simplerspec: streamlining spectral data processing and modeling for spectroscopy applications

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

2019-12-17

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 my R knowledge a bit.

There is a plethora of chemometrics and other statistical learning toolboxes, and many of them are available via CRAN, for example. 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. Back then I thought streamlining all analysis tasks would produce a sustainable basis for model development and sharing with collaborators. In particular, simplifying repetitive boilerplate code was the motivation when I started building simplerspec step by step. The package aims to provide a rapid prototyping pipeline for various spectroscopy applications that share common tasks.

Hands-on

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:

```
# command line option
git clone https://github.com/philipp-baumann/simplerspec-pedometron-article.git
# or via RStudio: see https://happygitwithr.com/rstudio-git-github.html
```

For the installation of packages I would advise one of the two main procedures. Procedure one installs renv, which is then used to restore simplerspec and remaining versions of R packages 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")
```

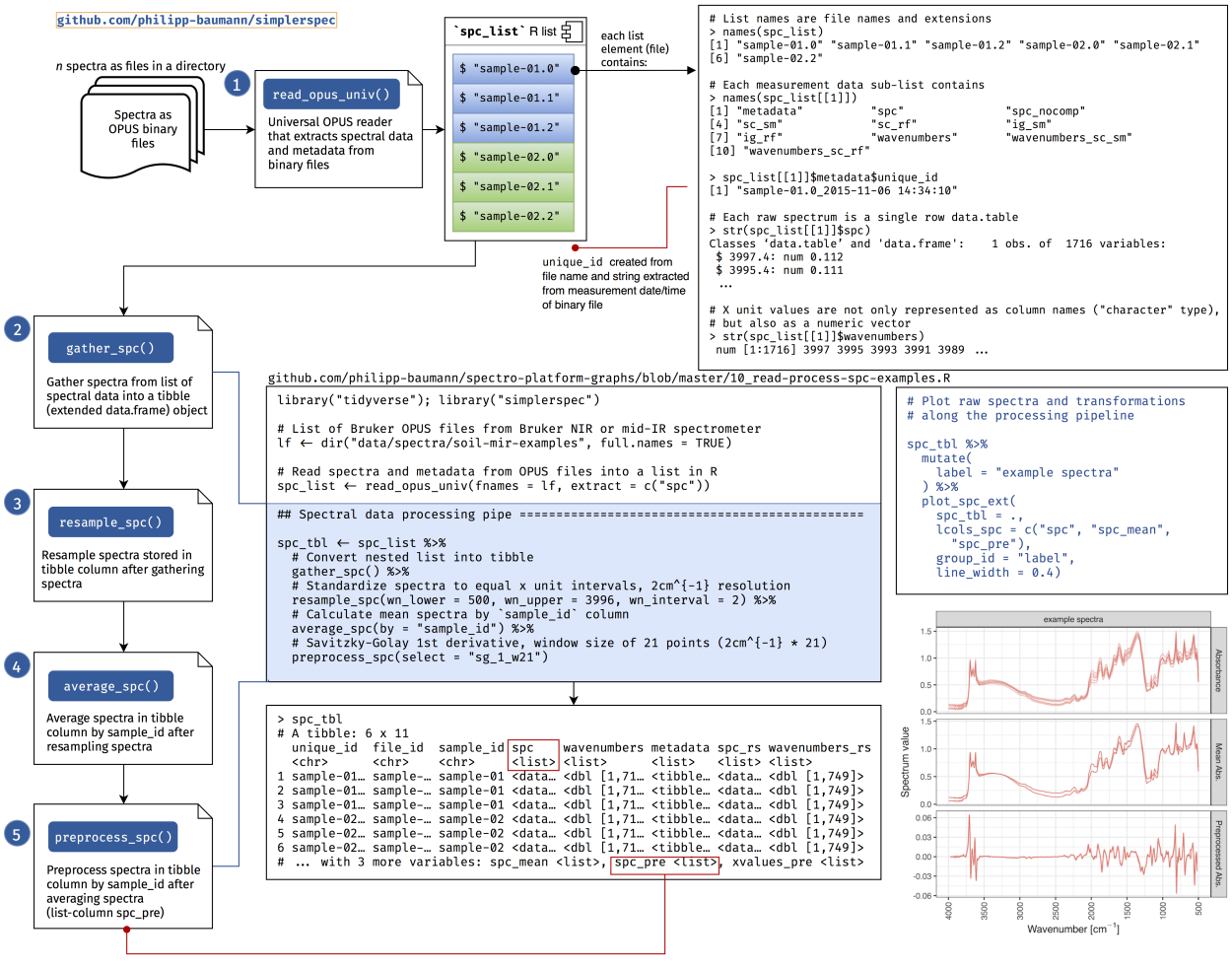
We use the example data set from my MSc thesis. First, let's load required packages.

```
# Package detection with "apply type" package load is not yet supported in renv
suppressPackageStartupMessages(
  xfun::pkg_attach("simplerspec", "here", "tidyverse", "data.table",
  "future", "doFuture", "remotes")
)
```

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 in spectral modeling and estimation (components 2 to 5 above), and provides user-friendly modules in the form of a standardized function pipeline. 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 are represented as a list of `data.table`s split by rows, also forming a column (list-column; see scheme above).

In a nutshell, spectral data processing can be done in one pipeline. Resampling in this context refers 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 the 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 read the final reference analysis data and merge it with the spectral tibble:

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

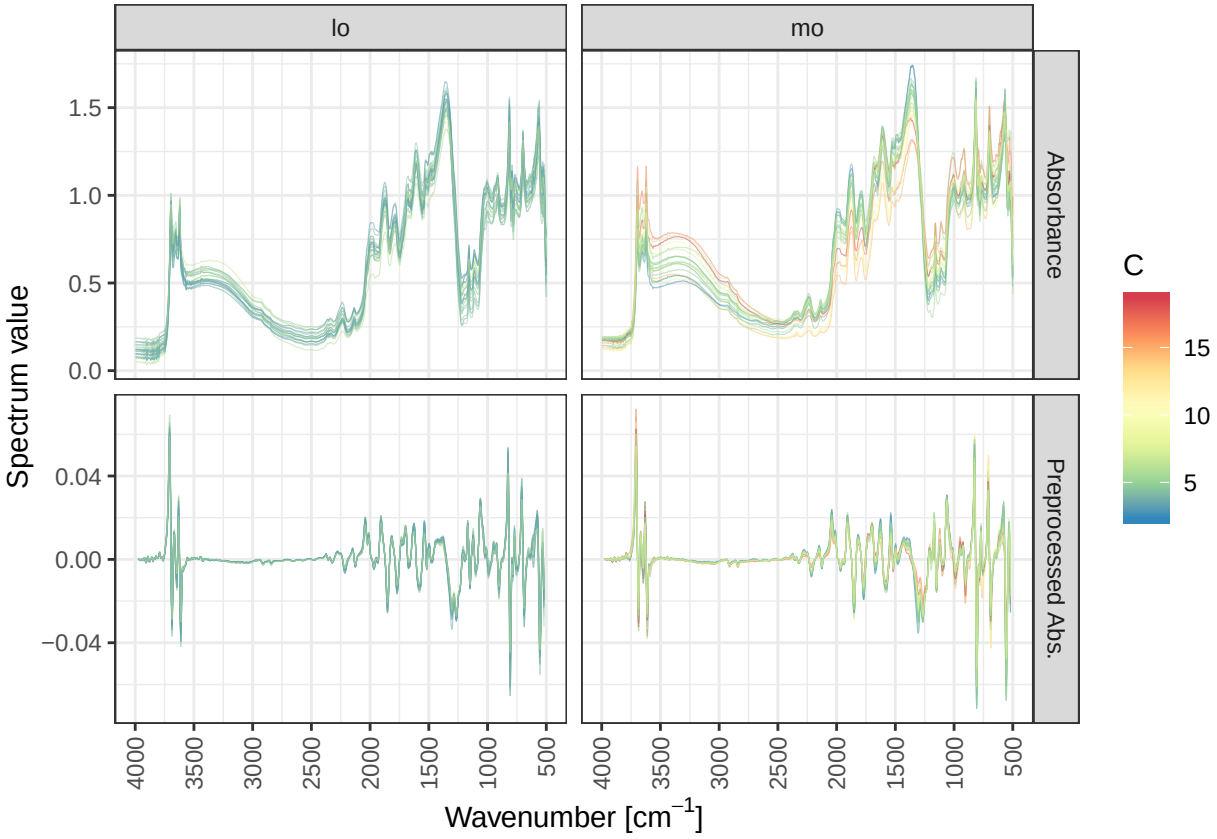
```
## [1] 94 36
```

```
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.

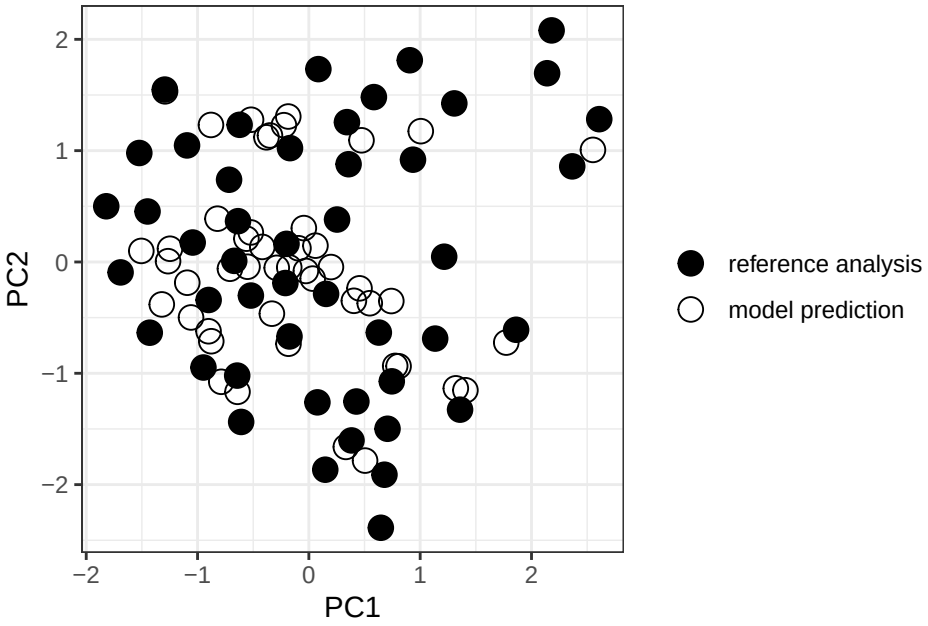
```
spc_refdata %>%
  filter(site %in% c("lo", "mo")) %>% # two landscapes in Burkina Faso
```

```
  plot_spc_ext(
    spc_tbl = .,
    lcols_spc = c("spc", "spc_pre"),
    lcol_measure = "C",
    group_id = "site")
```



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

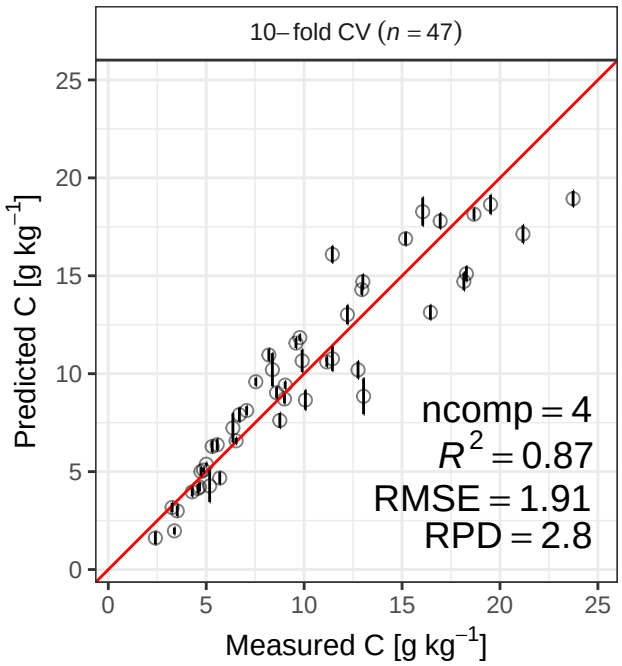
```
spc_tbl_selection <- select_ref_spc(spc_tbl = spc_proc, ratio_ref = 0.5)
spc_ref <- spc_tbl_selection$spc_ref
spc_pred <- spc_tbl_selection$spc_pred
# PCA biplot
spc_tbl_selection$p_pca
```



Lastly, we develop a partial least squares (PLS) calibration model for total Carbon (C), hypothetically assuming that we only have above selected 50% calibration data.

```
pls_carbon <- fit_pls(
  spec_chem = spc_refdata %>% filter(sample_id %in% spc_ref$sample_id),
  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, "]")))
```



What remains is the estimation of total C for the model prediction samples (component 5.ii) based on the model trained above (component 5.i) and the assessment thereof.

```
spc_ref_pred <- predict_from_spc(model_list = list("pls_carbon" = pls_carbon),
  spc_tbl = spc_pred %>% filter(sample_id %in% spc_pred$sample_id))
# Assess estimation of total C on prediction samples
assess_multimodels(
  data = spc_ref_pred %>% inner_join(spc_refdata %>% select(sample_id, C)),
  C = vars(o = "C", p = "pls_carbon"), .metrics = c("simplerspec"))
```

```
## # A tibble: 1 x 27
##   model     n   min   max  mean median  sdev    cv skewness_b1 kurtosis
##   <chr> <int> <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl>       <dbl>    <dbl>
## 1 C        47     1  24.7  9.86   7.75  6.18 0.626       0.990  -0.0845
## # ... with 17 more variables: rmse <dbl>, mse <dbl>, me <dbl>, bias <dbl>,
## #   msv <dbl>, sde <dbl>, mae <dbl>, r2 <dbl>, b <dbl>, rpd <dbl>,
## #   rpiq <dbl>, SB <dbl>, NU <dbl>, LC <dbl>, SB_prop <dbl>,
## #   NU_prop <dbl>, LC_prop <dbl>
```

Outro

The steps shown here using simplerspec are merely some first baby steps in a realm of possible spectral adventures. The package deals with simplifying standard tasks and currently mainly focuses on 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 (i.e., transferring knowledge from big spectral libraries into to new target set of soils) and spectral feature engineering (i.e., modifying spectra with operations that enable models to better discover predictor-response relationships) pipelines that tune automatically. If you have ideas to collaborate and develop new frameworks, just send me an email or interact via github.