R code for Robust soil mapping at the farm scale with vis-NIR spectroscopy (Ramirez-Lopez *et al.*, 2019)

Leo Ramirez-Lopez (BUCHI Labortechnik AG, Switzerland)
Alex Wadoux (Wageningen University, Netherlands)

2019-06-18

Contents

Preface **5**

1 Introduction **7**
- 1.1 Paper summary . . . 7
- 1.2 Study area . . . 7
- 1.3 In case of comments/questions/issues . . . 7
- 1.4 For citation or details please refer to: . . . 7
- 1.5 Notes . . . 8

2 Required `R` packages **9**

3 Required data **11**

4 Sampling **13**
- 4.1 Optimal calibration set size identification . . . 13
- 4.2 Plotting the results . . . 16
- 4.3 Final selection of the calibration set . . . 17

5 Splitting the data **19**

6 Transformation of the particle-size data **21**

7 Vis–NIR modelling and predictions **23**
- 7.1 Calibration and validation . . . 23
- 7.2 Property predictions in the prediction set . . . 26

8 Prepare the vis-NIR augmented dataset **29**

9 Spatial modeling **31**
- 9.1 Robust fitting of the spatial models . . . 31
- 9.2 Accounting for vis-NIR model errors in the spatial models . . . 37
- 9.3 Validation of the spatial models . . . 37
- 9.4 Mapping . . . 41

Preface

In the spirit of reproducible research, here we share the data and the computational code used for carrying out the analyses presented in our paper “Robust soil mapping at the farm scale with vis-NIR spectroscopy” which is by the way open access. Before you go thorugh this code we strongly recoment to read our paper.

Chapter 1

Introduction

Here you will find some basic informatzion about the paper and the `R` code opresented here.

1.1 Paper summary

Sustainable agriculture practices are often hampered by the prohibitive costs associated with the generation of fine-resolution soil maps. Recently, several papers have been published highlighting how visible and near infrared(vis-NIR) reflectance spectroscopy may offer an alternative to address this problem by increasing the density of soil sampling and by reducing the number of conventional laboratory analyses needed. However, for farm-scale soil mapping, previous studies rarely focused on sample optimization for the calibration of vis-NIR models or on robust modelling of the spatial variation of soil properties predicted by vis-NIR spectroscopy. In the present study, we used soil vis-NIR spectroscopy models optimized in terms of both number of calibration samples and accuracy for high-resolution robust farm-scale soil mapping and addressed some of the most common pitfalls identified in previous research. We collected 910 samples from 458 locations at two depths (A, 0-0.20 m; B, 0.80-1.0 m) in the state of Sao Paulo, Brazil. All soil samples were analysed by conventional methods and scanned in the vis-NIR spectral range. With the vis-NIR spectra only, we inferred statistically the optimal set size and the best samples with which to calibrate vis-NIR models. The calibrated vis-NIR models were validated and used to predict soil properties for the rest of the samples. The prediction error of the spectroscopic model was propagated through the spatial analysis, in which robust block kriging was used to predict particle-size fractions and exchangeable calcium content for each depth. The results indicated that statistical selection of the calibration samples based on vis-NIR spectra considerably decreased the need for conventional chemical analysis for a given level of mapping accuracy.

1.2 Study area

The study area is located in Brazil between the municipalities of Barra Bonita and Mineiros do Tiete (in the state of Sao Paulo).

1.3 In case of comments/questions/issues

In case you have any question comments/questions/issue related to the code presented here please go to: https://github.com/l-ramirez-lopez/VNIR_spectroscopy_for_robust_soil_mapping/issues and create a new issue.

1.4 For citation or details please refer to:

Ramirez-Lopez, L., Wadoux, A. C., Franceschini, M. H. D., Terra, F. S., Marques, K. P. P., Sayao, V. M., & Dematte, J. A. M. (2019). Robust soil mapping at the farm scale with vis-NIR spectroscopy. European Journal of Soil Science.

1.5 Notes

- *Reproducibility*: slight discrepancies between the results of reoported in the paper and the results you get in your `R` console might be expected in case you use different `R` package versions and different random number generators.
- *To advanced R users*: In order to make the `R` code more readable/interpretable we have opted for using `for` loops instead of vectorized functions. There are several aspects of the code that can be improved to reach better computational efficiency.

Chapter 2

Required R packages

In order to run all the code presented in the next sections you will need to have the following packages installed

You can install the required packages using the following code

```r
requiredpackages <- c("resemble", "prospectr", "clhs", "matrixStats", "doParallel",
                      "ggplot2", "tidyr", "tidyverse", "georob", "rgdal", "raster", "RColorBrewer")
toinstall <- requiredpackages[!requiredpackages %in% rownames(installed.packages())]
if (length(toinstall) > 0) {
                      install.packages(toinstall)
}
lapply(requiredpackages, FUN = library, character.only = TRUE)
```

Chapter 3

Required data

In this section you will find the instructions on how to download and read the soil dataset used in our paper.

You can download the file 'SoilNIRSaoPaulo.rds', alternatively you can visit to the GitHub repository of the paper where the file resides.

This `rds` file contains a `data.frame` with 910 rows (samples) and the following variables:

- Nr: An arbitrary sample number.
- ID: A `factor` indicating the sample IDs. The first character is a letter which indicates the depth layer at which the sample was collected (A: 0-20 cm and B: 80-100 cm).

- POINT_X: The X (geographical) coordinate.
- POINT_Y: The Y (geographical) coordinate.
- Sand: The percentage of sand contnet in the soil sample.
- Silt: The percentage of silt contnet in the soil sample.
- Clay: The percentage of clay contnet in the soil sample.
- Ca: The exchangeable Calcium content in the sample ($mmol_c\ kg^1$).
- set: A `factor` indicating whether the sample is used for model's validation or if it can be used to calibrate models.
- spc: A set of 307 variables representing the wavelengths from 502 nm to 2338 nm in steps of 6 nm.

Further information on this dataset can be found in our paper

We recommend to create a local folder (e.g. "./myworkingdirectory"). If you downloaded the file to this local folder then:

```
workingd <- "/myworkingdirectory"
setwd(workingd)
data <- readRDS("SoilNIRSaoPaulo.rds")
```

Alternatively, you can also read the file directly from the GitHub repository of the paper:

```
nirfile <- file("https://github.com/l-ramirez-lopez/VNIR_spectroscopy_for_robust_soil_mapping/raw/master
data <- readRDS(nirfile)
names(data)
```

In addition you also need a `R` object of class (`SpatialPolygonsDataFrame` of the package `sp`) which contains the polygon of the study area. You can also download the file (polygon.rds) containing this object by clicking here.

Chapter 4

Sampling

4.1 Optimal calibration set size identification

In this section we show how the optimal size for a calibration set is identified using the mean squared Euclidean distance (msd) between estimates of the probability density functions ($pdfs$) of the whole set of samples and the pdfs of subsets with different sizes.
First we compute the principal components (PCs) of the NIR spectra of the whole set of calibration candidate samples. Then for each component we compute kernel density estimates (KDE):

```
## Apply standard normal variate
data$spc_snv <- standardNormalVariate(data$spc)

## extract the validation samples into a new set/object
valida <- data[data$set == "validation",]

## remove the validation samples from data
data <- data[data$set == "cal_candidate",]

## --- 2. Perform a principal component analysis  ----
## Compress the data
pcaall <- orthoProjection(Xr = data$spc_snv,
                          X2 = NULL,
                          Yr = NULL,
                          method = "pca",
                          pcSelection = list("cumvar", 0.99),
                          center = TRUE,
                          scaled = FALSE)

## standardize the socres
pcaall$scores.std <- sweep(pcaall$scores, MARGIN = 2, STATS = pcaall$sc.sdv, FUN = "/")

## compute the max and min of each score for the limits of the density estimations
max.sc <- colMins(pcaall$scores.std)
min.sc <- colMaxs(pcaall$scores.std)

## compute the mean and sd of each score for the comparisons with the samples
mean.sc <- colMeans(pcaall$scores.std)
sd.sc <- colSds(pcaall$scores.std)

# number of points in the density distribution
nxdens <- 500

## matrix where the density values will be stored
ix <- 1
```

```r
sc.dens <- data.frame(seq(min.sc[ix], max.sc[ix], length = nxdens),
                      matrix(NA, nxdens, length(min.sc)))
colnames(sc.dens) <- c("x", paste("densc", 1:length(min.sc), sep = ""))

## Kernel density estimates (KDE) of each component
d.bandwidths <- rep(NA, length(min.sc))
names(d.bandwidths) <- colnames(pcaall$scores.std)
for(i in 1:length(min.sc)){
  idsty <- density(pcaall$scores.std[,i],
                   bw = "nrd0",
                   n = nxdens, from = min.sc[i], to = max.sc[i],
                   kernel = "gaussian")
  sc.dens[,i+1] <- idsty$y
  d.bandwidths[i] <- idsty$bw
}

## Create the vector of different sample set sizes to be tested
css <- seq(10, 400, by = 10)
```

Now we will iterate over the different calibration set sizes (`css`). We are going to use the LHS algorithm to select subsets from our set of candiate samples for calibartion. This is done using the (standardized) scores of the principal components (PCs) of the spectra (`pcaall$scores.std`). For each calibration subset we'll compute the mean squared Euclidean distance (*msd*) between estimates of the probability density functions (*pdfs*) of the whole set of samples and the *pdfs* of samples in the subset. The *msd* is computed using kernel density estimates (*KDE*) of the *pdfs*. To obtain reliable estimates of *msd* as a function of the sample set size, we will repeat 10 times (`repetitions`) the whole sampling procedure for each `css`, the final *msd* will be the the average of the *msd*s obtained at each iteration. The following pseudo-code summarizes the procedure to compute the *msd* (see the '*Calibration samples*' section in our paper for more details):

```
Input:  PCs of the whole set of candidate samples (p);
        KDEs of the PCs of the whole set of candidate samples;
Output: msd

1 for each repetition do:
2   for each proposed sampling size do:
3     select from the PCs a subset (s);
4     for each component in s and p:
5       compute the msd between KDE(s,component) and KDE(p,component);
6     end
7   end
8 end

9 aggregate the results of the msds obatined for all
the repetition iterations for each sample set size
```

First we'll compute the *msd*s for ecah repetition and we will write each set of results into our working directory.

```r
## Sample with LHS (latin hypercube sampling)

## These three nested loops might take a while
## (aprox. 16 min per repetition, i.e. a bit more than a couple of hours
## for the whole thing)
## (for reducing the computation time the loops
## can be vectorized using the apply family of functions.
## Furtheromre parallelization of the loop for the repetitions
## can also be applied)
## We present the computations with nested loops for interpretability
## reasons

repetitions <- 10

## root name for the results of each iteration
```

```r
filerootname <- "6pcs_resultsclhs_rep"

## Create the data.frame where the results will be stored
## at each iteration
results.clhs <-  data.frame(css = css,
                            msd = rep(NA, length(css)),
                            mndiff = rep(NA, length(css)),
                            sddiff = rep(NA, length(css)))
for(k in 1:repetitions){
  results.clhs[,-1] <- NA

  ## Define a file name
  fn <- paste(filerootname, k,".txt", sep = "")
  if(fn %in% list.files()){
    results.clhs <- read.table(fn, header = T, sep = "\t")
  }

  iter.p <- 1 + sum(rowSums(!is.na(results.clhs)) == ncol(results.clhs))

  for(i in iter.p:length(css)){
    set.seed(k)
    i.calidx <- clhs(x = as.data.frame(pcaall$scores.std),
                     size = css[i],
                     iter = 10000)

    ## Compute the KDEs of each PC
    j.sc.dens <- msd.sc  <- sc.dens
    for(j in 1:length(min.sc)){
      ## use the same bandwidth (bw) as in the whole set of candidates
      j.sc.dens[,j + 1] <- density(x = pcaall$scores.std[i.calidx,j],
                                   bw = d.bandwidths[j],
                                   n = nxdens,
                                   from = min.sc[j],
                                   to = max.sc[j],
                                   kernel = "gaussian")$y
    }
    results.clhs$msd[i] <- mean(colMeans((j.sc.dens[,-1] - sc.dens[,-1])^2, na.rm = T))
    results.clhs$mndiff[i] <- mean(abs(colMeans(pcaall$scores.std[i.calidx,])))
    results.clhs$sddiff[i] <- mean(abs(colSds(pcaall$scores.std[i.calidx,]) - 1))

    ## write the results obtained so far...
    if(iter == length(nmsrepsclhs)){
      final.clhs <- final.clhs/iter
      write.table(x = final.clhs,
                  file = "6pcs_final.clhs.txt",
                  sep = "\t",
                  row.names = FALSE)
    }

    print(results.clhs[1:i,])
  }
}
```

After executing the above nested loops, you should have obtained 10 different *.txt files in your working directorty (in this case the root name of the files is 6pcs_resultsclhs_rep[n].txt where [n] represents the repetition number). These files contain the *msd* results obtained at each repetition iteration. Now we'll calculate the final *msd* as the average of the ones obtained at each repetition iteration.

```r
## Specify a file name for the file where the final results will be stored
finalresultsfile <- "6pcs_final.clhs.txt"
```

```
nmsrepsclhs <- paste(filerootname, 1:repetitions, ".txt", sep = "")
final.clhs <- 0
for(i in nmsrepsclhs){
  iter <- which(i == nmsrepsclhs)
  results.clhs <- read.table(i, header = T, sep = "\t")
  results.clhs$mndiff <- abs(results.clhs$mndiff)
  final.clhs <- final.clhs + results.clhs

  ## write a table with the final results
  if(iter == length(nmsrepsclhs)){
    final.clhs <- final.clhs/iter
    write.table(x = final.clhs,
                file = finalresultsfile,
                sep = "\t",
                row.names = FALSE)
  }
}

## Compute the standard devitions of the msd results
final.clhs_sd <- 0
for(i in nmsrepsclhs){
  iter <- which(i == nmsrepsclhs)
  results.clhs <- read.table(i, header = T, sep = "\t")
  results.clhs$mndiff <- abs(results.clhs$mndiff)
  final.clhs_sd <- (results.clhs - final.clhs_sd)^2
  if(iter == length(nmsrepsclhs)){
    final.clhs_sd <- (final.clhs_sd/iter)^0.5
  }
}
```

After executing the above code, a file with the final *msd* estimations is writen in your working directory. In addition the standard deviations of the *msd* for the different set sizes was also computed and stored in the object `final.clhs_sd`.

4.2 Plotting the results

To plot the *msd* results using `ggplot` (Figure 3 in our paper):

```
final.clhs_plot <- data.frame(final.clhs[,1:2],
                              sd_lower = final.clhs[,2] - final.clhs_sd[,2],
                              sd_upper = final.clhs[,2] + final.clhs_sd[,2])

p.tmp <- ggplot(final.clhs_plot) + geom_line(aes(x = css, msd, colour = "msd")) +
  theme_bw() +
  theme(axis.title.y = element_text(face= "bold.italic", colour = grey(0.2), size=18),
        axis.text.y  = element_text(angle=0, vjust =0.5, hjust =0.5, size=14),
        legend.title = element_text(colour = "white", size=20)) +
  theme(axis.title.x = element_text(face= "bold", colour = grey(0.2), size=18),
        axis.text.x  = element_text(angle = 0, vjust=0, size=14)) +
  theme(legend.position = "none") +
  theme(legend.text = element_text(face="bold", colour = grey(0.2), size=18)) +
  labs(y = "msd", x = "Calibration set size") +
  #coord_cartesian(ylim = c(0, 8)) +
  theme(strip.background = element_rect(fill = "grey"),
        strip.text.x = element_text(size = 16, colour = "black", angle = 0))

p.tmp +
  geom_ribbon(aes(ymin=sd_lower, ymax=sd_upper, x=css, colour = "bands"), alpha = 0.2) +
  scale_colour_manual(name = '', values = c("bands" = NA, "msd" = "black"))
```

4.3 Final selection of the calibration set

The previous analysis allows to infer an optimal calibration set size. This optimal size is indicated by the point at which no substantial reduction of the *msd* is observed. In our case this assestment was intuetively done by looking at the *msd* vs. *css* plot. We set the optimal calibration set size (`ocss`) 180 samples:

```
# Optimal calibration sample set size
ocss <- 180
```

Once the optimal calibration set size is identified, we can propose/sample different calibration sets (solutions) containing 180 samples. In our paper we proposed 10 different solutions and we selected the one that returned the mimimum *msd* (this subset will be then used as the final calibration set in later analyses):

```
## Compute the maximum and minimum of each score for the limits of the
## density estimations
max.sc <- colMins(pcaall$scores.std)
min.sc <- colMaxs(pcaall$scores.std)

## Compute the mean and standard deviation of each score for the comparisons
## with the samples
mean.sc <- colMeans(pcaall$scores.std)
sd.sc <- colSds(pcaall$scores.std)

## Set a number of solutions to test
solutions <- 10

## object where the sample indices of the different solutions will be stored
calidx <- NULL

## object where the msds will be stored
results.clhs.ocss <- rep(NA, length = solutions)

for (i in 1:solutions) {
                    ## set seed for the random number generation
                    set.seed(round(i * exp(4)))

                    ## sample the ith solution
                    i.calidx <- clhs(x = as.data.frame(pcaall$scores.std), size = ocss, iter = 10000)

                    ## store the indices of the selected samples
                    calidx[[i]] <- i.calidx

                    ## Compute the KDEs
                    j.sc.dens <- msd.sc <- sc.dens
                    for (j in 1:length(min.sc)) {
                                        j.sc.dens[, j + 1] <- density(x = pcaall$scores.std[i.calidx, j]
                                                            n = nxdens, from = min.sc[j], to = max.sc[j]
                    }
                    ## compute the msds
                    results.clhs.ocss[i] <- mean(colMeans((j.sc.dens[, -1] - sc.dens[, -1])^2,
                                        na.rm = T))
}

## Identify the index of the best group
plot(results.clhs.ocss)

## best solution
which.min(results.clhs.ocss)

## get the indices of the samples in the final solution
calibration.idx <- calidx[[which.min(results.clhs.ocss)]]

## get the IDs of the samples in the final solution
```

```
cal_smpls <- as.character(data$ID[calibration.idx])
```

Save the IDs of the selected calibration samples into your working directory

```
writeLines(text = cal_smpls, con = "calibration_samples_ids.txt", sep = "\n")
```

Chapter 5

Splitting the data

At this point we can split the data into calibration, validation, and predition sets:

- The calibration set comprises the samples identified in the previous section. The IDs of these samples are in the object `cal_smpls`.
- The validation set are the ones in the original set and that are labeled as `validation`. In the previous section the validation samples where extracted into a separate object (`valida`).
- The prediction set includes all the samples not selected for calibration and that were initially labeled as `cal_candidate`.

To split the data you can execute the following:

```
train <- data[as.character(data$ID) %in% cal_smpls, ]
pred <- data[!(as.character(data$ID) %in% c(cal_smpls)), ]

train$layer <- as.factor(substr(train$ID, 1, 1))
valida$layer <- as.factor(substr(valida$ID, 1, 1))
pred$layer <- as.factor(substr(pred$ID, 1, 1))

train$ID <- factor(train$ID)
valida$ID <- factor(valida$ID)
pred$ID <- factor(pred$ID)
```

Optionally, we can get rid of all the unncessary data (`R` objects that will not be used from now on):

```
## necessary objects
reqobjects <- c("train", "pred", "valida", "cal_smpls", "o2rm")

## objects to be removed
o2rm <- ls()[!ls() %in% reqobjects]

## remove the objects
rm(list = o2rm)
```

Alternatively…

```
## If you have saved the IDs of the calibration samples into your working
## directory you can:
cal_smpls <- readLines("calibration_samples_ids.txt")
```

and then…

```
## necessary objects
reqobjects <- c("cal_smpls", "o2rm")

## objects to be removed
o2rm <- ls()[!ls() %in% reqobjects]
```

```
## read again the data
nirfile <- file("https://github.com/l-ramirez-lopez/VNIR_spectroscopy_for_robust_soil_mapping/raw/master
data <- readRDS(nirfile)

## extract the validation samples into a new set/object
valida <- data[data$set == "validation", ]
data <- data[data$set == "cal_candidate", ]

train <- data[as.character(data$ID) %in% cal_smpls, ]
pred <- data[!(as.character(data$ID) %in% c(cal_smpls)), ]

train$layer <- as.factor(substr(train$ID, 1, 1))
valida$layer <- as.factor(substr(valida$ID, 1, 1))
pred$layer <- as.factor(substr(pred$ID, 1, 1))

train$ID <- factor(train$ID)
valida$ID <- factor(valida$ID)
pred$ID <- factor(pred$ID)
```

Chapter 6

Transformation of the particle-size data

Sand, silt and clay contents are reported as proportions that sum to 100%. However, models formulated for each of these fractions do not guarantee that their individual predictions sum to 100%. To avoid this compositional constraint, the particle-size data ($V = clay, silt, sand$) for both depths ($l = A : 00.2m, B : 0.81.0m$) were transformed using the additive log-ratio (alr) transformation (Odeh et al., 2003):

$$Y_{l,i} = \frac{V_{l,i}}{V_{l,r}} \quad \forall \quad i = 1, 2, ..(r-1) \quad \forall \quad l \in (A, B)$$

where $Y_{l,i}$ is the resulting transformed variable, $V_{l,i}$ is the ith variable of the set of compositional variables (silt and clay contents) at depth l, $V_{l,r}$ designates a reference compositional variable at depth l and r is the total number of compositional variables. In ou paper, we used the sand content as reference ($V_{l,r}$).

```r
## The above equation can be simply applied in two lines of code

## Calibration dataset
## alr for silt contnet
train$alr_Silt <- log(train$Silt/train$Sand)
## alr for clay contnet
train$alr_Clay <- log(train$Clay/train$Sand)

## Validation dataset
## alr for silt contnet
valida$alr_Silt <- log(valida$Silt/valida$Sand)
## alr for clay contnet
valida$alr_Clay <- log(valida$Clay/valida$Sand)

## Prediction dataset
## alr for silt contnet
pred$alr_Silt <- log(pred$Silt/pred$Sand)
## alr for clay contnet
pred$alr_Clay <- log(pred$Clay/pred$Sand)
```

Chapter 7

Vis–NIR modelling and predictions

A prediction model based on vis–NIR spectra was fitted for each soil property using the selected optimal calibration subset. Each model was developed using a memory-based learning (MBL) algorithm (from the `resemble` package). Please check the paper for additional details on the MBL lagorithm used (see section *Vis–NIR modelling and predictions* subsections 1-3).

7.1 Calibration and validation

First define some basic aspects of the MBL which will be commmon to all the properties for which NIR modeling is going to be performed.

```r
## Add to the train data a variable indicating to what sampling point each
## sample belongs to. This variable will be used internally by the mbl
## function during the internal validations.
train$samplegroup <- substr(train$ID, 2, 100)

## Dissimilarity metric: partial least squares (pls)
dmetric <- "pls"

## Threshold distances to be tested Note: the number of threshold distances
## tested here is large, it could be reduced
diss2test <- seq(0.3, 1.5, by = 0.05)

## Define the minimum and maximum number of neighbors that must be retained
## for each local model.  Example: If with a given threshold distance only 70
## neighbors are selected, then the function will be fored to take the
## especified minimum number of neighbors to be included in the model. In
## this case 120.  We also set that the maximum number of neighbors that can
## be included can be all the samples in the calibration set.
kminmax <- c(120, nrow(train$spc))

## Regression method: Weighted average pls (wapls)
rmethod <- "wapls1"

## set the maximum and minimum number of pls factors for the local wapls
## regressions
pls.f <- c(minpls = 5, maxpls = 20)

## Adjust some additional parameters that control some aspects of the mbl
ctrl <- mblControl(sm = dmetric, pcSelection = list("opc", 40), valMethod = "NNv",
                   returnDiss = TRUE, scaled = FALSE, center = TRUE)
```

We also create some `data.frames` were we are going to store the prediction results for the validation set

```
## Create a data frame and store the validation results Squared correlation
## coefficient (R2) Root mean squared error (RMSE) Mean error (ME) (particle
## size predictions are also back-transformed to the original space to report
## the R2, RMSE and ME).
valrestuls <- data.frame(Property = c("Ca", "alr_clay", "alr_silt", "Clay",
                    "Silt", "Sand"), RMSE = NA, R2 = NA, ME = NA)

## Create a data frame and store the variance of the residuals for each layer
## (this will be later used for geostatistical analyses)
residualvariances <- data.frame(Property = c("Ca", "alr_Clay", "alr_Silt"),
                    layerA = NA, layerB = NA, layersAB = NA)
```

Now we perfom MBL and predictions for soil exchangeable Ca^{2+}:

```
## Run the MBL and find the optimal threshold distance for neighbor selection
## and predict Ca for validation - MBL fits a wapls model to each sample in
## Xu using the calibration samples {Yr,Xr} - The internal validation
## predictions these are predictions of the nearest samples of each Xu found
## in Xr - Nearest neighbor validation (NNv) -
sbl_ca <- mbl(Yr = train$Ca, Xr = train$spc, Xu = valida$spc, mblCtrl = ctrl,
                    group = train$samplegroup, dissUsage = "none", k.diss = diss2test, k.range = kminmax
                    pls.c = pls.f, method = rmethod)
sbl_ca

## Best threshold distance
idx.best.ca <- which.min(sbl_ca$nnValStats$st.rmse)
best.kdiss.ca <- sbl_ca$nnValStats$k.diss[idx.best.ca]

## Get the predicted values for the validation set
pred_valCa <- getPredictions(sbl_ca)[, idx.best.ca]

## R2
valrestuls$R2[valrestuls$Property == "Ca"] <- cor(valida$Ca, pred_valCa)^2
# RMSE
valrestuls$RMSE[valrestuls$Property == "Ca"] <- mean((valida$Ca - pred_valCa)^2)^0.5
# ME
valrestuls$ME[valrestuls$Property == "Ca"] <- mean(valida$Ca - pred_valCa)

## Residual variances layer A
residualvariances$layerA[residualvariances$Property == "Ca"] <- var(valida$Ca[valida$layer ==
                    "A"] - pred_valCa[valida$layer == "A"])
## layer B
residualvariances$layerB[residualvariances$Property == "Ca"] <- var(valida$Ca[valida$layer ==
                    "B"] - pred_valCa[valida$layer == "B"])
## layers A and B
residualvariances$layersAB[residualvariances$Property == "Ca"] <- var(valida$Ca -
                    pred_valCa)
```

MBL and predictions for $alr(clay)$...

```
sbl_alrclay <- mbl(Yr = train$alr_Clay, Xr = train$spc, Xu = valida$spc, mblCtrl = ctrl,
                    group = train$samplegroup, dissUsage = "none", k.diss = diss2test, k.range = kminmax
                    pls.c = pls.f, method = rmethod)

## Best threshold distance
idx.best.alrclay <- which.min(sbl_alrclay$nnValStats$st.rmse)
best.kdiss.alrclay <- sbl_alrclay$nnValStats$k.diss[idx.best.alrclay]

## Get the predicted values for the validation set
pred_val_alrclay <- getPredictions(sbl_alrclay)[, idx.best.alrclay]

## R2
```

```
valrestuls$R2[valrestuls$Property == "alr_clay"] <- cor(valida$alr_Clay, pred_val_alrclay)^2
# RMSE
valrestuls$RMSE[valrestuls$Property == "alr_clay"] <- mean((valida$alr_Clay -
                    pred_val_alrclay)^2)^0.5
# ME
valrestuls$ME[valrestuls$Property == "alr_clay"] <- mean(valida$alr_Clay - pred_val_alrclay)

## Residual variances layer A
residualvariances$layerA[residualvariances$Property == "alr_Clay"] <- var(valida$alr_Clay[valida$layer =
                    "A"] - pred_val_alrclay[valida$layer == "A"])
## layer B
residualvariances$layerB[residualvariances$Property == "alr_Clay"] <- var(valida$alr_Clay[valida$layer =
                    "B"] - pred_val_alrclay[valida$layer == "B"])
## layers A and B
residualvariances$layersAB[residualvariances$Property == "alr_Clay"] <- var(valida$alr_Clay -
                    pred_val_alrclay)
```

...and $alr(silt)$...

```
sbl_alrsilt <- mbl(Yr = train$alr_Silt, Xr = train$spc, Xu = valida$spc, mblCtrl = ctrl,
                    group = train$samplegroup, dissUsage = "none", k.diss = diss2test, k.range = kminmax
                    pls.c = pls.f, method = rmethod)

## Best threshold distance
idx.best.alrsilt <- which.min(sbl_alrsilt$nnValStats$st.rmse)
best.kdiss.alrsilt <- sbl_alrsilt$nnValStats$k.diss[idx.best.alrsilt]

## Get the predicted values for the validation set
pred_val_alrsilt <- getPredictions(sbl_alrsilt)[, idx.best.alrsilt]

## R2
valrestuls$R2[valrestuls$Property == "alr_silt"] <- cor(valida$alr_Clay, pred_val_alrsilt)^2
# RMSE
valrestuls$RMSE[valrestuls$Property == "alr_silt"] <- mean((valida$alr_Clay -
                    pred_val_alrsilt)^2)^0.5
# ME
valrestuls$ME[valrestuls$Property == "alr_silt"] <- mean(valida$alr_Clay - pred_val_alrsilt)

## Residual variances layer A
residualvariances$layerA[residualvariances$Property == "alr_Silt"] <- var(valida$alr_Silt[valida$layer =
                    "A"] - pred_val_alrsilt[valida$layer == "A"])
## layer B
residualvariances$layerB[residualvariances$Property == "alr_Silt"] <- var(valida$alr_Silt[valida$layer =
                    "B"] - pred_val_alrsilt[valida$layer == "B"])
## layers A and B
residualvariances$layersAB[residualvariances$Property == "alr_Silt"] <- var(valida$alr_Silt -
                    pred_val_alrsilt)
```

Save the table of the variance of the residuals into your working directory. This data will be later used in the spatial analyses.

```
write.table(x = residualvariances, file = "vnir_residual_variances.txt", sep = "\t",
                    row.names = FALSE)
```

To asses the accuracies and precisions of the predictions (in the validation set) for particle-size distribution, we need to back-transform the additive log-ratio transformed variables to the original clay, silt and sand contents:

```
## Alr back-transformations Clay contents
valClay_alr <- 100 * exp(pred_val_alrclay)/(1 + exp(pred_val_alrclay) + exp(pred_val_alrsilt))
## Silt contents
valSilt_alr <- 100 * exp(pred_val_alrsilt)/(1 + exp(pred_val_alrclay) + exp(pred_val_alrsilt))
## Sand contents
valSand_alr <- 100 * 1/(1 + exp(pred_val_alrclay) + exp(pred_val_alrsilt))
```

Now we can compute the parameters to asses accuracies and precisions:

```r
## Clay content R2
valrestuls$R2[valrestuls$Property == "Clay"] <- (cor(valida$Clay, valClay_alr))^2
# RMSE
valrestuls$RMSE[valrestuls$Property == "Clay"] <- mean((valida$Clay - valClay_alr)^2)^0.5
# ME
valrestuls$ME[valrestuls$Property == "Clay"] <- mean(valida$Clay - valClay_alr)

## Silt content R2
valrestuls$R2[valrestuls$Property == "Silt"] <- (cor(valida$Silt, valSilt_alr))^2
# RMSE
valrestuls$RMSE[valrestuls$Property == "Silt"] <- mean((valida$Silt - valSilt_alr)^2)^0.5
# ME
valrestuls$ME[valrestuls$Property == "Silt"] <- mean(valida$Silt - valSilt_alr)

## Sand content R2
valrestuls$R2[valrestuls$Property == "Sand"] <- (cor(valida$Sand, valSand_alr))^2
# RMSE
valrestuls$RMSE[valrestuls$Property == "Sand"] <- mean((valida$Sand - valSand_alr)^2)^0.5
# ME
valrestuls$ME[valrestuls$Property == "Sand"] <- mean(valida$Sand - valSand_alr)
```

Examine the `valrestuls` object...

```r
valrestuls
```

7.2 Property predictions in the prediction set

After validating the vis-NIR models we can apply them to the prediction set. We can start by creating a `data.frame` where the predictions will be stored. In this `data.frame` the predicted values will be stored under the following variable names: `Ca_spec`, `alr_Clay_spec` and `alr_Silt_spec`:

```r
vnirpredictions <- data.frame(Ca_spec = rep(NA, nrow(pred)), alr_Clay_spec = rep(NA,
                    nrow(pred)), alr_Silt_spec = rep(NA, nrow(pred)))

vnirpredictions <- data.frame(Ca_spec = rep(NA, nrow(pred)), alr_Clay_spec = rep(NA,
                    nrow(pred)), alr_Silt_spec = rep(NA, nrow(pred)))

## Add additional relevant information from the original prediction set
vnirpredictions <- cbind(pred[, c("ID", "POINT_X", "POINT_Y", "set", "Ca", "Clay",
                    "Silt", "Sand", "alr_Clay", "alr_Silt")], vnirpredictions)

## Re-encode the set variable to 'prediction'
vnirpredictions$set <- factor("prediction")
```

For exchangeable Ca^{2+} predictions...

```r
# Predict Ca in the prediction set based on the optimized threshold distance
# for neighbor selection
sbl_ca_pred <- mbl(Yr = train$Ca, Xr = train$spc, Xu = pred$spc, mblCtrl = ctrl,
                    group = train$samplegroup, dissUsage = "none", k.diss = best.kdiss.ca, k.range = kmi
                    pls.c = pls.f, method = rmethod)
## get the predicted values and store them in nirpredictions
vnirpredictions$Ca_spec <- getPredictions(sbl_ca_pred)[, 1]
```

For $alr(clay)$ predictions...

```r
sbl_alrclay_pred <- mbl(Yr = train$alr_Clay, Xr = train$spc, Xu = pred$spc,
                    mblCtrl = ctrl, group = train$samplegroup, dissUsage = "none", k.diss = best.kdiss.a
                    k.range = kminmax, pls.c = pls.f, method = rmethod)
## get the predicted values and store them in nirpredictions
vnirpredictions$alr_Clay_spec <- getPredictions(sbl_alrclay_pred)[, 1]
```

For $alr(silt)$ predictions...

```
sbl_alrsilt_pred <- mbl(Yr = train$alr_Silt, Xr = train$spc, Xu = pred$spc,
                    mblCtrl = ctrl, group = train$samplegroup, dissUsage = "none", k.diss = best.kdiss.a
                    k.range = kminmax, pls.c = pls.f, method = rmethod)
## get the predicted values and store them in nirpredictions
vnirpredictions$alr_Silt_spec <- getPredictions(sbl_alrsilt_pred)[, 1]
```

Examine the `vnirpredictions` object...

```
vnirpredictions
summary(vnirpredictions)
```

Chapter 8

Prepare the vis-NIR augmented dataset

Here we'll create the data that will be used for Spatial modelling. This dataset will contain

- Nr: The arbitrary sample number.
- ID: The `factor` indicating the sample IDs.

- POINT_X: The X (geographical) coordinate.
- POINT_Y: The Y (geographical) coordinate.
- layer: A `factor` indicating the depth layer at which the sample was collected (A: 0-20 cm and B: 80-100 cm).
- set: A `factor` indicating whether the sample was used for vis-NIR calibrations (`train`), for vis-NIR predictions (`prediction`) or if it belongs to model's validation (`validation`). The samples labeled as validation are the same samples initially labeled as validation in the original dataset.
- Ca: The exchangeable Calcium content in the sample ($mmol_c\ kg^1$, measured by conventional laboratory methods)
- Clay: The percentage of clay contnet in the soil sample (measured by conventional laboratory methods).
- Silt: The percentage of silt contnet in the soil sample (measured by conventional laboratory methods).
- Sand: The percentage of sand contnet in the soil sample (measured by conventional laboratory methods).
- alr_Clay: The additive log-ratio transformed clay contnets (measured by conventional laboratory methods).
- alr_Silt: The additive log-ratio transformed silt contnets (measured by conventional laboratory methods).
- Ca_spec: This is the vis-NIR augmented exchangeable Ca^{2+} contents.
- alr_Clay_spec: This is the vis-NIR augmented additive log-ratio transformed clay contnets.

- alr_Silt_spec: This is the vis-NIR augmented additive log-ratio transformed silt contnets.

For the vis-NIR augmented variables (alr_Clay_spc, alr_Silt_spc and Ca_spec) there are three classes of values:

- The values of the samples that are labeled as `train` come from the conventional laboratory methods (e.g. for Ca_spec the values of these samples for this variable are identical to the corresponding values in the variable Ca).

- The values of the samples that are labeled as `prediction` come from the predictions done with the respective vis-NIR model.

- The values of the samples that are labeled as `validation` are treated as missing (i.e. `NA`s).

```r
## samples for the set 'prediction'
vnirpredictions

## samples for the set 'train'
vnirtrain <- train[, c("ID", "POINT_X", "POINT_Y", "set", "Ca", "Clay", "Silt",
                       "Sand", "alr_Clay", "alr_Silt")]
vnirtrain$set <- factor("train")
vnirtrain$Ca_spec <- vnirtrain$Ca
vnirtrain$alr_Clay_spec <- vnirtrain$alr_Clay
vnirtrain$alr_Silt_spec <- vnirtrain$alr_Silt
```

```
## samples for the set 'validation'
vnirvalidation <- valida[, c("ID", "POINT_X", "POINT_Y", "set", "Ca", "Clay",
                    "Silt", "Sand", "alr_Clay", "alr_Silt")]
vnirvalidation$set <- factor(vnirvalidation$set)
vnirvalidation$Ca_spec <- NA
vnirvalidation$alr_Clay_spec <- NA
vnirvalidation$alr_Silt_spec <- NA
```

Now create a single `data.frame` containing the three data sets...

```
vniraugmented <- rbind(vnirtrain, vnirpredictions, vnirvalidation)

vniraugmented$layer <- factor(substr(vniraugmented$ID, 1, 1))

## Reorganize the variables
vniraugmented <- vniraugmented[, c("ID", "POINT_X", "POINT_Y", "layer", "set",
                    "Ca", "Clay", "Silt", "Sand", "alr_Clay", "alr_Silt", "Ca_spec", "alr_Clay_spec",
                    "alr_Silt_spec")]
```

Compute some statistics for the final data set...

```
## Names of the properties
props <- c("Ca", "Clay", "Silt", "Sand", "alr_Clay", "alr_Silt", "Ca_spec",
                    "alr_Clay_spec", "alr_Silt_spec")

## Compute the statistics: mean, standard deviation and the quantiles ('0%',
## '25%', '50%', '75%' and'100%')
statsprops <- aggregate(vniraugmented[, props], by = list(set = vniraugmented$set,
                    layer = vniraugmented$layer), FUN = function(x) {
                    c(mean = mean(as.matrix(x), na.rm = TRUE), sd = sd(as.matrix(x), na.rm = TRUE),
                                        quantile(x, na.rm = TRUE))
})

## Reorganize the object containing the results of the statistics
statsprops <- lapply(props, FUN = function(x, object, ids) {
                    object <- cbind(object[, keep], as.data.frame(statsquant[[x]]))

}, object = statsprops, ids = c("set", "layer"))
names(statsprops) <- props

statsprops <- do.call("rbind", statsprops)
statsprops$property <- gsub(".[0-9]", "", rownames(statsprops))
statsprops[is.na(statsprops)] <- NA

## Reorganize the order of the variables
statsprops <- statsprops[, c("set", "layer", "property", "mean", "sd", "0%",
                    "25%", "50%", "75%", "100%")]

statsprops
```

Optionally, save this data in your working directory

```
write.table(x = vniraugmented, file = "vniraugmented.txt", sep = "\t", row.names = FALSE)
```

Chapter 9

Spatial modeling

In case you have previosly saved both, the vis-NIR augmented data set and the table of the variance of the residuals into your working directory, and you do not have these data in your R enviroment you can execute the code below:

```r
## vniraugmented.txt is spuppposed to be saved in your working directory
vniraugmented <- read.table(file = "vniraugmented.txt", header = TRUE, sep = "\t")
```

If you have saved the table of the variance of the residuals into your working directory you can execute the code below:

```r
## vnir_residual_variances.txt is spuppposed to be saved in your working
## directory
residualvariances <- read.table(file = "vnir_residual_variances.txt", header = TRUE,
                                sep = "\t")
```

Alternatively, you can clean your R enviroment and leave only the data that will be used for the spatial analyses:

```r
## necessary objects
reqobjects2 <- c("vniraugmented", "residualvariances")

## objects to be removed
o2rm2 <- ls()[!ls() %in% reqobjects2]

## remove the objects
rm(list = o2rm2)
```

Split the data and organize it by layers and spatial fit and validation sets

```r
vniraugmented <- as_tibble(vniraugmented)

## split the data sets by layer and by spatial fit and spatial validation
fitlayera <- vniraugmented %>% filter(layer == "A", set != "validation")
fitlayerb <- vniraugmented %>% filter(layer == "B", set != "validation")

vallayera <- vniraugmented %>% filter(layer == "A", set == "validation")
vallayerb <- vniraugmented %>% filter(layer == "B", set == "validation")

plot(vallayera$POINT_X, vallayera$POINT_Y, xlab = "X", ylab = "Y")
points(fitlayera$POINT_X, fitlayera$POINT_Y, col = "red", cex = 1.5)
```

9.1 Robust fitting of the spatial models

Specify some basic aspects/parameters required for fitting the spatial models...

```r
## Define a lag distance for the estimation of the variagram
lagdist <- seq(0, 1500, by = 100)
```

```
## Choose the variogram model
varmodel <- "RMexp"

## Define what parameters need to be adjusted to fit the geo model
fitparam <- default.fit.param(scale = FALSE, alpha = TRUE, variance = FALSE)

## A tuning constant for the robust REML algorithm for the spatial models
## (see tuniing.psi parameter of the georob function)
tpsi <- 2000

## Control some aspects of the spatial predictions
gcntrl <- control.predict.georob(extended.output = TRUE, full.covmat = TRUE)
```

Define the tables where the fitted variogram parameters will be stored

```
## The variables for which a spatial model will be fitted
gprops <- c("Ca", "alr_Clay", "alr_Silt", "Ca_spec", "alr_Clay_spec", "alr_Silt_spec")

## names of the variogram parameters
varparamames <- c("variance", "snugget", "nugget", "scale")

## Create the table
variogtablea <- data.frame(properties = gprops, data = rep(c("Laboratory", "vis-NIR augmented"),
                    each = length(gprops)/2), variance = NA, snugget = NA, nugget = NA, scale = NA)
variogtableb <- variogtablea
```

9.1.1 Laboratory-based data

Here we fit the spatial models of the soil properties whose values comes from conventional laboratory analyes only...

9.1.1.1 Layer A

Exchangeable Ca^{2+}...

```
## Check the sample variogram
vario_Ca_lab_a <- sample.variogram(object = fitlayera$Ca, locations = fitlayera[,
                    c("POINT_X", "POINT_Y")], lag.dist.def = lagdist, estimator = "matheron")
## Plot the sample variogram
plot(x = vario_Ca_lab_a$lag.dist, y = vario_Ca_lab_a$gamma, ylim = c(0, max(vario_Ca_lab_a$gamma)),
                    xlab = "lag distance", ylab = "gamma", pch = 16, col = "red", main = "Ca, laboratory
grid()

## Check the plot above to select some starting values of the variogram
## parameters
startp_Ca_lab_a <- c(variance = 135, nugget = 10, scale = 920)
## Fit
Ca_lab_a <- georob(Ca ~ 1, data = fitlayera, locations = ~POINT_X + POINT_Y,
                    variogram.model = varmodel, param = startp_Ca_lab_a, fit.param = fitparam,
                    tuning.psi = tpsi)
summary(Ca_lab_a)

## Store the variogram parameters
variogtablea[variogtablea$properties == "Ca", varparamames] <- Ca_lab_a$variogram.object[[1]]$param[varp
```

$alr(clay)$

```
## Check the sample variogram
vario_alr_Clay_lab_a <- sample.variogram(object = fitlayera$alr_Clay, locations = fitlayera[,
                    c("POINT_X", "POINT_Y")], lag.dist.def = lagdist, estimator = "matheron")
## Plot the sample variogram
plot(x = vario_alr_Clay_lab_a$lag.dist, y = vario_alr_Clay_lab_a$gamma, ylim = c(0,
                    max(vario_alr_Clay_lab_a$gamma)), xlab = "lag distance", ylab = "gamma",
                    pch = 16, col = "red", main = "alr(clay) laboratory - Layer A")
grid()

## Check the plot above to select some starting values of the variogram
## parameters
startp_alr_Clay_lab_a <- c(variance = 0.8, nugget = 0.3, scale = 1220)

## Fit
alr_Clay_lab_a <- georob(alr_Clay ~ 1, data = fitlayera, locations = ~POINT_X +
                    POINT_Y, variogram.model = varmodel, param = startp_alr_Clay_lab_a, fit.param = fitp
                    tuning.psi = tpsi)
summary(alr_Clay_lab_a)
## Store the variogram parameters
variogtablea[variogtablea$properties == "alr_Clay", varparamames] <- alr_Clay_lab_a$variogram.object[[1
```

alr(*silt*)

```
## Check the sample variogram
vario_alr_Silt_lab_a <- sample.variogram(object = fitlayera$alr_Silt, locations = fitlayera[,
                    c("POINT_X", "POINT_Y")], lag.dist.def = lagdist, estimator = "matheron")
## Plot the sample variogram
plot(x = vario_alr_Silt_lab_a$lag.dist, y = vario_alr_Silt_lab_a$gamma, ylim = c(0,
                    max(vario_alr_Silt_lab_a$gamma)), xlab = "lag distance", ylab = "gamma",
                    pch = 16, col = "red", main = "alr(silt) laboratory - Layer A")
grid()

## Check the plot above to select some starting values of the variogram
## parameters
startp_alr_Silt_lab_a <- c(variance = 1.31, nugget = 0.3, scale = 812)

## Fit
alr_Silt_lab_a <- georob(alr_Silt ~ 1, data = fitlayera, locations = ~POINT_X +
                    POINT_Y, variogram.model = varmodel, param = startp_alr_Silt_lab_a, fit.param = fitp
                    tuning.psi = tpsi)
summary(alr_Silt_lab_a)
## Store the variogram parameters
variogtablea[variogtablea$properties == "alr_Silt", varparamames] <- alr_Silt_lab_a$variogram.object[[1
```

9.1.1.2 Layer B

Exchangeable Ca^{2+}...

```
## Check the sample variogram
vario_Ca_lab_b <- sample.variogram(object = fitlayerb$Ca, locations = fitlayerb[,
                    c("POINT_X", "POINT_Y")], lag.dist.def = lagdist, estimator = "matheron")
## Plot the sample variogram
plot(x = vario_Ca_lab_b$lag.dist, y = vario_Ca_lab_b$gamma, ylim = c(0, max(vario_Ca_lab_b$gamma)),
                    xlab = "lag distance", ylab = "gamma", pch = 16, col = "red", main = "Ca, laboratory
grid()

## Check the plot above to select some starting values of the variogram
## parameters
startp_Ca_lab_b <- c(variance = 127, nugget = 0.1, scale = 850)
## Fit
Ca_lab_b <- georob(Ca ~ 1, data = fitlayerb, locations = ~POINT_X + POINT_Y,
                    variogram.model = varmodel, param = startp_Ca_lab_b, fit.param = fitparam,
                    tuning.psi = tpsi)
summary(Ca_lab_b)
## Store the variogram parameters
variogtableb[variogtableb$properties == "Ca", varparamames] <- Ca_lab_b$variogram.object[[1]]$param[varp
```

alr(*clay*)

```r
## Check the sample variogram
vario_alr_Clay_lab_b <- sample.variogram(object = fitlayerb$alr_Clay, locations = fitlayerb[,
                    c("POINT_X", "POINT_Y")], lag.dist.def = lagdist, estimator = "matheron")
## Plot the sample variogram
plot(x = vario_alr_Clay_lab_b$lag.dist, y = vario_alr_Clay_lab_b$gamma, ylim = c(0,
                    max(vario_alr_Clay_lab_b$gamma)), xlab = "lag distance", ylab = "gamma",
                    pch = 16, col = "red", main = "alr(clay) laboratory - Layer B")
grid()

## Check the plot above to select some starting values of the variogram
## parameters
startp_alr_Clay_lab_b <- c(variance = 1.17, nugget = 0.1, scale = 973, alpha = 0.69)
## Fit
alr_Clay_lab_b <- georob(alr_Clay ~ 1, data = fitlayerb, locations = ~POINT_X +
                    POINT_Y, variogram.model = varmodel, param = startp_alr_Clay_lab_b, fit.param = fitp
                    tuning.psi = tpsi)
summary(alr_Clay_lab_b)
## Store the variogram parameters
variogtableb[variogtableb$properties == "alr_Clay", varparamames] <- alr_Clay_lab_b$variogram.object[[1]
```

alr(*silt*)

```r
## Check the sample variogram
vario_alr_Silt_lab_b <- sample.variogram(object = fitlayerb$alr_Silt, locations = fitlayerb[,
                    c("POINT_X", "POINT_Y")], lag.dist.def = lagdist, estimator = "matheron")
## Plot the sample variogram
plot(x = vario_alr_Silt_lab_b$lag.dist, y = vario_alr_Silt_lab_b$gamma, ylim = c(0,
                    max(vario_alr_Silt_lab_b$gamma)), xlab = "lag distance", ylab = "gamma",
                    pch = 16, col = "red", main = "alr(silt) laboratory - Layer B")
grid()

## Check the plot above to select some starting values of the variogram
## parameters
startp_alr_Silt_lab_b <- c(variance = 1.1, nugget = 0.2, scale = 423)

## Fit
alr_Silt_lab_b <- georob(alr_Silt ~ 1, data = fitlayerb, locations = ~POINT_X +
                    POINT_Y, variogram.model = varmodel, param = startp_alr_Silt_lab_b, fit.param = fitp
                    tuning.psi = tpsi)
summary(alr_Silt_lab_b)
## Store the variogram parameters
variogtableb[variogtableb$properties == "alr_Silt", varparamames] <- alr_Silt_lab_b$variogram.object[[1]
```

9.1.2 Augmented vis-NIR data

Here we fit the spatial models of the soil properties whose values come from the vis-NIR augmented data...

9.1.2.1 Layer A

Exchangeable Ca^{2+} (vis-NIR augmented) ...

```r
# Check the sample variogram
vario_Ca_spec_a <- sample.variogram(object = fitlayera$Ca_spec, locations = fitlayera[,
                    c("POINT_X", "POINT_Y")], lag.dist.def = lagdist, estimator = "matheron")
## Plot the sample variogram
plot(x = vario_Ca_spec_a$lag.dist, y = vario_Ca_spec_a$gamma, ylim = c(0, max(vario_Ca_spec_a$gamma)),
                    xlab = "lag distance", ylab = "gamma", pch = 16, col = "red", main = "Ca, vis-NIR au
grid()

## Check the plot above to select some starting values of the variogram
## parameters
```

```
startp_Ca_spec_a <- c(variance = 112, nugget = 1, scale = 1023)
## Fit
Ca_spec_a <- georob(Ca ~ 1, data = fitlayera, locations = ~POINT_X + POINT_Y,
                    variogram.model = varmodel, param = startp_Ca_spec_a, fit.param = fitparam,
                    tuning.psi = tpsi)
summary(Ca_spec_a)
## Store the variogram parameters
variogtablea[variogtablea$properties == "Ca_spec", varparamames] <- Ca_spec_a$variogram.object[[1]]$para
```

alr(*clay*) (vis-NIR augmented)

```
## Check the sample variogram
vario_alr_Clay_spec_a <- sample.variogram(object = fitlayera$alr_Clay_spec,
                    locations = fitlayera[, c("POINT_X", "POINT_Y")], lag.dist.def = lagdist,
                    estimator = "matheron")
## Plot the sample variogram
plot(x = vario_alr_Clay_spec_a$lag.dist, y = vario_alr_Clay_spec_a$gamma, ylim = c(0,
                    max(vario_alr_Clay_spec_a$gamma)), xlab = "lag distance", ylab = "gamma",
                    pch = 16, col = "red", main = "alr(clay) vis-NIR augmented - Layer A")
grid()

## Check the plot above to select some starting values of the variogram
## parameters
startp_alr_Clay_spec_a <- c(variance = 1.134, nugget = 0.1, scale = 955)

## Fit
alr_Clay_spec_a <- georob(alr_Clay_spec ~ 1, data = fitlayera, locations = ~POINT_X +
                    POINT_Y, variogram.model = varmodel, param = startp_alr_Clay_spec_a, fit.param = fit
                    tuning.psi = tpsi)
summary(alr_Clay_spec_a)
## Store the variogram parameters
variogtablea[variogtablea$properties == "alr_Clay_spec", varparamames] <- alr_Clay_spec_a$variogram.obje
```

alr(*silt*) (vis-NIR augmented)

```
## Check the sample variogram
vario_alr_Silt_spec_a <- sample.variogram(object = fitlayera$alr_Silt_spec,
                    locations = fitlayera[, c("POINT_X", "POINT_Y")], lag.dist.def = lagdist,
                    estimator = "matheron")
## Plot the sample variogram
plot(x = vario_alr_Silt_spec_a$lag.dist, y = vario_alr_Silt_spec_a$gamma, ylim = c(0,
                    max(vario_alr_Silt_spec_a$gamma)), xlab = "lag distance", ylab = "gamma",
                    pch = 16, col = "red", main = "alr(silt) vis-NIR augmented - Layer A")
grid()

## Check the plot above to select some starting values of the variogram
## parameters
startp_alr_Silt_spec_a <- c(variance = 0.845, nugget = 0.1, scale = 1282)

## Fit
alr_Silt_spec_a <- georob(alr_Silt_spec ~ 1, data = fitlayera, locations = ~POINT_X +
                    POINT_Y, variogram.model = varmodel, param = startp_alr_Silt_spec_a, fit.param = fit
                    tuning.psi = tpsi)
summary(alr_Silt_spec_a)
## Store the variogram parameters
variogtablea[variogtablea$properties == "alr_Silt_spec", varparamames] <- alr_Silt_spec_a$variogram.obje
```

9.1.2.2 Layer B

Exchangeable Ca^{2+} (vis-NIR augmented) …

```
## Check the sample variogram
vario_Ca_spec_b <- sample.variogram(object = fitlayerb$Ca_spec, locations = fitlayerb[,
                    c("POINT_X", "POINT_Y")], lag.dist.def = lagdist, estimator = "matheron")
## Plot the sample variogram
plot(x = vario_Ca_spec_b$lag.dist, y = vario_Ca_spec_b$gamma, ylim = c(0, max(vario_Ca_spec_b$gamma)),
                    xlab = "lag distance", ylab = "gamma", pch = 16, col = "red", main = "Ca, vis-NIR au
grid()

## Check the plot above to select some starting values of the variogram
## parameters
startp_Ca_spec_b <- c(variance = 142, nugget = 0.1, scale = 1025)
## Fit
Ca_spec_b <- georob(Ca ~ 1, data = fitlayerb, locations = ~POINT_X + POINT_Y,
                    variogram.model = varmodel, param = startp_Ca_spec_b, fit.param = fitparam,
                    tuning.psi = tpsi)
summary(Ca_spec_b)
## Store the variogram parameters
variogtableb[variogtableb$properties == "Ca_spec", varparamames] <- Ca_spec_b$variogram.object[[1]]$para
```

alr(*clay*) (vis-NIR augmented)

```
## Check the sample variogram
vario_alr_Clay_spec_b <- sample.variogram(object = fitlayerb$alr_Clay_spec,
                    locations = fitlayerb[, c("POINT_X", "POINT_Y")], lag.dist.def = lagdist,
                    estimator = "matheron")
## Plot the sample variogram
plot(x = vario_alr_Clay_spec_b$lag.dist, y = vario_alr_Clay_spec_b$gamma, ylim = c(0,
                    max(vario_alr_Clay_spec_b$gamma)), xlab = "lag distance", ylab = "gamma",
                    pch = 16, col = "red", main = "alr(clay) vis-NIR augmented - Layer B")
grid()

## Check the plot above to select some starting values of the variogram
## parameters
startp_alr_Clay_spec_b <- c(variance = 1.21, nugget = 0.1, scale = 1000, alpha = 0.68)
## Fit
alr_Clay_spec_b <- georob(alr_Clay_spec ~ 1, data = fitlayerb, locations = ~POINT_X +
                    POINT_Y, variogram.model = varmodel, param = startp_alr_Clay_spec_b, fit.param = fit
                    tuning.psi = tpsi)
summary(alr_Clay_spec_b)
## Store the variogram parameters
variogtableb[variogtableb$properties == "alr_Clay_spec", varparamames] <- alr_Clay_spec_b$variogram.obje
```

alr(*silt*) (vis-NIR augmented)

```
## Check the sample variogram
vario_alr_Silt_spec_b <- sample.variogram(object = fitlayerb$alr_Silt_spec,
                    locations = fitlayerb[, c("POINT_X", "POINT_Y")], lag.dist.def = lagdist,
                    estimator = "matheron")
## Plot the sample variogram
plot(x = vario_alr_Silt_spec_b$lag.dist, y = vario_alr_Silt_spec_b$gamma, ylim = c(0,
                    max(vario_alr_Silt_spec_b$gamma)), xlab = "lag distance", ylab = "gamma",
                    pch = 16, col = "red", main = "alr(silt) vis-NIR augmented - Layer B")
grid()

## Check the plot above to select some starting values of the variogram
## parameters
startp_alr_Silt_spec_b <- c(variance = 1.43, nugget = 0.1, scale = 850, alpha = 0.63)
## Fit
alr_Silt_spec_b <- georob(alr_Silt_spec ~ 1, data = fitlayerb, locations = ~POINT_X +
                    POINT_Y, variogram.model = varmodel, param = startp_alr_Silt_spec_b, fit.param = fit
                    tuning.psi = tpsi)
summary(alr_Silt_spec_b)
## Store the variogram parameters
variogtableb[variogtableb$properties == "alr_Silt_spec", varparamames] <- alr_Silt_spec_b$variogram.obje
```

9.2 Accounting for vis-NIR model errors in the spatial models

Our vis-NIR models (used to create the vis-NIR augmented data) have an error which we estimated in previous sections and which is stored in the `residualvariances` object. The uncertainty of these models was propagated through spatial predictions in order to obtain more realistic performance results of the spatial predictions. We followed the approach given in Viscarra Rossel et al. (2016b) where the variance of the residuals of the vis-NIR predictions at each layer is used for propagating the vis-NIR erros (see section *Spectroscopicmodel error* in our paper). For the vis-NIR augmented data we assume that the uncertainty of the whole set is given by the variances of the predictions. However, this dataset contains both, laboratory data (aprox. 26%) and vis-NIR predicted data (aprox. 74%), therefore the variance we assume here is expected to be larger than the actual one in the augmented data. You can try to account for this.

```
## Add the residual variances to the snugget parameter of the variograms
## corresponding to the vis-NIR augmented data Note that snugget was 0 for
## all the fitted variograms Ca (layer A)
variogtablea$snugget[variogtablea$properties == "Ca_spec"] <- residualvariances$layerA[residualvariances
                    "Ca"]
## alr(clay) (layer A)
variogtablea$snugget[variogtablea$properties == "alr_Clay_spec"] <- residualvariances$layerA[residualvar
                    "alr_Clay"]
## alr(silt) (layer A)
variogtablea$snugget[variogtablea$properties == "alr_Silt_spec"] <- residualvariances$layerA[residualvar
                    "alr_Silt"]

## Ca (layer B)
variogtableb$snugget[variogtableb$properties == "Ca_spec"] <- residualvariances$layerB[residualvariances
                    "Ca"]
## alr(clay) (layer B)
variogtableb$snugget[variogtableb$properties == "alr_Clay_spec"] <- residualvariances$layerB[residualvar
                    "alr_Clay"]
## alr(silt) (layer B)
variogtableb$snugget[variogtableb$properties == "alr_Silt_spec"] <- residualvariances$layerB[residualvar
                    "alr_Silt"]
```

9.3 Validation of the spatial models

Create a `data.frame` for each layer to store the predictions in the validation set and another two to store the validation results...

```
## The final variables to be predicted. In the case of Clay, silt and Sand
## they are estimated from the predictions of alr(clay) and alr(silt)
bprops <- c("Ca", "Clay", "Silt", "Sand", "Ca_spec", "Clay_spec", "Silt_spec",
                    "Sand_spec")

## a quick function to create columns of a given length (lg)
idfcol <- function(x, lg) {
                    data.frame(lg, fix.empty.names = FALSE)
}

## object where the spatial predictions will be stored
sppredsvala <- sapply(c("ID", bprops), FUN = idfcol, lg = rep(NA, nrow(vallayera)))
sppredsvala <- data.frame(do.call("cbind", sppredsvala))
sppredsvala$ID <- vallayera$ID

sppredsvalb <- sapply(c("ID", bprops), FUN = idfcol, lg = rep(NA, nrow(vallayerb)))
sppredsvalb <- data.frame(do.call("cbind", sppredsvalb))
sppredsvalb$ID <- vallayerb$ID

## object where the validation of the spatial predictions will be stored
spvala <- data.frame(properties = bprops, data = rep(c("Laboratory", "vis-NIR augmented"),
                    each = length(bprops)/2), R2 = NA, RMSE = NA, ME = NA)
spvalb <- spvala
```

9.3.1 Laboratory-based data

9.3.1.1 Layer A

Exchangeable Ca^{2+}...

```
pred_Ca_lab_a <- predict(Ca_lab_a, newdata = as.data.frame(vallayera), control = gcntrl)

sppredsvala$Ca <- pred_Ca_lab_a$pred$pred

spvala$R2[spvala$properties == "Ca"] <- cor(vallayera$Ca, sppredsvala$Ca)^2
spvala$RMSE[spvala$properties == "Ca"] <- mean((vallayera$Ca - sppredsvala$Ca)^2)^0.5
spvala$ME[spvala$properties == "Ca"] <- mean(vallayera$Ca - sppredsvala$Ca)
```

Estimation of clay, silt and sand contents from the predictions of $alr(clay)$ and $alr(silt)$...

```
## predict alr(clay)
pred_alr_Clay_lab_a <- predict(alr_Clay_lab_a, newdata = as.data.frame(vallayera),
                    control = gcntrl)
## predict alr(silt)
pred_alr_Silt_lab_a <- predict(alr_Silt_lab_a, newdata = as.data.frame(vallayera),
                    control = gcntrl)

## Back-transfrom to clay, silt and sand contents
dvr.Silt <- exp(pred_alr_Silt_lab_a$pred$pred + (0.5 * (pred_alr_Silt_lab_a$pred$var.target -
                    pred_alr_Silt_lab_a$pred$cov.pred.target)))
dvr.Clay <- exp(pred_alr_Clay_lab_a$pred$pred + (0.5 * (pred_alr_Clay_lab_a$pred$var.target -
                    pred_alr_Clay_lab_a$pred$cov.pred.target)))
dvn.pred <- 1 + dvr.Silt + dvr.Clay

## Store the back-transformed values
sppredsvala$Clay <- 100 * (dvr.Clay/dvn.pred)
sppredsvala$Silt <- 100 * (dvr.Silt/dvn.pred)
sppredsvala$Sand <- 100/dvn.pred

## Estimate the validation parameters for Clay
spvala$R2[spvala$properties == "Clay"] <- cor(vallayera$Clay, sppredsvala$Clay)^2
spvala$RMSE[spvala$properties == "Clay"] <- mean((vallayera$Clay - sppredsvala$Clay)^2)^0.5
spvala$ME[spvala$properties == "Clay"] <- mean(vallayera$Clay - sppredsvala$Clay)
## Silt
spvala$R2[spvala$properties == "Silt"] <- cor(vallayera$Silt, sppredsvala$Silt)^2
spvala$RMSE[spvala$properties == "Silt"] <- mean((vallayera$Silt - sppredsvala$Silt)^2)^0.5
spvala$ME[spvala$properties == "Silt"] <- mean(vallayera$Silt - sppredsvala$Silt)
## Sand
spvala$R2[spvala$properties == "Sand"] <- cor(vallayera$Sand, sppredsvala$Sand)^2
spvala$RMSE[spvala$properties == "Sand"] <- mean((vallayera$Sand - sppredsvala$Sand)^2)^0.5
spvala$ME[spvala$properties == "Sand"] <- mean(vallayera$Sand - sppredsvala$Sand)
```

9.3.1.2 Layer B

Exchangeable Ca^{2+}...

```
pred_Ca_lab_b <- predict(Ca_lab_b, newdata = as.data.frame(vallayerb), control = gcntrl)

sppredsvalb$Ca <- pred_Ca_lab_b$pred$pred

spvalb$R2[spvala$properties == "Ca"] <- cor(vallayerb$Ca, sppredsvalb$Ca)^2
spvalb$RMSE[spvala$properties == "Ca"] <- mean((vallayerb$Ca - sppredsvalb$Ca)^2)^0.5
spvalb$ME[spvala$properties == "Ca"] <- mean(vallayerb$Ca - sppredsvalb$Ca)
```

Estimation of clay, silt and sand contents from the predictions of $alr(clay)$ and $alr(silt)$...

```
## predict alr(clay)
pred_alr_Clay_lab_b <- predict(alr_Clay_lab_b, newdata = as.data.frame(vallayerb),
                    control = gcntrl)
## predict alr(silt)
pred_alr_Silt_lab_b <- predict(alr_Silt_lab_b, newdata = as.data.frame(vallayerb),
                    control = gcntrl)

## Back-transfrom to clay, silt and sand contents
dvr.Silt <- exp(pred_alr_Silt_lab_b$pred$pred + (0.5 * (pred_alr_Silt_lab_b$pred$var.target -
                    pred_alr_Silt_lab_b$pred$cov.pred.target)))
dvr.Clay <- exp(pred_alr_Clay_lab_b$pred$pred + (0.5 * (pred_alr_Clay_lab_b$pred$var.target -
                    pred_alr_Clay_lab_b$pred$cov.pred.target)))
dvn.pred <- 1 + dvr.Silt + dvr.Clay

## Store the back-transformed values
sppredsvalb$Clay <- 100 * (dvr.Clay/dvn.pred)
sppredsvalb$Silt <- 100 * (dvr.Silt/dvn.pred)
sppredsvalb$Sand <- 100/dvn.pred

## Estimate the validation parameters for Clay
spvalb$R2[spvala$properties == "Clay"] <- cor(vallayerb$Clay, sppredsvalb$Clay)^2
spvalb$RMSE[spvala$properties == "Clay"] <- mean((vallayerb$Clay - sppredsvalb$Clay)^2)^0.5
spvalb$ME[spvala$properties == "Clay"] <- mean(vallayerb$Clay - sppredsvalb$Clay)
## Silt
spvalb$R2[spvala$properties == "Silt"] <- cor(vallayerb$Silt, sppredsvalb$Silt)^2
spvalb$RMSE[spvala$properties == "Silt"] <- mean((vallayerb$Silt - sppredsvalb$Silt)^2)^0.5
spvalb$ME[spvala$properties == "Silt"] <- mean(vallayerb$Silt - sppredsvalb$Silt)
## Sand
spvalb$R2[spvala$properties == "Sand"] <- cor(vallayerb$Sand, sppredsvalb$Sand)^2
spvalb$RMSE[spvala$properties == "Sand"] <- mean((vallayerb$Sand - sppredsvalb$Sand)^2)^0.5
spvalb$ME[spvala$properties == "Sand"] <- mean(vallayerb$Sand - sppredsvalb$Sand)
```

9.3.2 Vis-NIR augmented-based data

Here, for the spatial predictions in the vis-NIR augmented dataset we use the variogram parameters which include the residual variances of the vis-NIR models. For this we use the `param`argument of the `predict` function in the `georob` pacakge. #### Layer A Exchangeable Ca^{2+}...

```
pred_Ca_spec_a <- predict(Ca_spec_a, newdata = as.data.frame(vallayera), control = gcntrl,
                    param = unlist(variogtablea[variogtablea$properties == "Ca_spec", varparamames]))


sppredsvala$Ca_spec <- pred_Ca_spec_a$pred$pred

spvala$R2[spvala$properties == "Ca_spec"] <- cor(vallayera$Ca, sppredsvala$Ca_spec)^2
spvala$RMSE[spvala$properties == "Ca_spec"] <- mean((vallayera$Ca - sppredsvala$Ca_spec)^2)^0.5
spvala$ME[spvala$properties == "Ca_spec"] <- mean(vallayera$Ca - sppredsvala$Ca_spec)
```

Estimation of clay, silt and sand contents from the predictions of $alr(clay)$ and $alr(silt)$...

```
## predict alr(clay)
pred_alr_Clay_spec_a <- predict(alr_Clay_spec_a, newdata = as.data.frame(vallayera),
                    control = gcntrl, param = unlist(variogtablea[variogtablea$properties ==
                                        "alr_Clay_spec", varparamames]))
## predict alr(silt)
pred_alr_Silt_spec_a <- predict(alr_Silt_spec_a, newdata = as.data.frame(vallayera),
                    control = gcntrl, param = unlist(variogtablea[variogtablea$properties ==
                                        "alr_Silt_spec", varparamames]))

## Back-transfrom to clay, silt and sand contents
dvr.Silt <- exp(pred_alr_Silt_spec_a$pred$pred + (0.5 * (pred_alr_Silt_spec_a$pred$var.target -
                    pred_alr_Silt_spec_a$pred$cov.pred.target)))
```

```
dvr.Clay <- exp(pred_alr_Clay_spec_a$pred$pred + (0.5 * (pred_alr_Clay_spec_a$pred$var.target -
                    pred_alr_Clay_spec_a$pred$cov.pred.target)))
dvn.pred <- 1 + dvr.Silt + dvr.Clay

## Store the back-transformed values
sppredsvala$Clay_spec <- 100 * (dvr.Clay/dvn.pred)
sppredsvala$Silt_spec <- 100 * (dvr.Silt/dvn.pred)
sppredsvala$Sand_spec <- 100/dvn.pred

## Estimate the validation parameters for Clay
spvala$R2[spvala$properties == "Clay_spec"] <- cor(vallayera$Clay, sppredsvala$Clay_spec)^2
spvala$RMSE[spvala$properties == "Clay_spec"] <- mean((vallayera$Clay - sppredsvala$Clay_spec)^2)^0.5
spvala$ME[spvala$properties == "Clay_spec"] <- mean(vallayera$Clay - sppredsvala$Clay_spec)
## Silt
spvala$R2[spvala$properties == "Silt_spec"] <- cor(vallayera$Silt, sppredsvala$Silt_spec)^2
spvala$RMSE[spvala$properties == "Silt_spec"] <- mean((vallayera$Silt - sppredsvala$Silt_spec)^2)^0.5
spvala$ME[spvala$properties == "Silt_spec"] <- mean(vallayera$Silt - sppredsvala$Silt_spec)
## Sand
spvala$R2[spvala$properties == "Sand_spec"] <- cor(vallayera$Sand, sppredsvala$Sand_spec)^2
spvala$RMSE[spvala$properties == "Sand_spec"] <- mean((vallayera$Sand - sppredsvala$Sand_spec)^2)^0.5
spvala$ME[spvala$properties == "Sand_spec"] <- mean(vallayera$Sand - sppredsvala$Sand_spec)
```

9.3.2.1 Layer B

Exchangeable Ca^{2+}...

```
pred_Ca_spec_b <- predict(Ca_spec_b, newdata = as.data.frame(vallayerb), control = gcntrl,
                    param = unlist(variogtableb[variogtableb$properties == "Ca_spec", varparamames]))

sppredsvalb$Ca_spec <- pred_Ca_spec_b$pred$pred

spvalb$R2[spvala$properties == "Ca_spec"] <- cor(vallayerb$Ca, sppredsvalb$Ca_spec)^2
spvalb$RMSE[spvala$properties == "Ca_spec"] <- mean((vallayerb$Ca - sppredsvalb$Ca_spec)^2)^0.5
spvalb$ME[spvala$properties == "Ca_spec"] <- mean(vallayerb$Ca - sppredsvalb$Ca_spec)
```

Estimation of clay, silt and sand contents from the predictions of $alr(clay)$ and $alr(silt)$...

```
## predict alr(clay)
pred_alr_Clay_spec_b <- predict(alr_Clay_spec_b, newdata = as.data.frame(vallayerb),
                    control = gcntrl, param = unlist(variogtableb[variogtableb$properties ==
                                        "alr_Clay_spec", varparamames]))
## predict alr(silt)
pred_alr_Silt_spec_b <- predict(alr_Silt_spec_b, newdata = as.data.frame(vallayerb),
                    control = gcntrl, param = unlist(variogtableb[variogtableb$properties ==
                                        "alr_Silt_spec", varparamames]))

## Back-transfrom to clay, silt and sand contents
dvr.Silt <- exp(pred_alr_Silt_spec_b$pred$pred + (0.5 * (pred_alr_Silt_spec_b$pred$var.target -
                    pred_alr_Silt_spec_b$pred$cov.pred.target)))
dvr.Clay <- exp(pred_alr_Clay_spec_b$pred$pred + (0.5 * (pred_alr_Clay_spec_b$pred$var.target -
                    pred_alr_Clay_spec_b$pred$cov.pred.target)))
dvn.pred <- 1 + dvr.Silt + dvr.Clay

## Store the back-transformed values
sppredsvalb$Clay_spec <- 100 * (dvr.Clay/dvn.pred)
sppredsvalb$Silt_spec <- 100 * (dvr.Silt/dvn.pred)
sppredsvalb$Sand_spec <- 100/dvn.pred

## Estimate the validation parameters for Clay
spvalb$R2[spvala$properties == "Clay_spec"] <- cor(vallayerb$Clay, sppredsvalb$Clay_spec)^2
spvalb$RMSE[spvala$properties == "Clay_spec"] <- mean((vallayerb$Clay - sppredsvalb$Clay_spec)^2)^0.5
spvalb$ME[spvala$properties == "Clay_spec"] <- mean(vallayerb$Clay - sppredsvalb$Clay_spec)
```

```
## Silt
spvalb$R2[spvala$properties == "Silt_spec"] <- cor(vallayerb$Silt, sppredsvalb$Silt_spec)^2
spvalb$RMSE[spvala$properties == "Silt_spec"] <- mean((vallayerb$Silt - sppredsvalb$Silt_spec)^2)^0.5
spvalb$ME[spvala$properties == "Silt_spec"] <- mean(vallayerb$Silt - sppredsvalb$Silt_spec)
## Sand
spvalb$R2[spvala$properties == "Sand_spec"] <- cor(vallayerb$Sand, sppredsvalb$Sand_spec)^2
spvalb$RMSE[spvala$properties == "Sand_spec"] <- mean((vallayerb$Sand - sppredsvalb$Sand_spec)^2)^0.5
spvalb$ME[spvala$properties == "Sand_spec"] <- mean(vallayerb$Sand - sppredsvalb$Sand_spec)
```

9.4 Mapping

Now that we have validated the spatial models for both laboratory data and vis-NIR augmented data, we proceed to produce the maps of each property at each layer. First we have to read the polygon corresponding to the study area. Download the polygon to your working directory. This is a `R` object of class (`SpatialPolygonsDataFrame` of the package `sp`) which contains the polygon of the study area. Click here to download it. If you saved the file in your working directory you can:

```
polyfile <- file("polygon.rds")
shape <- readRDS(polyfile)
shape
```

or...

```
polyfile <- file("https://github.com/l-ramirez-lopez/VNIR_spectroscopy_for_robust_soil_mapping/raw/maste
shape <- readRDS(polyfile)
shape
```

Now create a template for the spatial predictions

```
## function to convert polygon to raster
pol2raster <- function(r, nrows = 10, ncols = 10) {
                    ext <- raster(extent(r), nrows, ncols)
                    crs(ext) <- crs(r)
                    fr <- rasterize(r, ext, field = 1, update = TRUE)
                    return(fr)
}

rncol <- (extent(shape)@ymax - extent(shape)@ymin)/10
rnrow <- (extent(shape)@xmax - extent(shape)@xmin)/10

rasg <- pol2raster(shape, rncol, rnrow)

## resolution here you can choose the resolution for the predicted maps for
## our paper we set this value to 10, however here we will set it to 50 for
## the sake of memory if you have a decent machine you can go finer
mresolution <- 50

rasg <- raster::resample(rasg, raster(resolution = mresolution, ext = extent(shape)),
                    method = "ngb")
rasg
plot(rasg)

## and here we have our template
sppx <- as(rasg, "SpatialPixelsDataFrame")
colnames(sppx@coords) <- c("POINT_X", "POINT_Y")
```

Create a `data.frame` for each layer to store the predicted values for the maps

```
spatialpredsa <- data.frame(matrix(NA, nrow(sppx), length(bprops)))
colnames(spatialpredsa) <- bprops
spatialpredsb <- spatialpredsa
```

9.4.1 Laboratory-based data

9.4.1.1 Layer A

Exchangeable Ca^{2+}...

```
spatialpredsa$Ca <- predict(Ca_lab_a, newdata = sppx)$pred
```

Clay, sand and silt contents..

```
alr_Clay_lab_a_map <- predict(alr_Clay_lab_a, newdata = sppx)
alr_Silt_lab_a_map <- predict(alr_Silt_lab_a, newdata = sppx)

spatialpredsa$Clay <- 100 * exp(alr_Clay_lab_a_map$pred)/(1 + exp(alr_Silt_lab_a_map$pred) +
                    exp(alr_Clay_lab_a_map$pred))
spatialpredsa$Silt <- 100 * exp(alr_Silt_lab_a_map$pred)/(1 + exp(alr_Silt_lab_a_map$pred) +
                    exp(alr_Clay_lab_a_map$pred))
spatialpredsa$Sand <- 100/(1 + exp(alr_Silt_lab_a_map$pred) + exp(alr_Clay_lab_a_map$pred))
```

9.4.1.2 Layer B

Exchangeable Ca^{2+}...

```
spatialpredsb$Ca <- predict(Ca_lab_b, newdata = sppx)$pred
```

Clay, sand and silt contents..

```
alr_Clay_lab_b_map <- predict(alr_Clay_lab_b, newdata = sppx)
alr_Silt_lab_b_map <- predict(alr_Silt_lab_b, sppx)

spatialpredsb$Clay <- 100 * exp(alr_Clay_lab_b_map$pred)/(1 + exp(alr_Silt_lab_b_map$pred) +
                    exp(alr_Clay_lab_b_map$pred))
spatialpredsb$Silt <- 100 * exp(alr_Silt_lab_b_map$pred)/(1 + exp(alr_Silt_lab_b_map$pred) +
                    exp(alr_Clay_lab_b_map$pred))
spatialpredsb$Sand <- 100/(1 + exp(alr_Silt_lab_b_map$pred) + exp(alr_Clay_lab_b_map$pred))
```

9.4.2 Vis-NIR augmented-based data

9.4.2.1 Layer A

Exchangeable Ca^{2+}...

```
spatialpredsa$Ca_spec <- predict(Ca_spec_a, newdata = sppx, param = unlist(variogtablea[variogtablea$pro
                    "Ca_spec", varparamames]))$pred
```

Clay, sand and silt contents..

```
alr_Clay_spec_a_map <- predict(alr_Clay_spec_a, newdata = sppx, param = unlist(variogtablea[variogtablea
                    "alr_Silt_spec", varparamames]))
alr_Silt_spec_a_map <- predict(alr_Silt_spec_a, newdata = sppx, param = unlist(variogtablea[variogtablea
                    "alr_Clay_spec", varparamames]))

spatialpredsa$Clay_spec <- 100 * exp(alr_Clay_spec_a_map$pred)/(1 + exp(alr_Silt_spec_a_map$pred) +
                    exp(alr_Clay_spec_a_map$pred))
spatialpredsa$Silt_spec <- 100 * exp(alr_Silt_spec_a_map$pred)/(1 + exp(alr_Silt_spec_a_map$pred) +
                    exp(alr_Clay_spec_a_map$pred))
spatialpredsa$Sand_spec <- 100/(1 + exp(alr_Silt_spec_a_map$pred) + exp(alr_Clay_spec_a_map$pred))
```

9.4.2.2 Layer B

Exchangeable Ca^{2+}...

```
spatialpredsb$Ca_spec <- predict(Ca_spec_b, newdata = sppx, param = unlist(variogtableb[variogtableb$pro
                    "Ca_spec", varparamames]))$pred
```

Clay, sand and silt contents..

```
alr_Clay_spec_b_map <- predict(alr_Clay_spec_b, newdata = sppx, param = unlist(variogtableb[variogtablel
                    "alr_Silt_spec", varparamames]))
alr_Silt_spec_b_map <- predict(alr_Silt_spec_b, newdata = sppx, param = unlist(variogtableb[variogtablel
                    "alr_Clay_spec", varparamames]))

spatialpredsb$Clay_spec <- 100 * exp(alr_Clay_spec_b_map$pred)/(1 + exp(alr_Silt_spec_b_map$pred) +
                    exp(alr_Clay_spec_b_map$pred))
spatialpredsb$Silt_spec <- 100 * exp(alr_Silt_spec_b_map$pred)/(1 + exp(alr_Silt_spec_b_map$pred) +
                    exp(alr_Clay_spec_b_map$pred))
spatialpredsb$Sand_spec <- 100/(1 + exp(alr_Silt_spec_b_map$pred) + exp(alr_Clay_spec_b_map$pred))
```

9.4.3 Plots

```
## compute the differences between maps (layer)
spatialpredsa$Cadiff <- spatialpredsa$Ca - spatialpredsa$Ca_spec
spatialpredsa$Claydiff <- spatialpredsa$Clay - spatialpredsa$Clay_spec
spatialpredsa$Siltdiff <- spatialpredsa$Silt - spatialpredsa$Silt_spec
spatialpredsa$Sanddiff <- spatialpredsa$Sand - spatialpredsa$Sand_spec

## compute the differences between maps (layer B)
spatialpredsb$Cadiff <- spatialpredsb$Ca - spatialpredsb$Ca_spec
spatialpredsb$Claydiff <- spatialpredsb$Clay - spatialpredsb$Clay_spec
spatialpredsb$Siltdiff <- spatialpredsb$Silt - spatialpredsb$Silt_spec
spatialpredsb$Sanddiff <- spatialpredsb$Sand - spatialpredsb$Sand_spec
```

Create a new `data.frame` for plotting purposes

```
pred_layer_a <- data.frame(rbind(sppx@coords, sppx@coords, sppx@coords), layer = "Depth A (0 - 0.2 m)",
                    method = c(rep("Laboratory-based", nrow(spatialpredsa)), rep("vis-NIR augmented",
                              nrow(spatialpredsa)), rep("Differences between maps", nrow(spati
                    Ca = unlist(spatialpredsa[, c("Ca", "Ca_spec", "Cadiff")]), Clay = unlist(spatialpre
                              c("Clay", "Clay_spec", "Claydiff")]), Silt = unlist(spatialpreds
                              c("Silt", "Silt_spec", "Siltdiff")]), Sand = unlist(spatialpreds
                              c("Sand", "Sand_spec", "Sanddiff")]))

pred_layer_b <- data.frame(rbind(sppx@coords, sppx@coords, sppx@coords), layer = "Depth B (0.8 - 1.0 m)"
                    method = c(rep("Laboratory-based", nrow(spatialpredsb)), rep("vis-NIR augmented",
                              nrow(spatialpredsb)), rep("Differences between maps", nrow(spati
                    Ca = unlist(spatialpredsb[, c("Ca", "Ca_spec", "Cadiff")]), Clay = unlist(spatialpre
                              c("Clay", "Clay_spec", "Claydiff")]), Silt = unlist(spatialpreds
                              c("Silt", "Silt_spec", "Siltdiff")]), Sand = unlist(spatialpreds
                              c("Sand", "Sand_spec", "Sanddiff")]))

finalmapping <- data.frame(rbind(pred_layer_a, pred_layer_b))
```

Define palette of color for the plot

```
myPalette <- colorRampPalette(rev(brewer.pal(11, "Spectral")), space = "Lab")
```

Define some ggplot parameters common to all plots

```
gfg <- facet_grid(layer ~ method)
gscf <- scale_fill_gradientn(colours = myPalette(30))
gcoord <- coord_equal()
gtheme <- theme_bw() + theme(legend.position = "top") + theme(axis.title.y = element_text(colour = grey(
                    size = 25), axis.text.y = element_text(angle = 0, vjust = 0.5, hjust = 0.5,
                    size = 16), legend.title = element_text(colour = "black", size = 25)) +
                    theme(axis.title.x = element_text(colour = grey(0.2), size = 25), axis.text.x = elem
                              vjust = 0, size = 16)) + theme(legend.text = element_text(colour
                    size = 15), legend.key.size = unit(0.95, "cm")) + theme(strip.background = element_r
                    strip.text.x = element_text(size = 25, colour = "black", angle = 0), strip.text.y =
                              colour = "black", angle = -90))
```

```
glabs <- labs(y = "Northings /m", x = "Eastings /m")
```

Create the maps for Ca^{2+}

```
ca_map <- ggplot(finalmapping[finalmapping$method != "Differences between maps",
                    ], aes(POINT_X, POINT_Y)) + geom_tile(aes(fill = Ca)) + gfg + gscf + gcoord +
                    gtheme + glabs + guides(fill = guide_colourbar(title = expression(paste("",
                    Ca^"++", " "/mmol[c], kg^-1, " "))))


ca_map_diff <- ggplot(finalmapping[finalmapping$method == "Differences between maps",
                    ], aes(POINT_X, POINT_Y)) + geom_tile(aes(fill = Ca)) + gfg + gscf + gcoord +
                    gtheme + glabs + guides(fill = guide_colourbar(title = expression(paste("Difference"
                    " "/mmol[c], kg^-1, " "))))

ca_map
ca_map_diff
```

Create the maps for *Clay*

```
Clay_map <- ggplot(finalmapping[finalmapping$method != "Differences between maps",
                    ], aes(POINT_X, POINT_Y)) + geom_tile(aes(fill = Clay)) + gfg + gscf + gcoord +
                    gtheme + glabs + guides(fill = guide_colourbar(title = "Clay content, % "))


Clay_map_diff <- ggplot(finalmapping[finalmapping$method == "Differences between maps",
                    ], aes(POINT_X, POINT_Y)) + geom_tile(aes(fill = Clay)) + gfg + gscf + gcoord +
                    gtheme + glabs + guides(fill = guide_colourbar(title = "Difference, %"))

Clay_map
Clay_map_diff
```

Create the maps for *Silt*

```
Silt_map <- ggplot(finalmapping[finalmapping$method != "Differences between maps",
                    ], aes(POINT_X, POINT_Y)) + geom_tile(aes(fill = Silt)) + gfg + gscf + gcoord +
                    gtheme + glabs + guides(fill = guide_colourbar(title = "Silt content, % "))


Silt_map_diff <- ggplot(finalmapping[finalmapping$method == "Differences between maps",
                    ], aes(POINT_X, POINT_Y)) + geom_tile(aes(fill = Silt)) + gfg + gscf + gcoord +
                    gtheme + glabs + guides(fill = guide_colourbar(title = "Difference, %"))

Silt_map
Silt_map_diff
```

Create the maps for *Sand*

```
Sand_map <- ggplot(finalmapping[finalmapping$method != "Differences between maps",
                    ], aes(POINT_X, POINT_Y)) + geom_tile(aes(fill = Sand)) + gfg + gscf + gcoord +
                    gtheme + glabs + guides(fill = guide_colourbar(title = "Sand content, % "))


Sand_map_diff <- ggplot(finalmapping[finalmapping$method == "Differences between maps",
                    ], aes(POINT_X, POINT_Y)) + geom_tile(aes(fill = Sand)) + gfg + gscf + gcoord +
                    gtheme + glabs + guides(fill = guide_colourbar(title = "Difference, %"))

Sand_map
Sand_map_diff
```

Odeh, I.O., Todd, A.J. & Triantafilis, J. 2003. Spatial prediction of soil particle-size fractions as compositional data. Soil Science, 168, 501–515.

Ramirez-Lopez, L., Wadoux, A.C., Franceschini, M.H.D., Terra, F.S., Marques, K.P.P., Sayão, V.M. and Demattê, J.A.M., 2019. Robust soil mapping at the farm scale with vis–NIR spectroscopy. European Journal of Soil Science.

Viscarra Rossel, R., Brus, D.J., Lobsey, C., Shi, Z. &McLachlan, G. 2016. Baseline estimates of soil organic carbon by proximal sensing: comparing design-based, model-assisted and model-based inference. Geoderma,265, 152–163.