Article

Philipp Baumann | philipp.baumann@usys.ethz.ch

2019-12-01

Intro

Albert Einstein would probably not have felt the necessity for simplerspec, as he would have followed his quote *"Everything should be made as simple as possible, but not simpler"*. In line with this recommendation, I was told that spectral analysis in R is standard practice and straight forward using the famous partial least squares (PLS) regression when I started my MSc back in July 2015. I had the chance to sample and model both soils and yam plants from 20 fields in 4 landscapes across the West African yam belt (see here for details). Since I was both fascinated by R, statistics, soils, and their interplay with plants, I started my first scientific journey with the premise that I just had to deepen a bit my R knowledge.

There is a plethora of chemometrics and other statistical learning toolboxes, and many of them are for example available via CRAN. Most of them are good at solving single tasks, but I somehow missed a clean common interface that interlinked the key steps required for spectral processing and modeling. While doing first analysis steps, my intuition told me that streamlining all analysis steps would aid in more efficiently estimating the composition and properties of natural materials. More importantly, it would allow a sustainable basis for model development and sharing with collaborators by simplifying repetitive boilerplate code. This was the motivation when I started continuously building simplerspec. The package aims to provide a rapid prototyping pipeline for various spectroscopy applications that share common tasks.

Hands-on

Enough of the talking, let's start. First, clone this repository to your local computer to reproduce the entire analysis in this hands-on. You can download a compressed archive manually, or use git to clone from this website:

```
git clone https://github.com/philipp-baumann/simplerspec-pedometron-article.git
```

For the installation of packages I would advise one of the two main procedures. Procedure one installs renv, which is then used restore simplerspec and remaining R packages versions as described `renv.lock` file in an isolated project library.

```
## Option 1 for installation
install.packages("renv"); renv::restore("renv.lock")
```

Procedure two below should also work, however comes without guarantee of identical package versions.

```
## Option 2 for installation
pkgs <- c("simplerspec", "here", "tidyverse", "data.table",
  "future", "doFuture", "remotes")
```

```
install.packages(pkgs)
remotes::install_github("philipp-baumann/simplerspec")
```

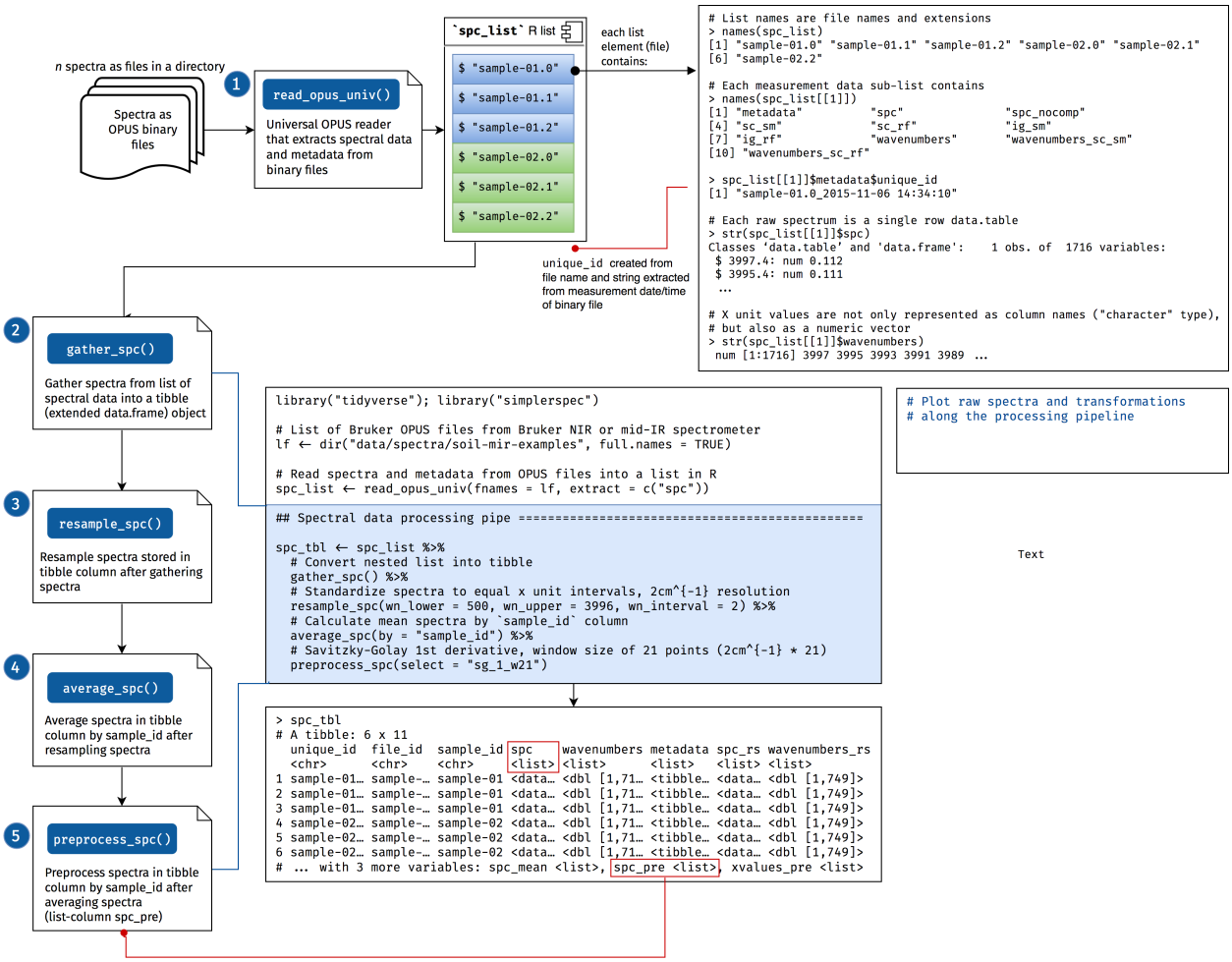
Now we are ready to dive into to the fundamentals of the package. We use the example data set from my MSc thesis. First, let's load required packages.

```
# Load required packages; `walk()` is like `lapply()`, but returns invisibly
suppressPackageStartupMessages(
  purrr::walk(pkgs, library, character.only = TRUE, quietly = TRUE)
)
```

A typical simple spectroscopy modeling project has the following components:

1. Soil sampling and sample preparation
2. Spectral measurements
3. Selection of calibration samples
4. Soil analytical reference analyses
5. a. Calibration or Recalibration
   b. Estimation of properties of new soils based on new spectra and established models.

Simplerspec focuses on the key tasks and provides user-friendly modules in the form of a standardized function pipeline. The spectral processing pipeline comprises basic steps that are often performed for spectral modeling and estimation, covering steps 2 to 5 of the above listed common spectroscopy components. Simplerspec uses prospectr for key steps and data.table for simple operations. The following scheme summarizes the spectral processing steps.



We assume that spectral measurements are done before chemical reference analyses as the former are faster and cheaper to do. Here we read the data from a Bruker Alpha mid-Infrared spectrometer. Currently, the package is limited to Bruker and ASD devices, however support for reading files from other devices and formats is planned within the package simplerspec.io.

```
# multicore futures are not supported when using RStudio (stability reasons)
plan(multisession)
registerDoFuture()
# availableCores()

# files to read
files_spc <- list.files(
  here("data", "spectra", "example-yamsys"), full.names = TRUE)
# read the files
suppressMessages(
  spc_list <- read_opus_univ(fnames = files_spc, extract = c("spc"),
    parallel = TRUE)
)
```

Typically, list information is nicely ordered, however printing is really verbose (see the spectral processing scheme for details). Therefore, we can gather the list into a so-called spectral tibble (`spc_tbl`; data.frame extension).

```
spc_tbl <- gather_spc(data = spc_list)
```

Instead of appending a matrix of spectra as a single column in a data.frame, spectra in a spectral tibble form a list-column. A list-column is basically a column consisting of a list instead of an atomic vector. With this we can extract this list column of spectra.

```
spc_dt <- data.table::rbindlist(spc_tbl$spc)
dim(spc_dt); class(spc_dt)
```

```
## [1]  284 1716
```

```
## [1] "data.table" "data.frame"
```

In a nutshell, spectral data processing can be done in one pipeline. Resampling in this context refers to to creating a new x axis interval in spectra. Spectra are averaged because there are 3 replicate measurements for each soil sample. Preprocessing is done to reduce scattering and noise in spectra.

```
spc_proc <-
  spc_tbl %>%
  resample_spc(wn_lower = 500, wn_upper = 3996, wn_interval = 2) %>%
  average_spc(by = "sample_id") %>%
  preprocess_spc(select = "sg_1_w21") %>%
  group_by(sample_id) %>%
  slice(1L) # remove replicate spectra (averaged)
```

After preprocessing, we can fuse the the final reference analysis data:

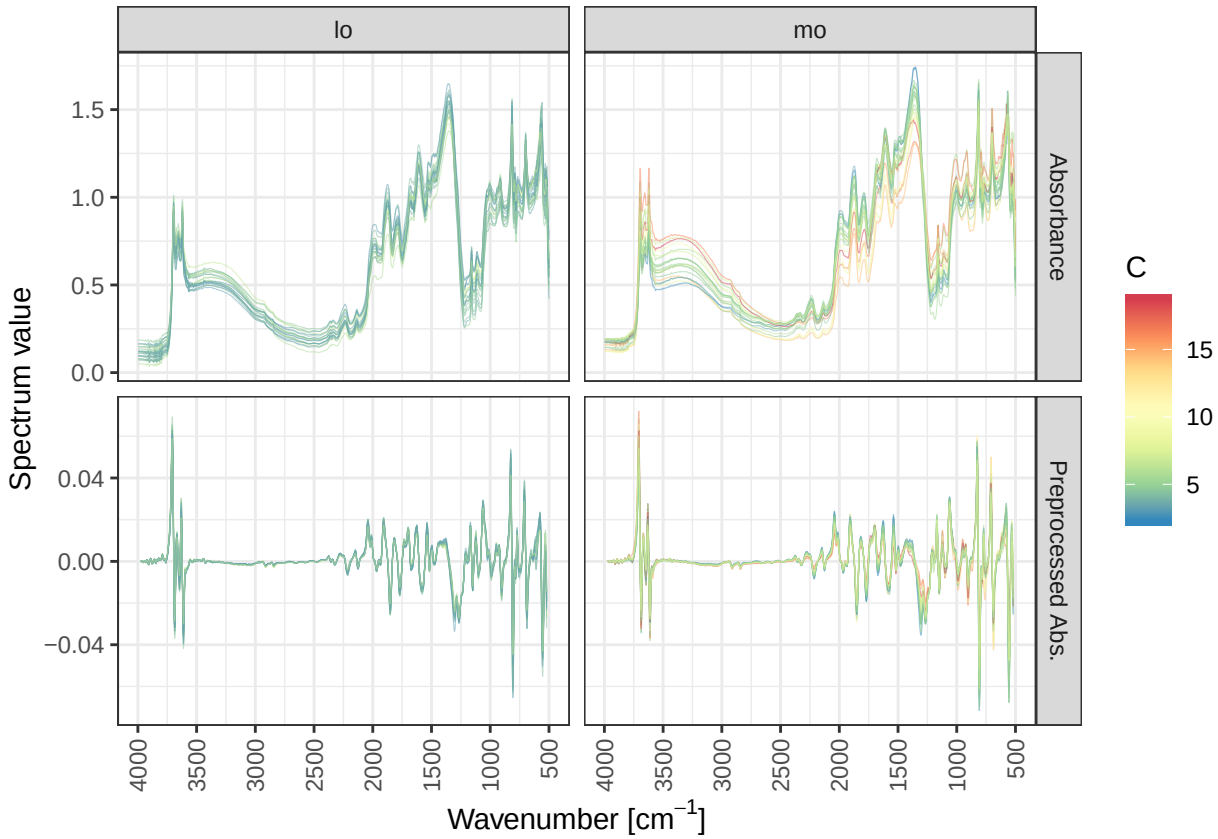
```
# see data/reference-data/metadata_soilchem_yamsys.txt for further details
reference_data <- fread(
  file = here("data", "reference-data", "soilchem_yamsys.csv")) %>%
  as_tibble()
# number of rows and columns
dim(reference_data)
```

```
## [1] 94 36
```

```
# Fuse spectra and reference data by `sample_id`
spc_refdata <-
  inner_join(
    x = spc_proc,
    y = reference_data %>% rename(sample_id = sample_ID),
    by = "sample_id"
  )
```

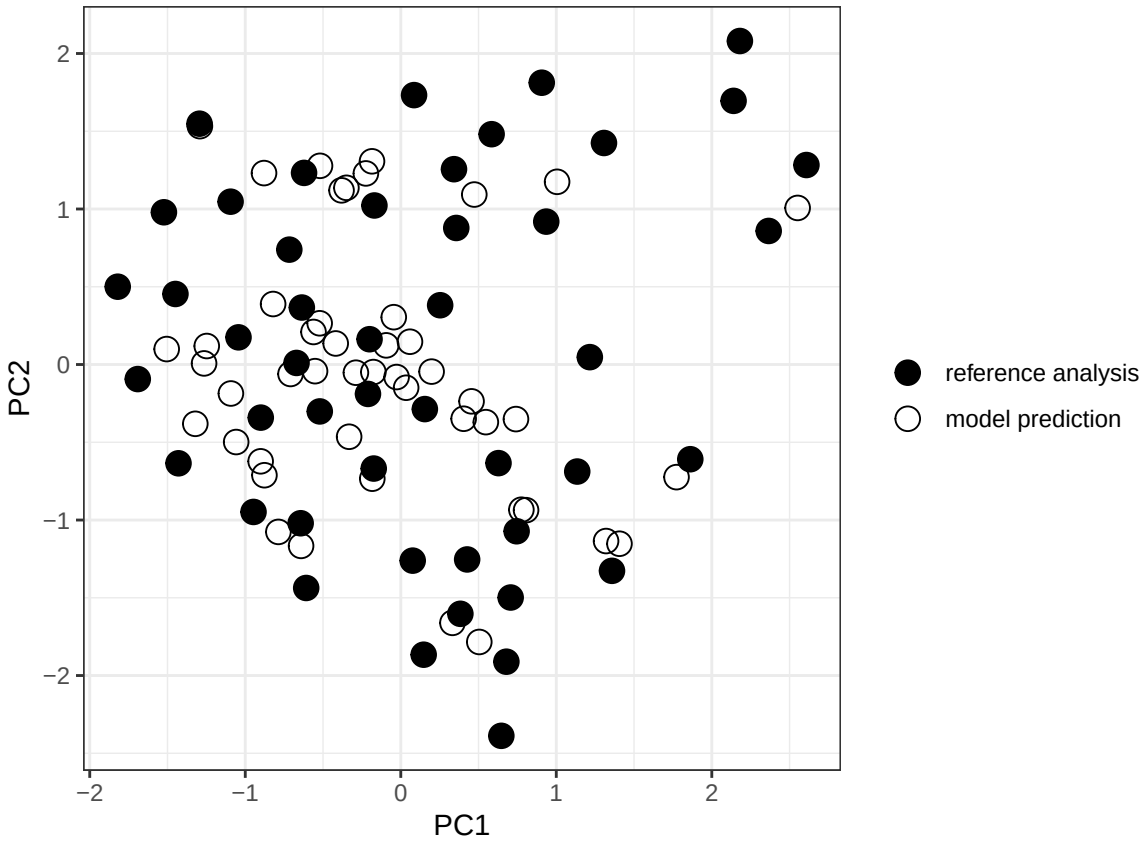
We can explore the final processed spectra.

```
spc_refdata %>%
  filter(site %in% c("lo", "mo")) %>%
  plot_spc_ext(
    spc_tbl = .,
    lcols_spc = c("spc", "spc_pre"),
    lcol_measure = "C",
    group_id = "site")
```



After preprocessing, we can proceed with selecting reference analytical samples based on Kennard-Stone.

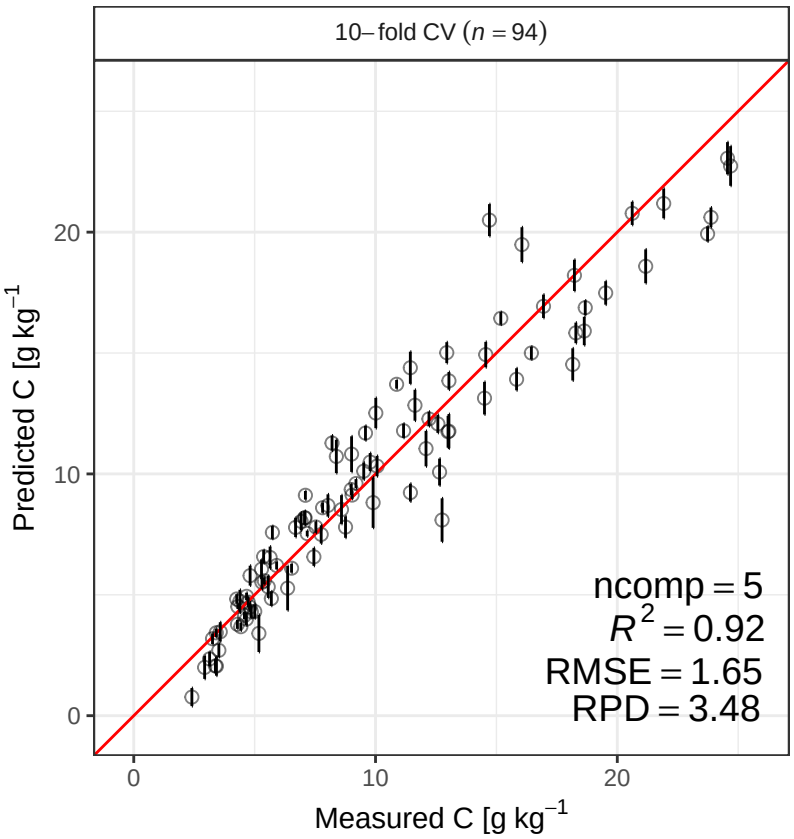
```
spc_tbl_selection <- select_ref_spc(spc_tbl = spc_proc, ratio_ref = 0.5)
# PCA biplot
spc_tbl_selection$p_pca
```



Lastly, we develop a partial least squares (PLS) calibration model.

```
pls_carbon <- fit_pls(spec_chem = spc_refdata, response = C,
  evaluation_method = "resampling", print = FALSE)
```

```
pls_carbon$p_model +
  xlab(expression(paste("Measured C [g", ~kg^-1))) +
  ylab(expression(paste("Predicted C [g", ~kg^-1)))
```



Outro

Simplerspec are some first baby steps in spectral adventures. The package deals with simplifying standard tasks and now has mainly exploration and teaching purposes. As an example, the Congo spectral platform uses some of its functionality. Complex problems and professional spectroscopy applications require transfer learning and spectral feature engineering pipelines that tune automatically. If you have ideas to collaborate and develop new frameworks, just send me an email or interact via github.