Using spectra to predict biodiversity

Jesús N. Pinto-Ledezma and Jeannine Cavender-Bares

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. When installing the package {rhdf5} R will ask the next **"Update all/some/none? [a/s/n]:"** please in your console type **n**.

```
packages <- c("maptools", "rgdal", "raster", "neonUtilities", "rasterdiv",
              "BiocManager", "dplyr", "tidyr", "ggplot2", "plyr", "reshape2")

# Install packages not yet installed
installed_packages <- packages %in% rownames(installed.packages())

if (any(installed_packages == FALSE)) {
  install.packages(packages[!installed_packages], dependencies = TRUE)
}

BiocManager::install("rhdf5")
```

```
sapply(packages, require, character.only = TRUE)
library(rhdf5)
```

DATA# 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**.

```
dir.create("Data/NEON")
```

Prepare data

```
NEON_plots <- readOGR(dsn = "Data/NEON/All_NEON_TOS_Plots_v7",
                      layer = "All_NEON_TOS_Plot_Polygons")
```

```
HARV_plots <- NEON_plots@data %>%
  filter(siteID == "HARV" & plotType == "distributed" & subtype == "basePlot") %>%
  arrange(plotID)

View(HARV_plots)
```

```
east <- HARV_plots$easting
names(east) <- HARV_plots$plotID

north <- HARV_plots$northing
names(north) <- HARV_plots$plotID

east
north
```

Download hyperspectral imagery from NEON AOP

```
coords_HARV_001 <- c(east[1], north[1])

byTileAOP(dpID = "DP3.30006.001", # NEON-AOP product
          site = "HARV", # Site code
          year = "2018", # Year
          check.size = TRUE,
          easting = coords_HARV_001[1], northing = coords_HARV_001[2], # Coordinates UTM
          savepath = "Data/NEON", # Path
          token = NA)
```

Data Exploration

```
f <- "Data/NEON/DP3.30006.001/2018/FullSite/D01/2018_HARV_5/L3/Spectrometer/Reflectance/NEON_D01_HARV_D
```

```
# Look at the HDF5 file structure
View(h5ls(f, all = TRUE))
```

```
# Get information about the wavelengths of HARV plot 001
wlInfo <- h5readAttributes(f, "/HARV/Reflectance/Metadata/Spectral_Data/Wavelength")

wlInfo
```

```
# Read wavelengths from the HDF5 file
WL <- h5read(f, "/HARV/Reflectance/Metadata/Spectral_Data/Wavelength")

head(WL)
tail(WL)
```

```
# Extract reflectance metadata
reflInfo <- h5readAttributes(f, "/HARV/Reflectance/Reflectance_Data")

reflInfo
```

```
# Read dimensions of the hyperspectral data
nRows <- reflInfo$Dimensions[1]
nCols <- reflInfo$Dimensions[2]
nBands <- reflInfo$Dimensions[3]
```

```
# Extract or "slice" data for band 9 from the HDF5 file
b34 <- h5read(f, "/HARV/Reflectance/Reflectance_Data",
              index = list(34, 1:nCols, 1:nRows))

# what type of object is b34?
class(b34)

## [1] "array"
```

```
# convert from array to matrix by selecting only the first band
b34 <- b34[1,,]

# check it
class(b34)

# plot the image
image(b34)
```

```
# oh, that is hard to visually interpret.
# what happens if we plot a log of the data?
image(log(b34))
```

```
# there is NO data value in our raster - let's define it
myNoDataValue <- as.numeric(reflInfo$Data_Ignore_Value)
myNoDataValue

# set all values equal to -9999 to NA
b34[b34 == myNoDataValue] <- NA

# plot the image now
image(b34)
```

```
# We need to transpose x and y values in order for our
# final image to plot properly
b34 <- t(b34)
image(log(b34), main = "Transposed Image")
```

```
# Extract the EPSG from the h5 dataset
myEPSG <- h5read(f, "/HARV/Reflectance/Metadata/Coordinate_System/EPSG Code")

# convert the EPSG code to a CRS string
myCRS <- crs(paste0("+init=epsg:", myEPSG))

# define final raster with projection info
# note that capitalization will throw errors on a MAC.
# if UTM is all caps it might cause an error!
b34ras <- raster(b34, crs = myCRS)

# view the raster attributes
b34ras
```

```
image(log(b34ras),
      xlab = "UTM Easting",
      ylab = "UTM Northing",
      main = "Properly Oriented Raster")
```

```
# Grab the UTM coordinates of the spatial extent
xMin <- reflInfo$Spatial_Extent_meters[1]
xMax <- reflInfo$Spatial_Extent_meters[2]
yMin <- reflInfo$Spatial_Extent_meters[3]
yMax <- reflInfo$Spatial_Extent_meters[4]

# define the extent (left, right, top, bottom)
rasExt <- extent(xMin, xMax, yMin, yMax)

# view the extent to make sure that it looks right
rasExt
```

```
# assign the spatial extent to the raster
extent(b34ras) <- rasExt

# look at raster attributes
b34ras
```

```
# write out the raster as a geotiff
writeRaster(b34ras,
            file = "Data/NEON/DP3.30006.001/HARV_plot_001_band_34.tif",
            format = "GTiff",
            overwrite = TRUE)
```

Creating a RGB raster

```
# file: the hdf file
# band: the band you want to process
# noDataValue: values to be omitted
# extent: raster extent
# CRS: coordinates system
# returns: a matrix containing the reflectance data for the specific band

band2Raster <- function(file, band, noDataValue, extent, CRS){
    # first, read in the raster
    out <- h5read(file,"/HARV/Reflectance/Reflectance_Data", index = list(band, NULL, NULL))
      # Convert from array to matrix
      out <- (out[1,,])
      # transpose data to fix flipped row and column order
    # depending upon how your data are formatted you might not have to perform this
    # step.
      out <- t(out)
    # assign data ignore values to NA
    # note, you might chose to assign values of 15000 to NA
    out[out == myNoDataValue] <- NA

    # turn the out object into a raster
    outr <- raster(out, crs = CRS)
```

```
    # assign the extents to the raster
    extent(outr) <- extent

    # return the raster object
    return(outr)
}
```

```
# create a list of the bands we want in our stack
rgb <- list(58, 34, 19) #list(58,34,19) when using full NEON hyperspectral dataset

# lapply tells R to apply the function to each element in the list
rgb_harv <- lapply(rgb, FUN = band2Raster, file = f,
                   noDataValue = myNoDataValue,
                   extent = rasExt,
                   CRS = myCRS)

# check out the properties or rgb_rast
# note that it displays properties of 3 rasters.
rgb_harv
```

```
# Create a raster stack from our list of rasters
rgb_harv_stack <- stack(rgb_harv)
rgb_harv_stack
```

```
# Create a list of band names
bandNames <- paste("Band_", unlist(rgb), sep = "")

# set the rasterStack's names equal to the list of bandNames created above
names(rgb_harv_stack) <- bandNames

# check properties of the raster list - note the band names
rgb_harv_stack

# scale the data as specified in the reflInfo$Scale Factor
rgb_harv_stack <- rgb_harv_stack/as.integer(reflInfo$Scale_Factor)

# plot one raster in the stack to make sure things look OK.
plot(rgb_harv_stack$Band_58, main = "Band 58")
```

```
# create a 3 band RGB image
plotRGB(rgb_harv_stack,
        r = 1, g = 2, b = 3,
        stretch = "lin")
```

```
# write out final raster
# note: if you set overwrite to TRUE, then you will overwite or lose the older
# version of the tif file! Keep this in mind.
writeRaster(rgb_harv_stack, file = "Data/NEON/DP3.30006.001/HARV_plot_001_RGB.tif",
            format = "GTiff", overwrite = TRUE)
```

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:

There you have it, we predicted traits from spectra.

```
# Calculate NDVI
# select bands to use in calculation (red, NIR)
ndvi_bands <- c(58, 90) #bands c(58, 90) in full NEON hyperspectral dataset

# create raster list and then a stack using those two bands
ndvi_harv <- lapply(ndvi_bands, FUN = band2Raster, file = f,
                    noDataValue = myNoDataValue,
                    extent = rasExt, CRS = myCRS)

ndvi_harv <- stack(ndvi_harv)

# make the names pretty
bandNDVINames <- paste("Band_", unlist(ndvi_bands), sep = "")
names(ndvi_harv) <- bandNDVINames

# view the properties of the new raster stack
ndvi_harv
```

```
#calculate NDVI
NDVI_func <- function(ras) {
      (ras[,2] - ras[,1])/(ras[,2]+ras[,1])
}

ndvi_calc <- calc(ndvi_harv, fun = NDVI_func)

plot(ndvi_calc, main = "NDVI for the NEON HARV Field Site")
```

```
# Now, play with breaks and colors to create a meaningful map
# add a color map with 4 colors
myCol <- rev(terrain.colors(4)) # use the 'rev()' function to put green as the highest NDVI value
# add breaks to the colormap, including lowest and highest values (4 breaks = 3 segments)
brk <- c(0, .25, .5, .75, 1)

# plot the image using breaks
plot(ndvi_calc, main = "NDVI for the NEON HARV Field Site", col = myCol, breaks = brk)
```

```
writeRaster(ndvi_calc, file = "Data/NEON/DP3.30006.001/HARV_plot_001_NDVI.tif",
            format = "GTiff", overwrite = TRUE)
```

Plot Spectral Signatures Derived from Hyperspectral Remote Sensing Data

Next, we will extract all reflectance values for one pixel. This makes up the spectral signature or profile of the pixel. To do that, we'll use the h5read() function. Here we pick an arbitrary pixel at (100,35), and use the NULL value to select all bands from that location.

```
# extract all bands from a single pixel
aPixel <- h5read(f, "/HARV/Reflectance/Reflectance_Data", index = list(NULL, 100, 35))

# The line above generates a vector of reflectance values.
# Next, we reshape the data and turn them into a dataframe
b <- adply(aPixel, c(1))

# create clean data frame
aPixeldf <- b[2]

# add wavelength data to matrix
aPixeldf$Wavelength <- WL

head(aPixeldf)
```

```
# grab scale factor from the Reflectance attributes
scaleFact <- reflInfo$Scale_Factor

# add scaled data column to DF
aPixeldf$scaled <- (aPixeldf$V1/as.vector(scaleFact))

# make nice column names
names(aPixeldf) <- c('Reflectance', 'Wavelength', 'ScaledReflectance')

head(aPixeldf)
tail(aPixeldf)
```

```
ggplot(data = aPixeldf) +
   geom_line(aes(x = Wavelength, y = ScaledReflectance)) +
   xlab("Wavelength (nm)") +
   ylab("Reflectance")
```

Select pixels and compare spectral signatures

```
plotRGB(rgb_harv_stack,
        r = 1, g = 2, b = 3,
        stretch = "lin")
```

Once you have clicked your five points, press the ESC key to save your clicked points and close the function before moving on to the next step. If you make a mistake in the step, run the plotRGB() function again to start over.

```
# change plotting parameters to better see the points and numbers generated from clicking
par(col = "red", cex = 3)

# use the 'click' function
clk <- click(rgb_harv_stack, id = T, xy = T, cell = T,
           type = "p", pch = 16, col = "magenta", col.lab = "red")
```

```
# convert raster cell number into row and column (used to extract spectral signature below)
clk$row <- clk$cell%/%nrow(rgb_harv_stack) + 1 # add 1 because R is 1-indexed
clk$col <- clk$cell%%ncol(rgb_harv_stack)
```

```
# create a new dataframe from the band wavelengths so that we can add
# the reflectance values for each cover type
```

```
Pixel_df <- as.data.frame(WL)

# loop through each of the cells that we selected
for(i in 1:length(clk$cell)){
# extract Spectra from a single pixel
aPixel <- h5read(f, "/HARV/Reflectance/Reflectance_Data",
                 index = list(NULL, clk$col[i], clk$row[i]))

# scale reflectance values from 0-1
aPixel <- aPixel/as.vector(scaleFact)

# reshape the data and turn into dataframe
b <- adply(aPixel, c(1))

# rename the column that we just created
names(b)[2] <- paste0("Point_", i)

# add reflectance values for this pixel to our combined data.frame called Pixel_df
Pixel_df <- cbind(Pixel_df, b[2])
}
```

```
# Use the melt() function to reshape the dataframe into a format that ggplot prefers
Pixel.melt <- melt(Pixel_df, id.vars = "WL", value.name = "Reflectance")

## Warning in melt(Pixel_df, id.vars = "wavelengths", value.name = "Reflectance"): The melt
## generic in data.table has been passed a data.frame and will attempt to redirect to the
## relevant reshape2 method; please note that reshape2 is deprecated, and this redirection is
## now deprecated as well. To continue using melt methods from reshape2 while both libraries
## are attached, e.g. melt.list, you can prepend the namespace like reshape2::melt(Pixel_df).
## In the next version, this warning will become an error.

# Now, let's plot some spectral signatures!
ggplot() +
  geom_line(data = Pixel.melt, mapping = aes(x = WL,
                                             y = Reflectance,
                                             color = variable), lwd = 1.5) +
  scale_colour_manual(values = c("green2", "green4", "chartreuse3", "tan4", "blue3"),
                      labels = c("Forest1", "Forest2", "Forest3", "Anthropogenic", "Water"))+
  labs(color = "Cover Type") +
  ggtitle("Land cover spectral signatures") +
  theme(plot.title = element_text(hjust = 0.5, size = 20)) +
  xlab("Wavelength")
```

```
# grab Reflectance metadata (which contains absorption band limits)
reflMetadata <- h5readAttributes(f, "/HARV/Reflectance" )

ab1 <- reflMetadata$Band_Window_1_Nanometers
ab2 <- reflMetadata$Band_Window_2_Nanometers

# Plot spectral signatures again with rectangles showing the absorption bands
ggplot() +
  geom_line(data = Pixel.melt, mapping = aes(x = WL,
                                             y = Reflectance,
                                             color = variable), lwd = 1.5) +
```

```
  geom_rect(mapping = aes(ymin = min(Pixel.melt$Reflectance),
                          ymax = max(Pixel.melt$Reflectance),
                          xmin = ab1[1], xmax = ab1[2]),
            color = "black", fill = "grey40", alpha = 0.8) +
  geom_rect(mapping = aes(ymin = min(Pixel.melt$Reflectance),
                          ymax = max(Pixel.melt$Reflectance),
                          xmin = ab2[1], xmax = ab2[2]),
            color = "black", fill = "grey40", alpha = 0.8) +
  scale_colour_manual(values = c("green2", "green4", "chartreuse3", "tan4", "blue3"),
                      labels = c("Forest1", "Forest2", "Forest3", "Anthropogenic", "Water")) +
  labs(color = "Cover Type") +
  ggtitle("Land cover spectral signatures") +
  theme(plot.title = element_text(hjust = 0.5, size = 20)) +
  xlab("Wavelength")
```

```
# Duplicate the spectral signatures into a new data.frame
Pixel.melt.masked <- Pixel.melt

# Mask out all values within each of the two atmospheric absorbtion bands
Pixel.melt.masked[Pixel.melt.masked$WL >
                    ab1[1] & Pixel.melt.masked$WL < ab1[2], ]$Reflectance <- NA

Pixel.melt.masked[Pixel.melt.masked$WL >
                    ab2[1] & Pixel.melt.masked$WL < ab2[2], ]$Reflectance <- NA

# Plot the masked spectral signatures
ggplot() +
  geom_line(data = Pixel.melt.masked, mapping = aes(x = WL,
                                                    y = Reflectance,
                                                    color = variable), lwd = 1.5) +
  scale_colour_manual(values = c("green2", "green4", "chartreuse3","tan4","blue3"),
                      labels = c("Forest1", "Forest2", "Forest3", "Anthropogenic", "Water"))+
  labs(color = "Cover Type")+
  ggtitle("Land cover spectral signatures")+
  theme(plot.title = element_text(hjust = 0.5, size=20))+
  xlab("Wavelength")
```

```
# It's always good practice to close the H5 connection before moving on!
# close the H5 file
H5close()
```

```
plot(ndvi_calc)
```

```
HARV_shannon <- Shannon(ndvi_calc, window = 5)
```

```
plot(HARV_shannon)
```

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.