

Set up your data and your working directory

Set up a working directory and put the two data files in that directory. Tell R that this is the directory you will be using, and read in your data:

```
setwd("path/for/your/directory")
```

Install and load the following packages.

```
library(devtools) # Tools to Make Developing R Packages Easier
devtools::install_github("meireles/spectrolab") ### or from CRAN, new version coming soon
library(spectrolab) # Class and Methods for Hyperspectral Data
install.packages("pls")
library(pls) # Partial Least Squares and Principal Component Regression
```

Spectral data processing

First, we will explore how to process spectral data.

There are two ways to get spectra into R: 1) converting a matrix or data.frame to spectra; 2) reading spectra from raw data files (formats: SVC's sig, Spectral Evolution's sed and ASD's asd).

Here is an example using a dataset matrix named **spec_data_for_preds.csv**.

```
### Create spectra from a matrix or data.frame
# Read data from the CSV file. If you don't use `check.names` = FALSE when reading
# the csv, R will usually add a letter to the column names (e.g. 'X450') which will
# cause problems when converting the matrix to spectra.
dada <- read.csv("Data/Spectra/spec_data_for_preds.csv", check.names = FALSE)
# remember to locate the data inside the main folder Data
```

Inspect the data.

```r
head(dada)
```

```r
dim(dada)
str(dada)
```

Once you have your spectra in a matrix or data frame in R, you can use the function **as.spectra()** to convert it to a spectra object. The matrix must have samples in rows and wavelengths in columns. The header of the wavelengths columns must be (numeric) wavelength labels. You also should declare which column has the sample names (which are mandatory) using the name_idx argument. If other columns are present (other than sample name and reflectances), their indices must be passed to as.spectra as the meta_idxs argument.

```r
# The sample names is in column 1.
dada_spec <- as.spectra(dada, name_idx = 1) ### make spectra object
# And now you have a spectra object with sample names and metadata...
dada_spec
```

Awesome, we just created our first spectral object. Now let's see how it looks like. This is an interactive plot and you can play a little with it.

```r
plot_interactive(dada_spec) ### look at spectra
```

The next step is to normalize the spectral data in order to remove differences in illumination. To do that we just need to use the function normalice.

```r
spec_vn <- normalize(dada_spec) ### model is build with vector normalized spectra, length of each spect
```

See the results

```r
plot(spec_vn[1:10], col = rainbow(10))
```

The spectra can be transformed back to a data frame just using the base function as.data.frame().

```r
# The spectra can be transformed to a data frame just using the base function as.data.frame
preds <- as.data.frame(spec_vn)
head(preds)
```

Inspecting and querying spectra

You can check out your spectra object in several ways. For instance, You may want to know how many spectra and how many bands are in there, retrieve the file names, etc. Of course you will need to plot the data, but that topic gets its own section further down.

```r
# Simply print the object
spec_vn
```

```r
# Get the dataset's dimensions
dim(spec_vn)
```

You can access to the individual components of the spectra. This is done with the functions **names()** for sample names, **wavelengths()** for wavelength labels, **reflectance()** for the reflectance matrix, and **meta()** for the associated metadata (in case you have any).

```r
# Vector of all sample names. Note: Duplicated sample names are permitted
n <- names(spec_vn)
# Vector of wavelengths
w <- wavelengths(spec_vn)
# Reflectance matrix
r <- reflectance(spec_vn)
```

```
# Metadata. Use simplify = TRUE to get a vector instead of a data.frame
m <- meta(spec_vn, simplify = TRUE)
```

Subsetting spectra

You can subset the spectra information using a notation similar to the [i, j] function used in matrices and data.frames. The first argument in [i,] matches sample names (rows), whereas the second argument [, j] matches the wavelength names (columns). Here are some examples of how [works in spectra:

- x[1:3,] will keep the first three samples of x, i.e. 1:3 are indexes.
- x["sp_1",] keeps all entries in x where sample names match "sp_1".
- x[, 800:900] will keep wavelengths between 800 and 900.
- x[, 1:5] will fail!. wavelengths cannot be subset by indexes!.

Subsetting lets you, for instance, exclude noisy regions at the beginning and end of the spectrum or limit the data to specific entries.

```
# Subset wavelength regions.
spec_sub_vis <- spec_vn[, 400:700]
# Subset spectra to all entries where sample_name matches "indiv_7" or get the first three samples
spec_sub_byname <- spec_vn["indiv_7", ]
spec_sub_byidx <- spec_vn[1:3, ]
```

Note that you can (1) subset samples using indexes and (2) use characters or numerics to subset wavelengths. As we said before, you cannot use indexes to subset wavelengths though.

```
# Subsetting samples by indexes works and so does subsetting wavelengths by numerics or characters.
spec_sub_byidx[1, "405"] == spec_sub_byidx[1, 405]
```

As you seen until now, working with spectra is similar to work with any data.frame or matrix. Now, lets see how spectra looks like is…

Plotting

The main function for plotting spectra is **plot()**. It will jointly plot each spectrum in the spectra object. You should be able to pass the usual plot arguments to it, such as col, ylab, lwd, etc. Lets see…

```
# Simple spectra plot
plot(spec_vn, lwd = 0.75, lty = 1, col = "grey25", main = "All Spectra")
```

You can also plot the quantiles of a spectra object with **plot_quantile()**. It's second argument, total_prob, is the total "mass" that the quantile encompasses. For instance, a total_prob = 0.95 covers 95% of the variation in the spectra object, i.e. it is the 0.025 to 0.975 quantile. The quantile plot can stand alone or be added to a current plot if add = TRUE.

```
# Stand along quantile plot
plot_quantile(spec_vn, total_prob = 0.8, col = rgb(1, 0, 0, 0.5), lwd = 0.5, border = TRUE)
title("80% spectral quantile")
```

The function **plot_regions()** helps shading different spectral regions. spectrolab provides a default_spec_regions() matrix as an example, but you obviously can customize it for your needs (see the help page for plot_regions for details).

```r
# Combined individual spectra, quantiles and shade spectral regions
plot(spec_vn, lwd = 0.25, lty = 1, col = "grey50", main = "Spectra, quantile and regions")
plot_quantile(spec_vn, total_prob = 0.8, col = rgb(1, 0, 0, 0.25),
              border = FALSE, add = TRUE)
plot_regions(spec_vn, regions = default_spec_regions(), add = TRUE)
```

Converting a spectra object into a matrix or data.frame

It is also possible to convert a spectra object to a matrix or data.frame using the **as.matrix()** or **as.data.frame()** functions. This is useful if you want to export your data in a particular format, such as csv.

```r
# Make a matrix from a `spectra` object
spec_as_mat = as.matrix(spec_vn, fix_names = "none")
spec_as_mat[1:4, 1:3]
```

```r
# Make a matrix from a `spectra` object
spec_as_df = as.data.frame(spec_vn, fix_names = "none", metadata = TRUE)
spec_as_df[1:4, 1:5]
```

Ok, enough of exploring and plotting spectra. Now we will use spectra to calculate some descriptive statistics and to calculate some diversity metrics.

Using spectra

Note that spectra goes from 400 to 2400 bands, following we will separate the spectra data into different spectrum. These are the **VIS** or Visible, **NIR** or Near Infrared, **SWIR1** and **SWIR2** or Short-wave Infared Imagery.

```r
VIS <- c(400:700)
NIR <- c(800:1300)
SWIR1 <- c(1550:1800)
SWIR2 <- c(2000:2400)
```

Now select and extract the bands corresponding to the VISIBLE spectrum.

```r
spec_VIS <- spec_vn[, VIS]
spec_VIS
```

```r
plot(spec_VIS, lwd = 0.75, lty = 1, col = "grey25", main = "Just VIS")
```

Cool, right? you can repeat the process for the other three spectrum.

Drop the first 2 columns for further analyses. Note that the rows are maintained in the same position, so you can bind the names of the species or individuals.

```r
spec_as_df_clean <- spec_as_df[, 3:2001]
spec_as_df_clean[1:10, 1:5]
```

Ok, now lets calculate some descriptive statistics to better understand the data. You can do by yourself, but, Jesús prepared a function to make this calculation automatically. The function was called **Descriptives()** and you just need to enter a vector of values.

```r
source("https://raw.githubusercontent.com/jesusNPL/BiodiversityScience/master/Spring2019/RFunctions/mix
```

Just for fun select one individual.

```r
ind1 <- spec_as_df_clean[1, ]
```

Using the selected individual we can estimate some descriptives

```r
Descriptives(as.numeric(ind1))
```

Now, you can repeat the process for the other individuals and see the differences with the other individuals in the dataset. Notice that the function **Descriptives()** can be used within a **for** loop, but also you can use the **apply** function. This function apply a specific function to a matrix or data.frame.

```r
descript_all_individuals <- apply(spec_as_df_clean, MARGIN = 1, FUN = Descriptives)
descript_all_individuals[[10]] # these are the descriptes for the individual number 10
```

Vegetation indices

Finally, we will calculate some vegetation indices. Until now, we explored how to handle spectra data and extract some descriptive statistics, so, you are now capable to calculate some simple vegetation indices.

1. Simple Ratio: SR = Rnir/Rr Where: R = reflectance, nir = band 845 and r = band 665.
2. Normalized Difference Vegetation Index: NDVI = (Rnir - Rr)/(Rnir + Rr) Where: nir = 845 and r = 665.
3. Photochemical Reflectance Index: PRI = (R531 - R570)/(R531 + R570). The numbers corresponde to the 531 and 570 bands, respectively and R is the reflectance.
4. Normalized Difference Water Index: NDWI = (R860 - R1240)/(R860 + R1240)
5. Water Balance Index: WBI = R900/R970.

Here is an example:

```r
#head(spec_as_df_clean)
SR_spec = spec_as_df[, "845"]/spec_as_df[, "665"]
SR_spec
```

Exercises part I

Do the calulations for the other four vegetation indices and using the function **Descriptives()** explain the results.

Using spectra to predict biodiversity

Building a PLSR model to predict traits from spectra

Now we will explore the power of spectra in predicting plant chemical components using the statistical machinery of the PLS models. You can clean up your Global Environment using **rm(list = ls())**, as we will use othe data.

Using the data **spec__chem__fiber** we will predict non-structural carbohydrates using spectra.

```r
allData <- read.csv("Data/Spectra/spec_chem_fiber.csv")
```

```r
dim(allData)
class(allData)
```

Resample the data to make model run faster (max interval 20 nm) and limit wavelength range to wavelegths sensitive to the chemical compound of interest.

```
spec_inter <- 20
allData <- allData[, c(1:6, seq(which(names(allData) %in% "X1200"),
                              which(names(allData) %in% "X2400"),
                              by = spec_inter))]
```

Some other arrangements in the data.

```
### setup dataset
dat <- data.frame(SampN = c(1:nrow(allData)))
dat$abbrev <- allData$abbrev ## optional info to keep
dat$NSC <- allData$nonstructural_perc ## select trait
dat$spec <- as.matrix(allData[, grepl("X", names(allData))]) ## add spectral data as matrix
dat <- na.exclude(dat)
```

Well, finally we will use the statistical machinery of PLS to predict non-structural carbohydrates (NSC) using spectra.

```
# PLSR
mod <- plsr(NSC ~ spec, ncomp = 25, data = dat, validation = "LOO")
summary(mod)
```

Now, using validation plots we can explore how well spectra is predicting NSC.

```
validationplot(mod, val.type = "RMSEP", estimate = "CV") # The lower the RMSEP the better
```

```
validationplot(mod, val.type = "R2", estimate = "CV")
```

Select optimal number of components using RMSEP minimum and R2 maximum. By inspecting the two previous plots we can set to 12 the numbers of components necessary for model predictions.

```
compi <- 12 ##
```

Create object to predict NSC.

```
b <- predplot(mod, ncomp = compi, which = "train")
a <- predplot(mod, ncomp = compi, which = "validation")
```

```
dat_predtrue <- data.frame(abbrev = character(nrow(dat)))
dat_predtrue$abbrev <- dat$abbrev
dat_predtrue$measured <- a[, colnames(a) == "measured"]
dat_predtrue$predicted_val <- a[, colnames(a) == "predicted"]
dat_predtrue$predicted_train <- b[, colnames(b) == "predicted"]
```

```
dim(dat_predtrue)
head(dat_predtrue)
```

Model validation using R2

```
mod_fit <- lm(measured ~ predicted_val, data = dat_predtrue)
summary(mod_fit)
```

It seems that spectra is predicting NSC relatively well, **R2 = 0.59**. Let's plot the results

```
plot(measured ~ predicted_val, data = dat_predtrue, ylab = "Measured", xlab = "Predicted")
abline(mod_fit, col = ("red"), lwd = 2)
abline(0, 1, lty = 2, lwd = 2)
```

Let's explore more statistical details by testing if the slope is significantly different from 1. In this case we want the slope not to be different from one.

```r
### Test if slope is sign diff from 1
mod1 <- nls(measured ~ k*predicted_val + d, data = dat_predtrue,
            start = list(k = 1, d = 0))
## mod1 ... alternative mod, fixed slope at 1 and intercept at 0
mod0 <- nls(measured ~ predicted_val + d, data = dat_predtrue, start = list(d = 0))
## mod0 ... our model, but intercept fixed at 0 (only look at deviation of slope)
anova (mod1, mod0)
```

Effectively, our slope is not different from one, so that is good!

Now, test if intercept is significantly different from zero.

```r
mod0d <- nls(measured ~ k*predicted_val, data = dat_predtrue, start = list(k = 1)) ## mod0 ... our mode
anova (mod1, mod0d) ## we want no sign. diff
```

Now using **Root Mean Square Error** let's explore the differences between the values predicted by our model and the observed values. Remember, the lower the RMSE the better.

```r
### Coefficients for predictions #####
coefNSC <- coef(mod, ncomp = compi, intercept = T)

#### RMSEP, root mean squared error of predictions, in orig units ####
rmse <- function(obs, pred) {sqrt(mean((obs-pred)^2))}
(rm <- rmse(dat_predtrue$measured, dat_predtrue$predicted_val))
### on average pred are x % off
```

Loadings plot, importance of wavelengths, can be improved by showing absolute values

```r
wvl <- as.numeric(substr(colnames(dat$spec), 2, nchar(colnames(dat$spec))))
loadingplot(mod,comps = compi, xaxt = "n", xlab = "wavelength")
axis(1, seq(0, length(wvl), by = 10), labels = seq(wvl[1], wvl[length(wvl)],
                                                   length.out = 7))
```

Extract model coefficients and look at them.

```r
(NSC_coef <- coef(mod, ncomp = compi, intercept = T))
```

Predicting traits from spectra

Finally, let's use our model coefficients to predict traits from spectra. To do that, we will use the first data set.

```r
dada <- read.csv("Data/Spectra/spec_data_for_preds.csv", check.names = F)
dada_spec <- as.spectra(dada, name_idx = 1) # make spectra object
spec_vn <- normalize(dada_spec) # model is build with vector normalized spectra,
# length of each spectrum =  unity, to remove illumination differences
plot(spec_vn[1:10], col = rainbow(10))
spec_pred <- resample(spec_vn, new_wvls = seq(1200, 2400, 20))
```

```r
meta(spec_pred, "NSC") <- reflectance(spec_pred) %*% coefNSC[-1] + coefNSC[1]
```

```r
meta(spec_pred, "NSC")
```

```r
preds <- as.data.frame(spec_pred)
head(preds)
```

There you have it, we predicted traits from spectra.

Exercises part II

In the previous analyses we explored how spectra can be using to predict non-structural carbohydrates (NSC). In the data set **spec_chem_fiber.csv** there are other two chemical components **hemicellulose** and **cellulose**, the challenge is to predict these two components using spectra as we did previously with NSC.

References

Mevik B-H. and Wehrens R. (2007). The pls package: Principal component and partial least squares regression in R. Journal of Statistical Software, 18(2):1–24.

Meireles JE., Schweiger AK and Cavender-Bares J. (2018). spectrolab: Class and Methods for Hyperspectral Data. R package version 0.0.8. https://github.com/meireles/spectrolab

Further readings

If you are interested in digging more about the PLS models, these two papers can help you!

Wold S., Martens H. and Wold H. (1983). The multivariate calibration problem in chemistry solved by the PLS method. In Ruhe A, Kagstrom B. Matrix pencils, Lecture Notes in Mathematics: Springer, Heidelberg. 286-293.

Martens H. (2001). Reliable and relevant modelling of real world data: a personal account of the development of PLS regression. Chemometrics and intelligent laboratory systems 58(2): 85-95.