Chapter 9
Estimating Soil Properties and Classes from Spectra

The most common way of estimating soil properties from pre-processed spectra is by calibrating a statistical model. If the response of the spectra at a particular wavelength follows the Beer-Lambert law, the degree of reflectance at a particular wavelength is proportional to the concentration of a soil property. In this case, a linear model can be fitted between this wavelength and the measured values of a soil property. In most cases, however, the response of the spectrum follows a complex form, i.e. the concentration of a soil property is related to several interacting wavelengths and overlapping regions of the spectrum. In recent years, chemometric methods based on multivariate statistical models and machine learning algorithms have considered the entire spectrum as a predictor. When there are many hundreds of predictor variables (wavelengths), the methods can be described as multivariate. Multivariate models can be calibrated using the whole spectrum, with the target variables being measured values of soil properties.

Predictions made by a calibrated model need to be validated. Three common validation methods exist: data splitting, cross-validation and additional probability sampling. In digital soil spectroscopy, one most often has only a single dataset for both calibration and validation. Collecting an additional probability sample to independently validate soil spectral models is generally not feasible. Both data splitting and cross-validation are sub-optimal compared to collecting an additional probability sample because the information contained in the dataset cannot be fully exploited during calibration. In most cases, unfortunately, this is the only option.

In data splitting, the dataset is split into two subsets, generally containing 75% and 25% of the data, which are used for calibration and validation, respectively. In cross-validation, the dataset is split into K subsamples, where the $K-1$ subsamples are used for calibration, and validation statistics are computed from the subsample left aside. Each soil sample is used once for validation. Cross-validation should be preferred over simple data splitting, especially when the dataset is small (Brus et al. 2011).

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2021

A. M. J.-C. Wadoux et al., *Soil Spectral Inference with R*, Progress in Soil Science,
https://doi.org/10.1007/978-3-030-64896-1_9

In this chapter, we use data-splitting for the sake of demonstration. More information on validation methods can be found in the statistical (e.g. Friedman et al. 2001, Chapter 7) or pedometrics (e.g. Brus et al. 2011) literature.

The set of packages used in this chapter are installed using the lines below. The book-associated `soilspec` package is also required; see Chap. 3 for information on its installation.

```
# specify all the packages used in the chapter and install them if they are not already
myPackages <- c("caret", "ggplot2", "soiltexture", "resemble",
                "randomForest", "Cubist", "lattice", "pls",
                "prospectr", "RcppArmadillo")

# define which packages are not installed in the current computer
notInstalled <- myPackages[!(myPackages %in% installed.packages()[ , "Package"])]

# install the missing packages
if(length(notInstalled)>0) install.packages(notInstalled)
```

9.1 Goodness of Fit Measures

The process of creating a model begins first with a calibration. After a model is calibrated, we can use it to make a prediction on new samples. An important step in the analysis is to evaluate the calibrated model by predicting the value of the soil property and to comparing it with its associated measured value.

In the following sections, we will review the most common indicators of the quality of a prediction made by a calibrated model. These indicators are routinely employed in soil spectroscopy and in the general statistical modelling literature.

Root mean square error (RMSE) The root mean square error (RMSE) is the standard deviation of the residuals between observed and predicted values of a variable. The RMSE evaluates the dispersion of the residuals. In other words, the RMSE tells how concentrated the data are around the line of the best fit between observed and predicted values. RMSE values are non-negatives; a value of 0 indicates a perfect fit between observed and predicted values. The RMSE value depends on the scale of the data and is therefore not suitable for comparison between datasets. In general, the lower the RMSE, the better the fit between observed and predicted values. The RMSE is computed as follows:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\text{obs}_i - \text{pred}_i)^2}, \tag{9.1}$$

where n is the validation sample size and obs and pred are vectors of observed and predicted values of the soil properties, respectively. The RMSE can simply be derived in R by:

```
RMSE <- function(obs, pred){
  sqrt(mean((pred - obs)^2, na.rm = TRUE))
}
```

Mean error (ME) or bias The mean error (ME) is often computed along with the RMSE to assess the bias of the predictions. The ME is simply the average of all the errors between predictions and observations. Ideally, the value of the ME is zero which indicates no bias in the prediction. Note that a ME of zero does not indicate that there is no error (the positive and negative errors cancel out), but that there is no systematic bias in the predictions made by the model. The ME is computed as follows:

$$\text{ME} = \frac{1}{n}\sum_{i=1}^{n}(\text{obs}_i - \text{pred}_i), \tag{9.2}$$

which in R gives:

```
ME <- function(obs, pred){
  mean(pred - obs, na.rm = TRUE)
}
```

Squared correlation coefficient (r^2) Pearson's squared correlation coefficient (r^2) is commonly used to assess the dispersion around the regression line. In other words, the r^2 represents the strength of the linear association between observed and predicted values with respect to the fitted regression line. The r^2 is such that the values are between 0 and 1. It is computed as follows:

$$r^2 = \left(\frac{\sum_{i=1}^{n}(\text{obs}_i - \overline{\text{obs}})(\text{pred}_i - \overline{\text{pred}})}{\sqrt{\sum_{i=1}^{n}(\text{obs}_i - \overline{\text{obs}})^2}\sqrt{\sum_{i=1}^{n}(\text{pred}_i - \overline{\text{pred}})^2}}\right)^2. \tag{9.3}$$

In R, this can be efficiently derived using the `cor` function from the `stats` package:

```
r2 <- function(obs, pred){
  cor(pred, obs, method = "spearman", use = "pairwise.complete.obs")^2
}
```

While the r^2 is widely used, there is a general confusion in the literature about what a r^2 is and how to compute it. When the r^2 is computed as the squared Pearson's r correlation coefficient, it measures the closeness of fit to the fitted linear regression line between observed and predicted, but does not indicate the closeness against a 1:1 line (observed versus predicted) which is of interest when validating. The r^2 is not sensitive to the departure of fitted regression line to the 45 degree line of agreement. In many cases, it is therefore not recommended to compute the r^2 as the squared correlation coefficient, in particular when predictions are biased.

Coefficient of determination (R2) The coefficient of determination (R2) is the amount of variance explained by the model. The R2 quantifies the improvement made by the model over simply using the mean of the observations as prediction (Janssen and Heuberger 1995). In the literature, the R2 is sometimes referred to as the amount of variance explained, a modelling efficiency coefficient (Wadoux et al. 2018), a skill score (Nussbaum et al. 2017) or a Nash-Sutcliffe model efficiency coefficient (Nash and Sutcliffe 1970). As for the r^2, its optimal value is 1, but it can be negative if the root mean square error exceeds the standard deviation of the data. It is computed as follows:

$$\mathrm{R2} = 1 - \frac{\sum_{i=1}^{n}(\mathrm{obs}_i - \mathrm{pred}_i)^2}{\sum_{i=1}^{n}(\mathrm{obs}_i - \overline{\mathrm{obs}})^2} \tag{9.4}$$

which is equal to $1 - SSE/SST$ where SSE is the sum of the squared error and SST of the total sum of squares. The R2 is derived in R by:

```
R2 <- function(obs, pred){
  # sum of the squared error
  SSE <- sum((pred - obs) ^ 2, na.rm = T)
  # total sum of squares
  SST <- sum((obs - mean(obs, na.rm = T)) ^ 2, na.rm = T)
  R2 <- 1 - SSE/SST
  return(R2)
}
```

Lin's concordance coefficient (ρ_c) The concordance correlation coefficient (ρ_c) was introduced by Lawrence and Lin (1989) to assess the agreement between observed and predicted values with respect to the 1:1 line. If the predictions are in perfect agreement with the observations, all the points fall on the 1:1 line. The ρ_c is given by:

$$\rho_c = \frac{2r\sigma_{\mathrm{pred}}\sigma_{\mathrm{obs}}}{\sigma_{\mathrm{obs}}^2 + \sigma_{\mathrm{pred}}^2 + (\mu_{\mathrm{obs}} - \mu_{\mathrm{pred}})^2} = rC_b, \tag{9.5}$$

where r is Pearson's correlation coefficient, σ is the standard deviation ($r\sigma_{\mathrm{pred}}\sigma_{\mathrm{obs}}$ is the covariance between observed and predicted values) and μ is the mean. Lawrence and Lin (1989) have shown that ρ_c reduces to rC_b where C_b is the bias correction factor defined as:

$$C_b = \left(\frac{v + 1/v + u^2}{2}\right)^{-1}, \tag{9.6}$$

with $v = \sigma_{\mathrm{pred}}/\sigma_{\mathrm{obs}}$ being the scale shift and $u = (\mu_{\mathrm{pred}} - \mu_{\mathrm{obs}})/\sqrt{\sigma_{\mathrm{pred}}\sigma_{\mathrm{obs}}}$ being the location shift relative to the scale. In other terms, ρ_c assesses the correlation

between observed and predicted values, with a bias correction. The ρ_c can be implemented in R by:

```
rhoC <- function(obs, pred) {
  n <- length(pred)
  sdPred <- sd(pred, na.rm = T)
  sdObs <- sd(obs, na.rm = T)
  r <- stats::cor(pred, obs, method = "pearson", use = "pairwise.complete.obs")
  # scale shift
  v <- sdPred / sdObs
  sPred2 <- var(pred, na.rm = T) * (n - 1) / n
  sObs2 <- var(obs, na.rm = T) * (n - 1) / n
  # location shift relative to scale
  u <- (mean(pred, na.rm = T) - mean(obs, na.rm = T)) / ((sPred2 * sObs2)^0.25)
  Cb <- ((v + 1 / v + u^2)/2)^-1
  rCb <- r * Cb
  return(rCb)
}
```

There are several implementations for computing ρ_c with associated confidence intervals and p-value, and the reader can find examples in the `DescTools` package with the `CCC` function or in the `epiR` package with the `epi.ccc` function.

Ratio of performance to deviation (RPD) The ratio of performance to deviation (RPD) was proposed by Williams and Thompson (1978) as the ratio of standard error in prediction to the standard deviation. The objective of the RPD is to scale the error in prediction with the standard deviation of the property. It is widely used in the infrared spectroscopy literature as a way to assess the goodness of fit of infrared spectroscopy models.

The RPD is calculated as follows:

$$\text{RPD} = \frac{\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(\text{obs}_i - \overline{\text{obs}})^2}}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\text{obs}_i - \text{pred}_i)^2}}, \tag{9.7}$$

which is equivalent to `sd(obs)/RMSE(obs, pred)`. This metric and its systematic use have been criticized, in particular because the standard deviation of the soil property used to scale the error is misleading in the case of skewed or non-normal observations. The RPD is computed in R by:

```
RPD <- function(obs, pred){
  sdObs <- sd(obs)
  RMSE <- sqrt(mean((pred - obs)^2))
  rpd <- sdObs/RMSE
  return(rpd)
}
```

It can be seen that RPD is proportionally related to the coefficient of determination (or R2). R2 is based on variance, while RPD is based on standard error. If we assume a normal distribution, then `RPD = 1/sqrt(1-R2)`. We can test it (Fig. 9.1).

```
# create a sequence of number from 0 to 1
R2val <- seq(0, 1, by = 0.02)
RPDval = 1/sqrt(1 - R2val)

# plot the R2 and RPD
plot(R2val, RPDval, type = "l")
```

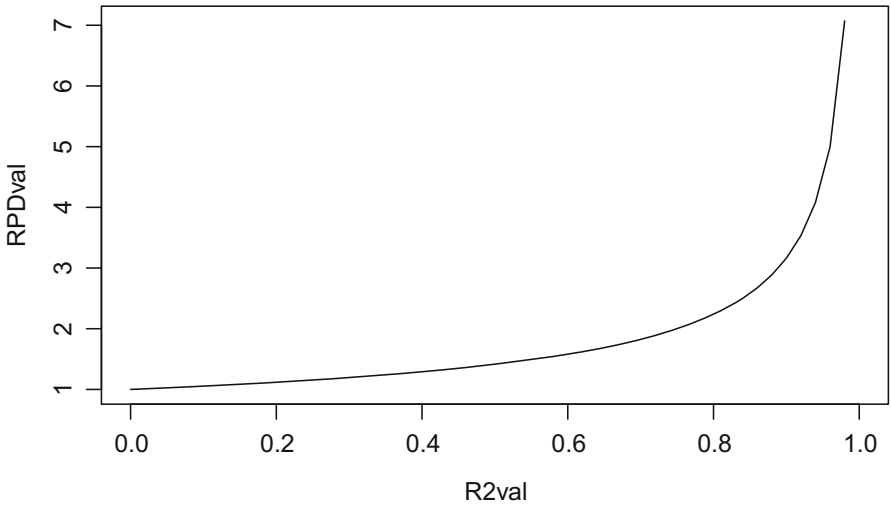

Fig. 9.1 Values of the R2 (x-axis) against those of the RPD (y-axis) computed on the validation dataset

Note that when R2 > 0.8, the RPD rapidly increases for larger R2 values. This suggests that a small increase in accuracy could be interpreted as a large boost in RPD.

Ratio of performance to inter-quartile distance (RPIQ) The ratio of performance to inter-quartile distance (RPIQ) has been proposed by Bellon-Maurel et al. (2010) to account for possibly non-normal distribution of the observations. The RPIQ is similar to the RPD, but it uses the inter-quartile range to represent the spread of the observations. It is therefore not sensitive to the statistical distribution of the observations. The values obtained by the RPIQ can be interpreted the same way than the RPD. The RPIQ is given by:

$$\text{RPIQ} = \frac{(Q_3(\text{obs}) - Q_1(\text{obs}))}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\text{obs}_i - \text{pred}_i)^2}}, \tag{9.8}$$

where $Q_1(\text{obs})$ and $Q_3(\text{obs})$ are the first (25%) third (75%) quantiles of the observations ($Q_3(\text{obs}) - Q_1(\text{obs})$ is the inter-quartile distance) and the denominator is the RMSE. In R, it can be computed as follows:

```
RPIQ <- function(obs, pred){
  q25 <- as.numeric(quantile(obs)[2])
  q75 <- as.numeric(quantile(obs)[4])
  iqDist <- q75 - q25
  RMSE <- sqrt(mean((pred - obs)^2))
  rpiq <- iqDist/RMSE
  return(rpiq)
}
```

From the literature, it appears that various arbitrary limits of RPD were set to characterize a good model performance. For example, in agricultural products, it was quoted by Batten (1998) that RPD values greater than 3 are useful for screening, values greater than 5 can be used for quality control and values greater than 8 can be used for any application. In soil science, the paper by Chang et al. (2001) made other three categories: Category A, RDP > 2.0; Category B, RDP 1.4–2.0; and Category C, RDP < 1.4. This was interpreted by other authors as the reference standard in model performance: excellent if RPD > 2 and non-reliable models when RPD < 1.4. Other authors also have slightly modified this to justify the quality of prediction.

A soil scientist would say their model is excellent as it has RPD > 2, while a plant scientist would disagree as an excellent model should have RPD > 3. Limitations of RPD are described in Minasny and McBratney (2013), and the myth of RPD as a single measure of accuracy is summarized in the article of Esbensen et al. (2014): 'It is a myth that the RPD statistics furthers an objective, across-model, comparative, unambiguous prediction validation figure-of-merit'.

We warn against using an arbitrary classification system to justify the model performance, as RPD and R2 and other measures can be easily affected by the distribution of the data. We can illustrate how these accuracy measures are sensitive to the data distribution. Consider a set of random numbers (Fig. 9.2):

```
# set the seed for repeatability
set.seed(1)

# generate 100 numbers from an uniform distribution between 0 and 1
x = runif(100)
y = runif(100)

# plot the numbers
plot(x, y)

# add a 1:1 line to the plot
abline(a = 0, b = 1)
```

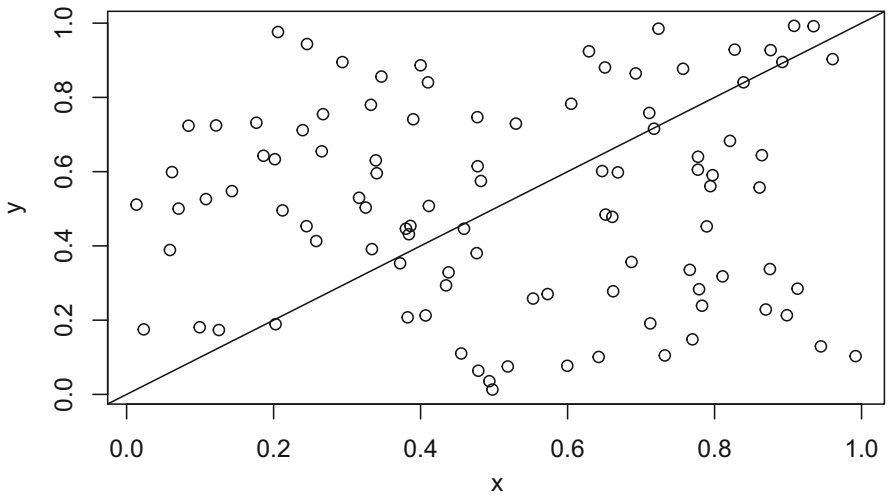

Fig. 9.2 Set of 100 randomly generated values between 0 and 1

Obviously the accuracy measure should say there is no clear relationship between the two random variables. We use the `eval` function from the book-associated `soilspec` package for this.

```
# evaluate the results
soilspec::eval(x, y, obj = "quant")
```

```
##    ME RMSE r2 R2 rhoC  RPD RPIQ
## 1   0 0.38  0 -1 0.02 0.71 1.18
```

Let us now see the effect of adding two extreme observations to the data (Fig. 9.3).

```
# generate two numbers from an uniform distribution between 8 and 10
x1 = runif(2, min = 8, max = 10)
y1 = runif(2, min = 8, max = 10)

# add the two new numbers to the vector of existing numbers
x = c(x, x1)
y = c(y, y1)

# plot the numbers
plot(x, y)

# add a 1:1 line to the plot
abline(a = 0, b = 1)
```

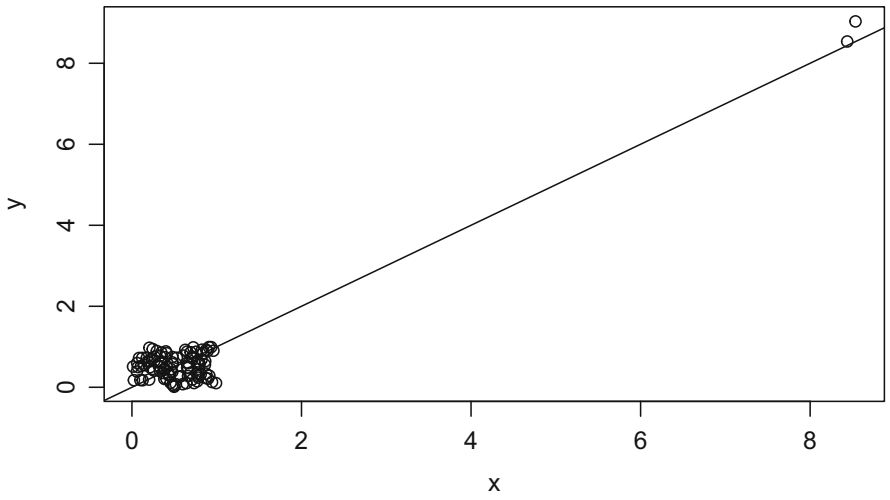

Fig. 9.3 Set of 100 randomly generated values between 0 and 1 and 2 values between 8 and 10

```
# evaluate the results
soilspec::eval(x, y, obj = "quant")
```

```
##      ME RMSE r2   R2 rhoC  RPD RPIQ
## 1 0.01 0.38  0 0.89 0.95 3.04 1.19
```

It is now visible that we can achieve RPD > 3, $\rho_c > 0.9$ and R2 = 0.9 indicating we have created an excellent model. The RPIQ seems not being much affected by outliers.

For categorical variables, the overall accuracy The overall accuracy (OA) measures the fraction of predictions that are correctly classified. Its optimal value is 1 (all if the predicted class equal the observed class) and falls to 0 if none of the predicted classes equal the observed classes. It is formally calculated as follows:

$$\text{OA} = \frac{\text{Number of correct prediction}}{\text{Total number of prediction}}, \tag{9.9}$$

where the numerator is an indicator having 1 if the predicted class equals the observed class and 0 otherwise and the denominator is the validation sample size. In R it is computed as follows:

```
OA <- function(obs, pred){
  # create a confusion matrix between observed and predicted classes
  cm = as.matrix(table(obs = obs, pred = pred))
  n <- length(obs)
  diag = diag(cm)
  OA <- sum(diag) / n
  return(OA)
}
```

As OA only calculates the overall accuracy, if the data is imbalanced. i.e. one class dominates over the others, a predictive model will only predict the dominant class and could provide high accuracy. To solve this problem, Cohen's kappa statistic is used.

For categorical variables, Cohen's kappa statistic Cohen's kappa statistic (Cohen 1960) is another measure of the agreement between predicted and observed classes. It is often referred to as the comparison of the overall accuracy to the expected random chance accuracy. Cohen's kappa is defined as the difference between the overall accuracy and the random chance accuracy divided by 1 minus the random chance accuracy. The statistic can be negative, but is more often comprised between 0 and 1, where 1 shows a perfect agreement between the predicted and observed classes and 0 no more agreement than what is expected by chance. It is derived as follows:

$$\text{Cohen's kappa} = \frac{\text{OA} - p_e}{1 - p_e}, \tag{9.10}$$

where OA is the overall accuracy derived previously and p_e is the expected probability of chance agreement. In other words, p_e is the expected random chance accuracy. In R Cohen's kappa statistic is computed as follows:

```
kappa <- function(obs, pred){
  # create a confusion matrix between observed and predicted classes
  cm = as.matrix(table(obs = obs, pred = pred))
  # number of observations per class
  rowsums = apply(cm, 1, sum)
  # number of predictions per class
  colsums = apply(cm, 2, sum)
  n <- length(obs)
  diag = diag(cm)
  accuracy <- sum(diag) / n
  p = rowsums / n # distribution of points over the actual classes
  q = colsums / n # distribution of points over the predicted classes
  expAccuracy = sum(p*q)

  kappa = (accuracy - expAccuracy) / (1 - expAccuracy)
  return(kappa)
}
```

9.2 Models for Quantitative Variables

A general problem in spectroscopy data is the large number of predictors, i.e. the number of spectral bands. The machine learning literature will describe this as a 'large p and small n' problem. In addition, the spectral bands are highly correlated.

One way of handling data with a high number of predictor variables such as in infrared spectroscopy is variable reduction. Another way is to select only relevant variables to use in the model (variable selection). Principal components and partial least squares regression (PLSR) methods are routinely used in chemometrics for variable reduction. Both models boil down to linear regression and principal component analysis. The PLSR model is particularly useful for prediction purposes. We will first explore these linear models and then continue with machine learning with cubist and random forest models.

In this section, we use the raw spectra described in Chap. 3 (Fig. 9.4). The steps for pre-processing are explained in the previous chapters.

```
# load the required packages
require(prospectr)
require(soilspec)

# load the data
data("datsoilspc")

# convert reflectance to absorbance
spectraA <- log(1/datsoilspc$spc)

# embed the soil property and the spectra in one single table
datsoilspc$spcA <- spectraA

# apply some smoothing to the spectra
oldWavs <- as.numeric(colnames(datsoilspc$spcA))
newWavs <- seq(min(oldWavs), max(oldWavs), by = 5)
datsoilspc$spcARs <- prospectr::resample(datsoilspc$spcA,
                                         wav = oldWavs,
                                         new.wav = newWavs,
                                         interpol = "linear")

# apply a standard normal variate transformation for baseline correction
datsoilspc$spcASnv <- standardNormalVariate(datsoilspc$spcARs)

# apply a moving average window to the standard normal variate spectra
datsoilspc$spcAMovav <- movav(datsoilspc$spcASnv, w = 11)

# convert the column names from integer to numeric
wavs <- as.numeric(colnames(datsoilspc$spcAMovav))

# plot first spectrum
matplot(x = wavs, y = t(datsoilspc$spcAMovav),
        xlab = "Wavelength /nm",
        ylab = "Absorbance",
        type = "l",
        lty = 1,
        col = rgb(red = 0.5, green = 0.5, blue = 0.5, alpha = 0.3))
```

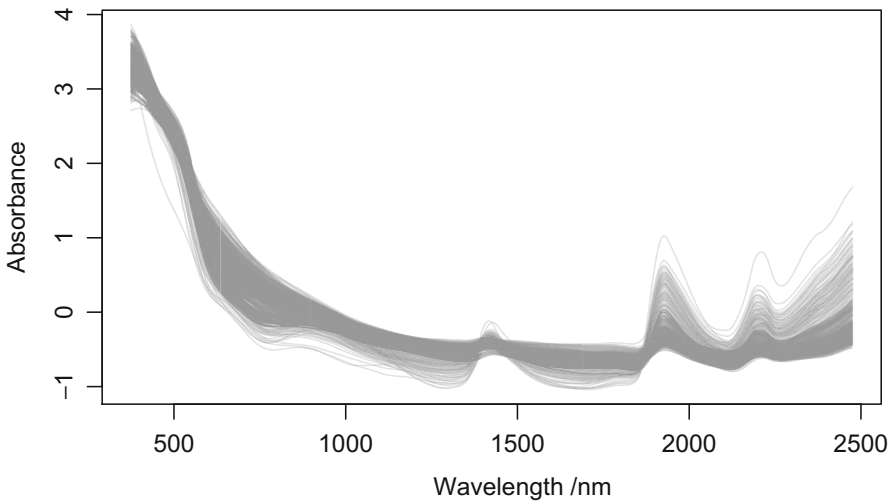

Fig. 9.4 Pre-processed absorbance spectra from the `datsoilspc` dataset provided in the book package `soilspec`

For the example, we further separate the data into calibration (75%) and validation (25%) by splitting randomly the dataset (Fig. 9.5).

```
# set the seed
set.seed(19101991)

# id of the rows to be used for calibration
calId <- sample(1:nrow(datsoilspc), size = round(0.75*nrow(datsoilspc)))

# separate the dataset into calibration and validation
datC <- datsoilspc[calId,]
datV <- datsoilspc[-calId,]

# plot the value of the Total Carbon content for both calibration and validation
par(mfrow=c(1,2))

# calibration
hist(datC$TotalCarbon,
     main = "",
     xlab = "Total carbon")

# validation
hist(datV$TotalCarbon,
     main = "",
     xlab = "Total carbon")
```

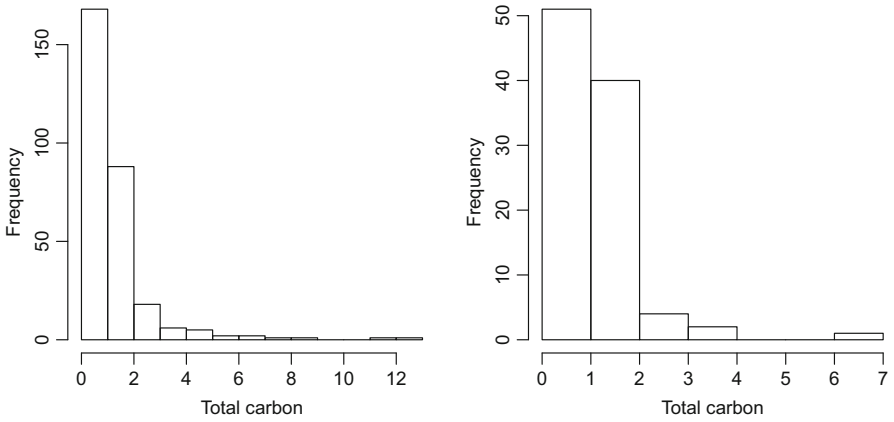

Fig. 9.5 Histograms of the soil total carbon content from the Geeves et al. (1994) dataset provided in the book package `soilspec` for both calibration (left) and validation (right)

The observation of the total carbon content is slightly skewed to the left. In a real-world case study, we might want to test whether a transformation makes the carbon values look normally distributed. The slight skewness is not critical in this case study.

9.2.1 Principal Component Regression

Principal component regression (PCR) boils down to principal component analysis (Sect. 6.2) and multiple linear regression by taking the principal component of the spectra and by building a linear regression on the component scores. Recall the matrix of scores $\mathbf{T}$ obtained by PCA of the matrix of spectral variables $\mathbf{X}$. In PCR, a linear regression model is fitted between the scores of the PC of $\mathbf{X}$ and the soil property (response variable) $\mathbf{y}$ (if only one soil property is predicted) or $\mathbf{Y}$ of size $n \times c$ where c is the number of soil properties of interest ($c = 1$ for a single response). The PCR model takes the following form:

$$\mathbf{Y} = \mathbf{T}\boldsymbol{\beta} + \mathbf{E}, \tag{9.11}$$

where $\boldsymbol{\beta}$ denotes the vector of regression coefficients, found by solving $\boldsymbol{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$, and $\mathbf{E}$ is a matrix of residuals, the same size than $\mathbf{Y}$.

The user must then decide how many components to use in the model. Usually, the optimal number of components is defined by cross-validation. The cross-validation can be done by using the `pls` package, but for illustration we will use the `princomp` and `lm` functions.

```
# first we perform a PCA on the spectra
pcspectra <- prcomp(datC$spcAMovav,
                    center = TRUE, scale = TRUE)

# calculate the percent of the variances explained by the PC
v <- pcspectra$sdev*pcspectra$sdev

# percentage of cumulative variances
cumv <- 100*cumsum(v)/sum(v)

# plot cumulative percentage of variances explained by the PCs
plot(cumv[1:20],
     type = "b",
     xlab = "PC",
     ylab = "% Cumulative variance")
```

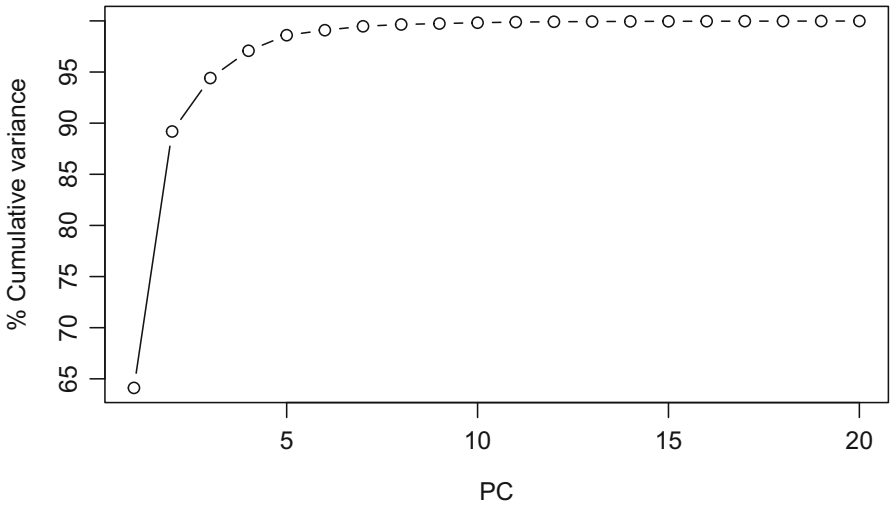

Fig. 9.6 Percentage of variance explained by the principal components against number of components

Figure 9.6 shows that around nine components capture more than 99% of the variation.

One can now fit a simple PC regression. First we specify the number of principal components to use in the model and then form the PC scores as the independent variables and soil property as the dependent variable in a linear model.

```
# specify number of components
npc <- 9

# select PC scores
sdata <- as.data.frame(pcspectra$x[,1:npc])

# fit a linear model Total C = PC1 + PC2 + ...
soilCPcrModel <- lm(datC$TotalCarbon ~ ., data = sdata)
```

```
# obtain a summary of the fit
summary(soilCPcrModel)
```

```
##
## Call:
## lm(formula = datC$TotalCarbon ~ ., data = sdata)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.2051 -0.4448 -0.0744  0.3529  4.5404
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.214198   0.046704  25.998  < 2e-16 ***
## PC1          0.027437   0.002848   9.635  < 2e-16 ***
## PC2          0.059561   0.004554  13.080  < 2e-16 ***
## PC3          0.001337   0.009975   0.134  0.89348
## PC4          0.157447   0.013955  11.282  < 2e-16 ***
## PC5          0.303688   0.018495  16.420  < 2e-16 ***
## PC6          0.014171   0.032577   0.435  0.66389
## PC7          0.115872   0.036931   3.138  0.00188 **
## PC8          0.400682   0.053989   7.422 1.36e-12 ***
## PC9         -0.043479   0.076195  -0.571  0.56871
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7994 on 283 degrees of freedom
## Multiple R-squared:  0.7196, Adjusted R-squared:  0.7107
## F-statistic:  80.7 on 9 and 283 DF,  p-value: < 2.2e-16
```

We can now assess the goodness of the fit by plotting the observed versus predicted values of the total carbon, for both the calibration and validation datasets (Fig. 9.7).

```
# predict on the calibration dataset
soilCPcrPred <- predict(soilCPcrModel, sdata)

# predict on the validation dataset
pcspectraV <- predict(pcspectra, datV$spcAMovav)
sdataNew <- as.data.frame(pcspectraV[, 1:npc])
soilVPcrPred <- predict(soilCPcrModel, sdataNew)

par(mfrow = c(1, 2))

# plot calibration
plot(datC$TotalCarbon, soilCPcrPred,
     xlab = "Observed",
     ylab = "Predicted",
     xlim = c(0, 12),
     ylim = c(0, 12),
     pch = 16)
abline(0, 1)

# plot validation
plot(datV$TotalCarbon, soilVPcrPred,
```

```
     xlab = "Observed",
     ylab = "Predicted",
     xlim = c(0, 12),
     ylim = c(0, 12),
     pch = 16)
abline(0, 1)
```

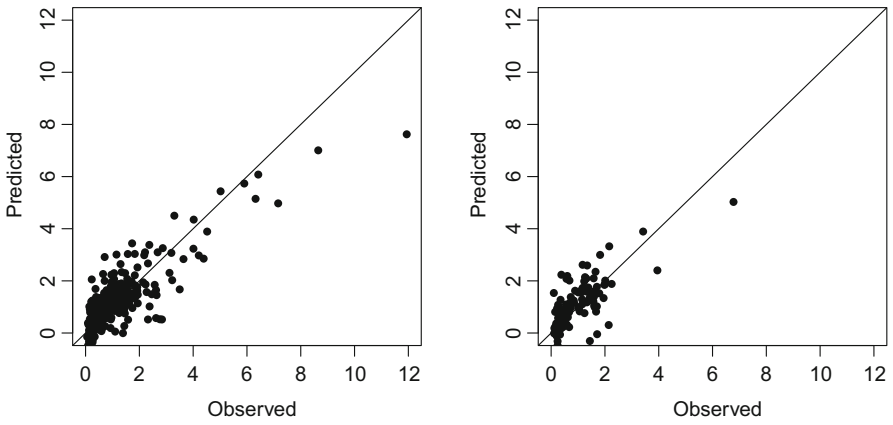

Fig. 9.7 Scatterplot of observed versus predicted value of the total carbon. The predictions are made by principal component regression

Derive some accuracy measures, for the example using the `eval` function and specifying the argument `obj = "quant"` for continuous variables:

```
# accuracy measures for calibration
soilspec::eval(datC$TotalCarbon, soilCPcrPred, obj = "quant")
```

```
##   ME RMSE   r2   R2 rhoC  RPD RPIQ
## 1  0 0.79 0.57 0.72 0.84 1.89 1.46
```

and for validation.

```
# accuracy measures for validation
soilspec::eval(datV$TotalCarbon, soilVPcrPred, obj = "quant")
```

```
##      ME RMSE   r2   R2 rhoC  RPD RPIQ
## 1 0.08 0.74 0.42 0.36  0.7 1.26 1.63
```

9.2.2 Partial Least Squares Regression

Partial least squares regression (PLSR) is a technique that attempts to combine PCA and multiple regression (Wold et al. 2001). It aims to predict a set of dependent

variables (soil properties) by extracting from the spectra a set of 'orthogonal' factors (or latent variables) which give the best prediction. The components in partial least squares are determined by the predictor variables $\mathbf{X}$ (as in PCR) but also by the response variable(s) ($\mathbf{y}$ or $\mathbf{Y}$ if multiple responses). The PLSR model takes the following form:

$$\mathbf{X} = \mathbf{T}\mathbf{P}^T + \mathbf{E}, \tag{9.12}$$

$$\mathbf{Y} = \mathbf{U}\mathbf{Q}^T + \mathbf{F}, \tag{9.13}$$

where $\mathbf{X}$, $\mathbf{T}$ and $\mathbf{P}$ are defined previously. Both $\mathbf{T}$ and $\mathbf{U}$ are score matrices of size $n \times p$ for $\mathbf{X}$ or $\mathbf{Y}$, respectively, and $\mathbf{P}$ (size $d \times b$) and $\mathbf{Q}$ (size $d \times c$) are loading matrices for $\mathbf{X}$ or $\mathbf{Y}$. Finally, $\mathbf{E}$ is the matrix of residuals of size $n \times b$ for $\mathbf{X}$ and $\mathbf{F}$ is the matrix of residuals of size $n \times c$ for $\mathbf{Y}$. A regression between $\mathbf{X}$ and $\mathbf{Y}$ is obtained by:

$$\mathbf{U} = \mathbf{T}\boldsymbol{\beta}, \tag{9.14}$$

where $\boldsymbol{\beta}$ is the vector of regression coefficients of the linear model. Substituting this relationship from the original model, predictions are obtained by:

$$\mathbf{Y} = \mathbf{U}\mathbf{Q}^T = \mathbf{T}\boldsymbol{\beta}\mathbf{Q}^T. \tag{9.15}$$

This is implemented in R with the `plsr` function from the `pls` package, made by Wehrens and Mevik (2007). As for the PCA and PCR, we must choose the optimal number of principal components (Fig. 9.8).

```
# load required package
require(pls)

# maximum number of components in the PLS model
maxc <- 30

# generate a PLS model based on calibration data
soilCPlsModel <- plsr(TotalCarbon ~ spcAMovav,
                      data = datC,
                      method = "oscorespls",
                      ncomp = maxc,
                      validation = "CV")

# this is the plsr function, using cross validation to evaluate the RMSEP
# as a function of number of components from one until maxc
plot(soilCPlsModel, "val",
     main = " ",
     xlab = "Number of components")
```

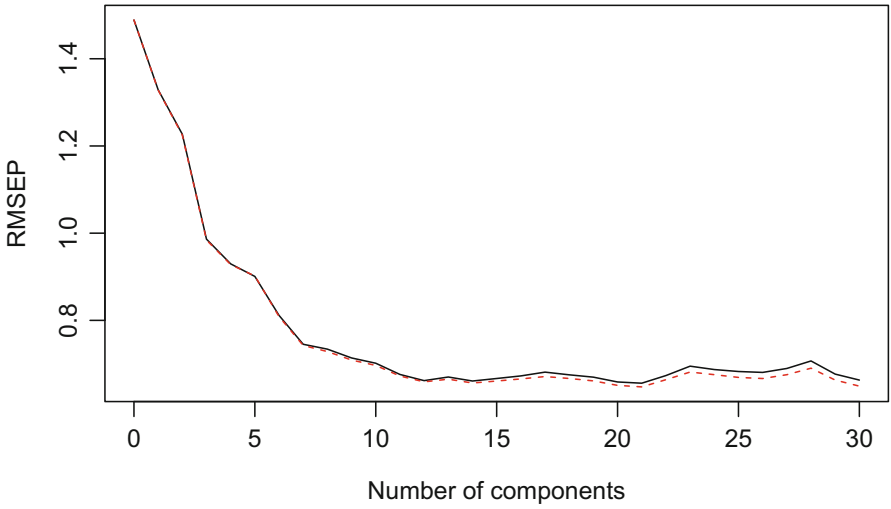

Fig. 9.8 Root mean square error of the prediction (RMSEP, black line) and bias-adjusted RMSEP (red dashed line) obtained by cross-validation against the number of components used in the PLSR model

More information on the RMSEP and bias-adjusted RMSEP values are obtained in the vignette of the `pls` package. Note that we use `method = oscorespls`, which is not the default of the `plsr` function. It is discussed later in this section. This figure from a 10-fold cross-validation on the calibration data shows that 14 components seem to produce a minimal RMSEP. So we use this number.

```
# number of components to use
nc <- 14
```

It is also possible to plot directly the predicted and observed values of the soil properties. The predictions are made using the fitted `pls_model` using `nc` principal components (Fig. 9.9).

```
# plot of cross-validated predictions
plot(soilCPlsModel,
     ncomp = nc,
     main = " ",
     xlab = "Observed",
     ylab = "Predicted")
```

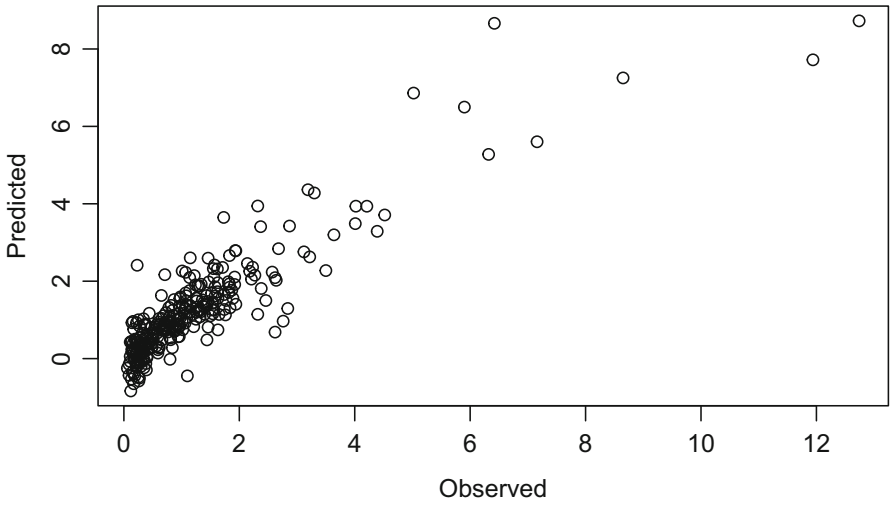

Fig. 9.9 Scatterplot of observed against predicted values of the total carbon. The predictions are made by partial least squares regression

This method plots observed against predicted based on the cross-validated predictions using `nc` number of components. Additional figures can be derived from the `soilCPlsModel` object. Note that the values in the x-axis are the indices of the wavelength (Fig. 9.10).

```
# the three first loadings
plot(soilCPlsModel,
     "loadings",
     comps = 1:3,
     xlab = "Index of the wavelength",
     ylab = "Loading value")
```

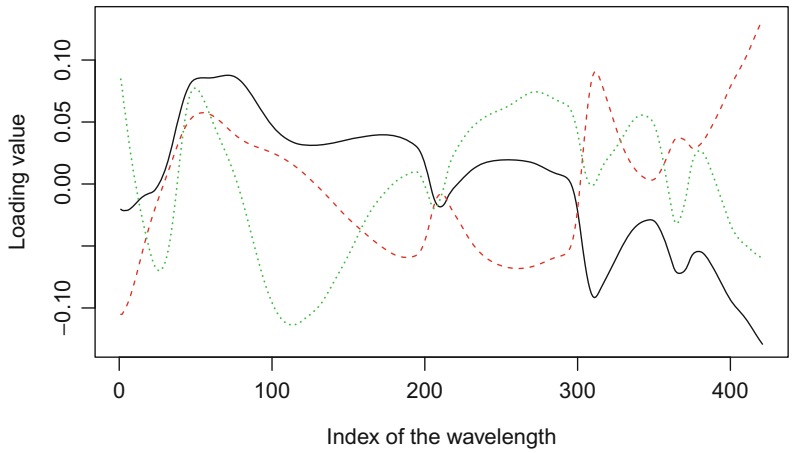

Fig. 9.10 First three loadings of the principal components of the spectra

Figure 9.10 shows the three first loadings of the PLS components of the spectra. It gives an idea which wavelengths have the most influence on each of the components, which can be used for spectral interpretation. In addition to the loadings of the PC, in PLSR the regression coefficients can be plotted. We do not use the default `plot` function of the `plsr` package and select instead manually the regression coefficient for the case of `nc = 14` (Fig. 9.11).

```
# plot the coefficient
plot(wavs, soilCPlsModel$coefficients[,1,nc],
     main = " ",
     type = "l",
     xlab = "Wavelength /nm",
     ylab = "Regression coefficient")
abline(h = 0)
```

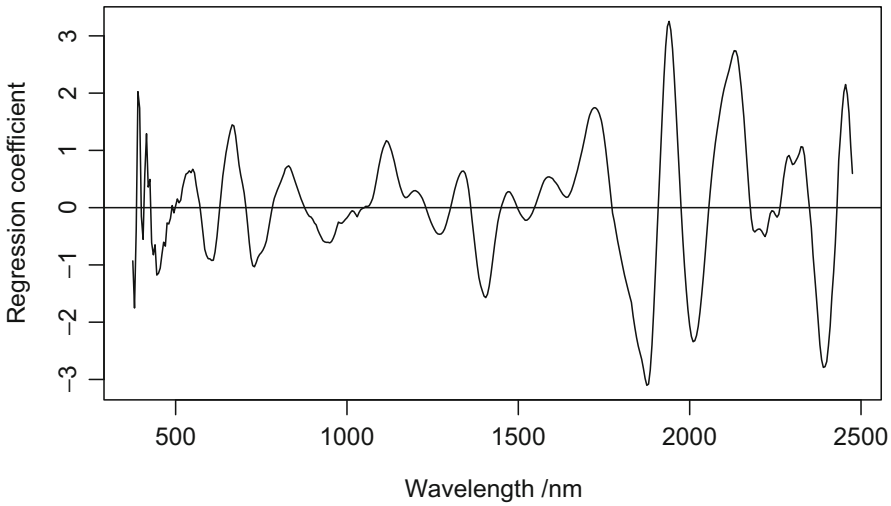

Fig. 9.11 Standardized regression coefficient of the PLSR model for predicting total carbon

This gives a more meaningful interpretation, the standardized coefficients of the regression. Wavelengths with a large (positive or negative) value of the regression coefficient are more influential in the prediction.

Alternative to the regression coefficients and the principal component loadings, one can use the variable importance on projection (VIP) proposed by Wold et al. (1993). It measures the importance of each wavelength in the projection of the components. It is calculated as a weighted sum of the squares of the PLS weights

(Ng et al. 2019). Influential variables have a variable importance on projection value greater than 1. As there is no direct implementation in the `pls` package, we provide the code below. For more information, the reader is redirected to the article of Wold et al. (1993). Note that the VIP calculation is valid only for the orthogonal score algorithm (`method = "oscorespls"` in the `pls` package) and for a single response PLSR model. By default, the `plsr` function in R uses the `kernelpls` algorithm.

```
# take the loadings, loading weights and scores
W <- soilCPlsModel$loading.weights
Q <- soilCPlsModel$Yloadings
TT <- soilCPlsModel$scores

# compute the variable importance, see Wold et al., (1993)
Q2 <- as.numeric(Q) * as.numeric(Q)
Q2TT <- Q2[1:nc] * diag(crossprod(TT))[1:nc]
WW <- W * W/apply(W, 2, function(x) sum(x * x))
vip <- sqrt(length(wavs) * apply(sweep(WW[, 1:nc], 2, Q2TT, "*"),
                                 1, sum)/sum(Q2TT))

# display the variable importance
plot(wavs, vip,
     xlab = "Wavelength /nm",
     ylab = "Importance",
     type = "l",
     lty = 1,
     col = rgb(red = 0.5, green = 0.5, blue = 0.5, alpha = 1))
abline(h = 1)
```

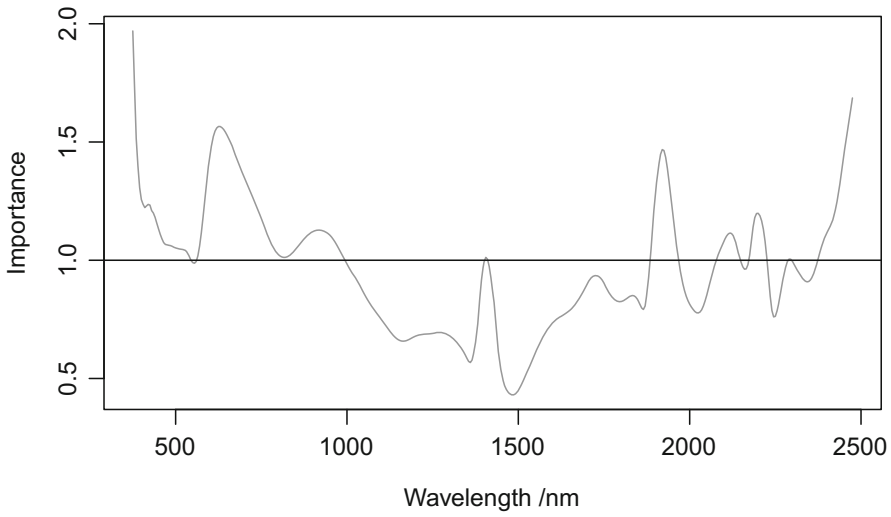

Fig. 9.12 Importance of each wavelength on projection of the principal components

Figure 9.12 shows that the important wavelengths in the projection of the PLS components to predict the total soil carbon are similar to the wavelengths found

important in the previous figures using the standardized regression coefficient or the first three principal component loadings.

Now that we have created the PLS model, we can use the model to predict using the spectra on the calibration and validation datasets (Fig. 9.13).

```
# predict on the calibration dataset
soilCPlsPred <- predict(soilCPlsModel, ncomp = nc, newdata = datC$spcAMovav)

# predict on the validation dataset
soilVplsPred <- predict(soilCPlsModel, ncomp = nc, newdata = datV$spcAMovav)

par(mfrow = c(1, 2))

# plot calibration
plot(datC$TotalCarbon, soilCPlsPred,
     xlab = "Observed",
     ylab = "Predicted",
     xlim = c(0, 12),
     ylim = c(0, 12),
     pch = 16)
abline(0, 1)

# plot validation
plot(datV$TotalCarbon, soilVplsPred,
     xlab = "Observed",
     ylab = "Predicted",
     xlim = c(0, 12),
     ylim = c(0, 12),
     pch = 16)
abline(0, 1)
```

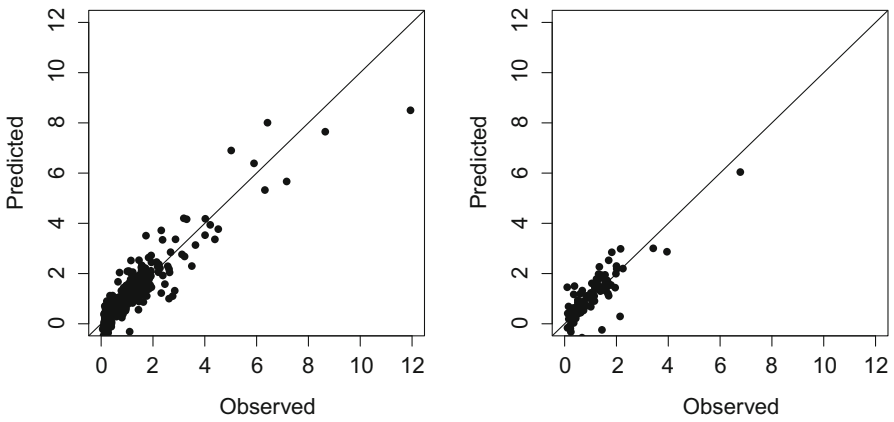

Fig. 9.13 Scatterplot of observed against predicted values of the total carbon. Predictions are made by a PLSR model. The left-hand side plot shows the observed against predicted values for the calibration dataset, while the right-hand side plot shows the observations against predictions for the validation dataset

Derive some accuracy measures, for the example using the `eval` function and specifying the argument `obj = "quant"` for continuous variables:

```
# accuracy measures for calibration
soilspec::eval(datC$TotalCarbon, soilCPlsPred, obj = "quant")
```

```
##   ME RMSE  r2   R2 rhoC  RPD RPIQ
## 1  0 0.56 0.8 0.86 0.92 2.63 2.04
```

and for validation.

```
# accuracy measures for validation
soilspec::eval(datV$TotalCarbon, soilVplsPred, obj = "quant")
```

```
##      ME RMSE   r2   R2 rhoC  RPD RPIQ
## 1 0.02 0.52 0.64 0.69 0.85 1.81 2.34
```

Bagging PLSR

One way of 'strengthening' the PLSR prediction is to generate multiple models and average the prediction (making an ensemble model). Bootstrap aggregating or bagging (Breiman 1996) manipulates the calibration data to generate different models. The bootstrap is a general statistical method used to assess the accuracy of a prediction by sampling the calibration data with replacement. Suppose the calibration data of size n is composed of predictors and response; we randomly generate B datasets based on the calibration data by sampling with replacement. For each of the bootstrap datasets, we fit a PLSR model. The bagging estimate is calculated as the average of all the model predictions.

Therefore, it combines the outputs of many models to produce a powerful 'committee', which is useful when dealing with data with high variation, as each realization will produce a model that fits a particular set of the data which may differ from other realizations. The important element of bagging is that by perturbing the calibration, it can cause significant changes in the predictor. The aggregated predictor averages the prediction over a collection of bootstrap samples, therefore reducing the variance of prediction. The accuracy of the prediction is increased when the prediction method is unstable, i.e. small changes in the calibration data used in bootstrap can result in large changes in the resulting predictor. It was used by McBratney et al. (2006) for soil prediction and quantifying its uncertainty. The improvement over a single PLSR could be small, but it may be more robust against noise in the spectra, and it is also possible to obtain uncertainty intervals of the prediction (Mevik et al. 2004).

A function called `fitBagPlsr` was created for this.

```
fitBagPlsr <- function (soilv, spec, nbag, maxc){
  nc <- maxc
  n <- length(soilv)
  vPls <- vector(nbag, mode = "list")
  calRmse <- matrix(0, nrow = nbag, ncol = 1)
  oobRmse <- matrix(0, nrow = nbag, ncol = 1)

  for (ibag in 1:nbag){
    # take a bootstrap sample with replacement
    s <- sample.int(n, replace = TRUE)
```

```
    #build a plsr model
    vPls[[ibag]] <- plsr(soilv[s] ~ spec[s, ], maxc)

    # compute calibration RMSE
    predV <- predict(vPls[[ibag]], ncomp = nc, newdata = spec[s,])
    err2 <- (soilv[s] - predV)^2
    calRmse[ibag] <- sqrt(mean(err2))

    # compute out-of-bag RMSE
    predC <- predict(vPls[[ibag]], ncomp = nc, newdata = spec[-s,])
    err2 <- (soilv[-s] - predC)^2
    oobRmse[ibag] <- sqrt(mean(err2))
  }

  # average the results
  avCalRmse <- mean(calRmse)
  avOobRmse <- mean(oobRmse)

  # return the results
  list(modelBpls = vPls, oobRmse = avOobRmse, calRmse = avCalRmse)
}
```

Based on our previous single PLS model, we use bagging to generate 50 bootstrapped models.

```
# number of bootstrap
nbag <- 50

# maximum number of components
maxc <- 14

# number of components used in the PLSR model, set to equal to maxc
nc <- maxc

# make the bootstrap
bagPlsr <- fitBagPlsr(datC$TotalCarbon, datC$spcAMovav,
                      nbag,
                      maxc)
```

The output of the function contains `model` which is the bootstrapped PLSR models, `oobRmse` which is the mean out-of-bag RMSE and `calRmse` which is the mean calibration RMSE.

Out of bag is the internal validation used in bootstrap. At each bootstrap, a sample of size n of the original data was sampled with replacement. That means that for each bootstrap about one-third of the data are not used in the calibration. This oob (out-of-bag) data are used to estimate the error of the model.

```
# average RMSE from oob estimates
bagPlsr$oobRmse
```

```
## [1] 0.7093668
```

```
# average RMSE from calibration
bagPlsr$calRmse
```

```
## [1] 0.5187187
```

Now that we have created the bagged PLS model, we can use the model to predict using the spectra on the calibration and validation datasets using the following function.

```
predictBagPlsr <- function(modelBpls, newspec, nbag, nc){

  n <- nrow(newspec)
  predV <- matrix(0, nrow = n, ncol = nbag)

  for (ibag in 1:nbag) {
    predV[,ibag] <- predict(modelBpls[[ibag]], ncomp = nc, newdata = newspec)
  }
  predAve <- apply(predV, 1, mean)
  predStd <- apply(predV, 1, sd)

  return(list(bagPred = predV, predAve = predAve, predStd = predStd))
}
```

In the function `predictBagPlsr`, the argument `modelBpls` is the bagged PLSR model, `newspec` is the spectra of the validation set, `nbag` is the number of bootstrap samples, and `nc` is the number of PCs used in PLSR. The function returns `bagPred` that is the bagged predicted values, `predAve` that is the mean of the predictions and `predStd` that is the standard deviation of the predictions (Fig. 9.14).

```
# predict on the calibration dataset
soilCBagplsPred <- predictBagPlsr(bagPlsr$modelBpls, datC$spcAMovav,
                                  nbag,
                                  nc)

# predict on the validation datset
soilVBagplsPred <- predictBagPlsr(bagPlsr$modelBpls, datV$spcAMovav,
                                  nbag,
                                  nc)

par(mfrow = c(1, 2))

# plot calibration
plot(datC$TotalCarbon, soilCBagplsPred$predAve,
     xlab = "Observed",
     ylab = "Predicted",
     xlim = c(0, 12),
     ylim = c(0, 12),
     pch = 16)
abline(0, 1)

# plot validation
plot(datV$TotalCarbon, soilVBagplsPred$predAve,
     xlab = "Observed",
     ylab = "Predicted",
     xlim = c(0, 12),
     ylim = c(0, 12),
     pch = 16)
abline(0, 1)
```

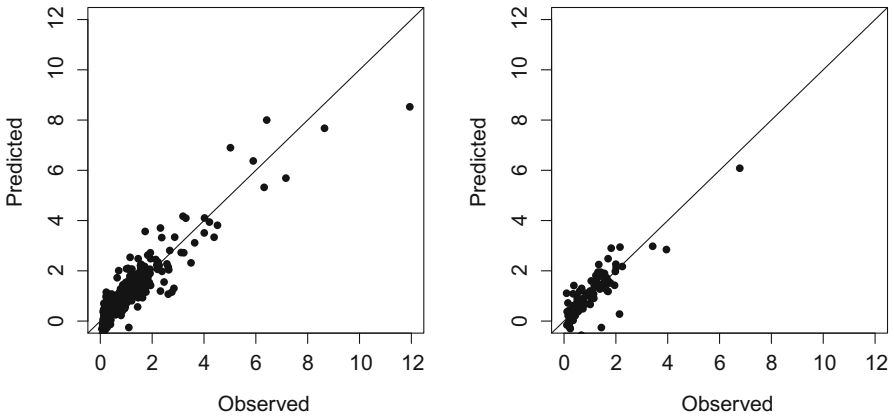

Fig. 9.14 Scatterplot of observed against predicted values of the total carbon. Predictions are made using a bagged partial least squares regression model

Derive some accuracy measures, for the example using the `eval` function and specifying the argument `obj = "quant"` for continuous variables:

```
# accuracy measures for calibration
soilspec::eval(datC$TotalCarbon, soilCBagplsPred$predAve, obj = "quant")
```

```
##   ME RMSE   r2   R2 rhoC  RPD RPIQ
## 1  0 0.56 0.79 0.86 0.92 2.64 2.04
```

and for validation.

```
# accuracy measures for validation
soilspec::eval(datV$TotalCarbon, soilVBagplsPred$predAve, obj = "quant")
```

```
##      ME RMSE   r2  R2 rhoC  RPD RPIQ
## 1 0.01  0.5 0.66 0.7 0.86 1.85 2.39
```

9.2.3 Cubist

Another way of handling large dimensional data is using variable selection techniques to find the best predictors. When the high dimensional data has been reduced to several components or important variables have been selected, they are used for prediction using either linear regression or data-mining tools. Regression trees, neural networks and support vector machines have been used for such predictions.

There are also data-mining tools which are designed to extract information on data containing large number of variables and large number of samples. This is potentially useful as the data reduction step need not be taken. Models that improve regression trees have been proposed, including random forest. While RF has been used successfully for prediction, the model form is complex, and interpretation can be difficult as no explicit formulae can be given.

Other forms of data-mining tools based on the idea of decision trees are the regression rules, rule-based regression or cubist model. This is in effect transforming regression into a classification problem, the model consists of a set of rules, and each rule consists of a linear model. The idea is similar to the regression tree algorithm; while regression trees have a value at each 'leaf', regression rules build a multivariate linear function. Regression rules are also analogous to piecewise linear functions.

The cubist model takes the form of:

```
Rule 1: [10 cases, mean -0.96, range -1.77 to 0.66, est err 0.27]
       if
    R880 <= 0.0144
    R1610 > -0.921
      then
    outcome = -4.06 + 1.27 * R540 + 1.61 * R1610
```

The model has several rules. Each rule has a 'condition' (reflectance at 880 nm <= 0.0144 & at 1610 nm > -0.921); if this condition is met by the data, then the prediction is the given linear function. The program also informs the statistics of each rule which refers to the range of values of the predicted and also the error of the model.

Cubist initially was a commercial regression-rules program, but now a public GNU code has been provided and ported in `Cubist` R package by Kuhn et al. (2012). The model of cubist is a set of comprehensible rules, where each rule has an associated linear model. Whenever a situation matches a rule's conditions, the associated model is used to calculate the predicted value. The first use of cubist for soil spectroscopy modelling was made by Minasny and McBratney (2008).

```
# load required package
require(Cubist)

# make a Cubist model on calibration dataset
soilCCubistModel <- cubist(x = datC$spcAMovav, y = datC$TotalCarbon)

# summary of the model
summary(soilCCubistModel)
```

```
##
## Call:
## cubist.default(x = datC$spcAMovav, y = datC$TotalCarbon)
##
##
## Cubist [Release 2.07 GPL Edition]  Tue Aug 25 15:51:17 2020
## ---------------------------------
##
##     Target attribute 'outcome'
##
## Read 293 cases (422 attributes) from undefined.data
##
## Model:
##
##   Rule 1: [106 cases, mean 0.347, range 0.06 to 1.94, est err 0.120]
##
##     if
##  1415 > -0.4015094
##     then
##  outcome = 4.663 - 447.07 850 + 1223.8 1410 - 1089.7 1400 + 326.8 860
##            - 240.31 2305 + 277.82 845 + 220.88 2310 - 493.2 1415
##            - 162.2 865 + 133.4 2165 + 458.8 1395 - 111.19 2170
##            + 150.9 2075 - 60 810 + 56.88 825 - 119.1 2100 - 36.37 625
```

```
##             + 35.28 630 - 55.7 2015 - 70.4 1815 + 65.8 1805 - 131.6 1455
##             + 32.19 2285 - 94.3 1405 + 22.26 800 + 95.1 1465 + 13.9 1940
##             + 39.6 1365 - 12.23 2380 + 54.4 1435 - 12.34 790 - 47.2 1430
##             + 9.65 2345 - 9.11 2350 - 9 2120 - 6 885 + 2.69 645
##             - 14.6 1445 + 5.7 2145 + 3.3 910
## etc... [shortened]
```

The summary of the model provides the full model. It also informs the rules in the model and the number of times (frequency) certain variables (wavelengths) are used as conditions and as predictors. We can plot these as an indicator of which wavelengths are useful in the model, which we will describe later.

The calibrated cubist model is then used to predict, on both the calibration and validation data (Fig. 9.15).

```
# predict on the calibration data
soilCCubistPredict <- predict(soilCCubistModel, datC$spcAMovav)

# predict on the calibration data
soilVCubistPredict <- predict(soilCCubistModel, datV$spcAMovav)

par(mfrow = c(1, 2))

# plot calibration
plot(datC$TotalCarbon, soilCCubistPredict,
     xlab = "Observed",
     ylab = "Predicted",
     xlim = c(0, 12),
     ylim = c(0, 12),
     pch = 16)
abline(0, 1)

# plot validation
plot(datV$TotalCarbon, soilVCubistPredict,
     xlab = "Observed",
     ylab = "Predicted",
     xlim = c(0, 12),
     ylim = c(0, 12),
     pch = 16)
abline(0, 1)
```

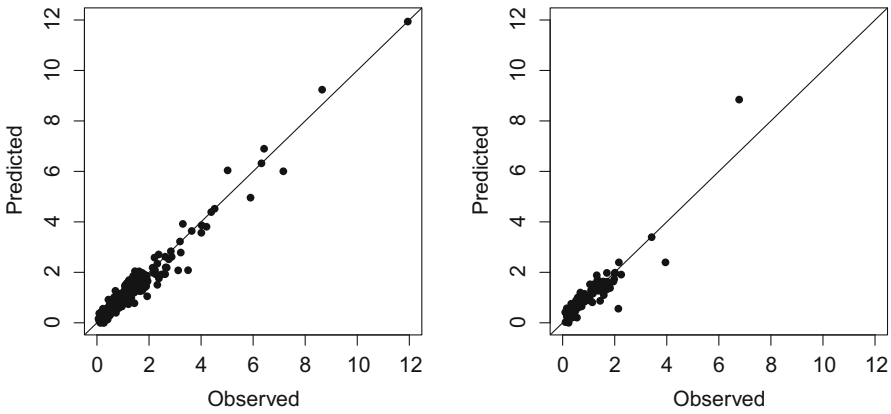

Fig. 9.15 Scatterplot of observed against predicted values of the total carbon. Predictions are made using a cubist model

Derive some accuracy measures, for the example using the `eval` function and specifying the argument `obj = "quant"` for continuous variables:

```
# accuracy measures for calibration
soilspec::eval(datC$TotalCarbon, soilCCubistPredict, obj = "quant")
```

```
##       ME RMSE   r2   R2 rhoC  RPD RPIQ
## 1 -0.02 0.27 0.91 0.97 0.98 5.46 4.23
```

and for validation.

```
# accuracy measure for validation
soilspec::eval(datV$TotalCarbon, soilVCubistPredict, obj = "quant")
```

```
##      ME RMSE  r2   R2 rhoC RPD RPIQ
## 1 0.03 0.39 0.8 0.82 0.92 2.4 3.11
```

Note that the validation statistics show that the calibrated cubist model actually gives us a worse prediction compared to PLS. From the model summary, we can also see that rules 4, 7 and 8 are only fitted to eight, five and seven observations. This could make the model overfit the data. So we need to simplify the model, by setting fewer rules. We re-run the model using two rules by adding the following options: `control = cubistControl(rules = 2)`.

```
# make a Cubist model on calibration dataset with 2 rules
soilCCubistModel2 <- cubist(x = datC$spcAMovav, y = datC$TotalCarbon,
                            control = cubistControl(rules = 2))

# summary of the model
summary(soilCCubistModel)
```

```
##
## Call:
## cubist.default(x = datC$spcAMovav, y = datC$TotalCarbon)
##
##
## Cubist [Release 2.07 GPL Edition]  Tue Aug 25 15:51:17 2020
## -----------------------------------
##
##     Target attribute `outcome'
##
## Read 293 cases (422 attributes) from undefined.data
##
## Model:
##
##   Rule 1: [106 cases, mean 0.347, range 0.06 to 1.94, est err 0.120]
##
##     if
##  1415 > -0.4015094
##     then
##  outcome = 4.663 - 447.07 850 + 1223.8 1410 - 1089.7 1400 + 326.8 860
##              - 240.31 2305 + 277.82 845 + 220.88 2310 - 493.2 1415
##              - 162.2 865 + 133.4 2165 + 458.8 1395 - 111.19 2170
##              + 150.9 2075 - 60 810 + 56.88 825 - 119.1 2100 - 36.37 625
##              + 35.28 630 - 55.7 2015 - 70.4 1815 + 65.8 1805 - 131.6 1455
##              + 32.19 2285 - 94.3 1405 + 22.26 800 + 95.1 1465 + 13.9 1940
##              + 39.6 1365 - 12.23 2380 + 54.4 1435 - 12.34 790 - 47.2 1430
##              + 9.65 2345 - 9.11 2350 - 9 2120 - 6 885 + 2.69 645
##              - 14.6 1445 + 5.7 2145 + 3.3 910
```

```
##
##   Rule 2: [155 cases, mean 1.263, range 0.31 to 3.5, est err 0.280]
## etc... [shortened]
```

The calibrated cubist model is then used to predict, on both the calibration and validation data (Fig. 9.16).

```
# predict on the calibration data
soilCCubist2Predict <- predict(soilCCubistModel2, datC$spcAMovav)

# predict on the calibration data
soilVCubist2Predict <- predict(soilCCubistModel2, datV$spcAMovav)

par(mfrow = c(1, 2))

# plot calibration
plot(datC$TotalCarbon, soilCCubist2Predict,
     xlab = "Observed",
     ylab = "Predicted",
     xlim = c(0, 12),
     ylim = c(0, 12),
     pch = 16)
abline(0, 1)

# plot validation
plot(datV$TotalCarbon, soilVCubist2Predict,
     xlab = "Observed",
     ylab = "Predicted",
     xlim = c(0, 12),
     ylim = c(0, 12),
     pch = 16)
abline(0, 1)
```

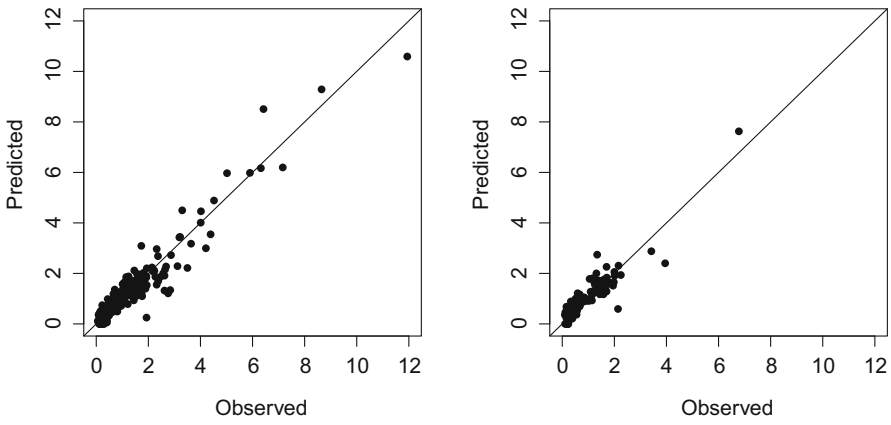

Fig. 9.16 Scatterplot of observed against predicted values of the total carbon. Predictions are made using a cubist model with only two rules

Derive some accuracy measures, for the example using the `eval` function and specifying the argument `obj = "quant"` for continuous variables:

```
# accuracy measures for calibration
soilspec::eval(datC$TotalCarbon, soilCCubist2Predict, obj = "quant")
```

```
##       ME RMSE   r2   R2 rhoC  RPD RPIQ
## 1 -0.01  0.4 0.86 0.93 0.96 3.76 2.91
```

and for validation.

```
# accuracy measures for validation
soilspec::eval(datV$TotalCarbon, soilVCubist2Predict, obj = "quant")
```

```
##      ME RMSE   r2   R2 rhoC  RPD RPIQ
## 1 0.06 0.39 0.79 0.82 0.91 2.38 3.08
```

We can also infer which wavelengths are useful in the model based on the model summary. The following plot shows which variables are important as predictors and as 'conditions' (Fig. 9.17).

```
# plot the variables used as predictors
plot(soilCCubistModel$usage[, 3], soilCCubistModel$usage[, 2],
     type = "h",
     col = "plum",
     xlab = "Wavelength /nm",
     ylab = "Model usage /%")

# plot the conditions in blue
lines(soilCCubistModel$usage[, 3], soilCCubistModel$usage[, 1],
      type = "h",
      col = "blue")  # see which variables are important

# add to the existing plot
par(new = T)

# add a spectra for visualization
plot(colnames(datC$spcAMovav), datC$spcAMovav[1, ],
     axes = F,
     ylim = c(-2, 2),
     xlab = " ", ylab = " ",
     type = "l",
     main = " ",
     xlim = c(500, 2450))
```

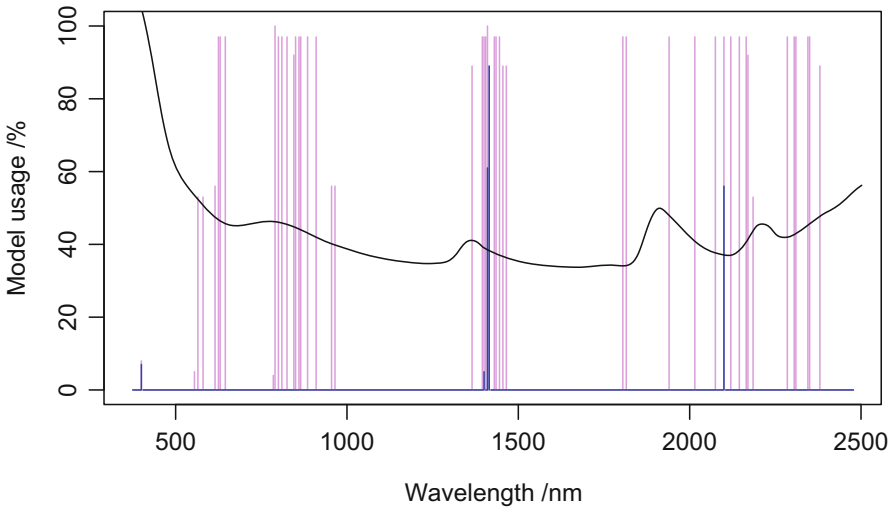

Fig. 9.17 Important variables of the cubist model to predict total carbon. The black line is an example pre-processed absorbance spectrum from the `datsoilspc` dataset, the pink vertical lines are the variables (wavelength) used as predictors in the cubist model, and the vertical blue lines are the cubist model conditions

9.2.4 Random Forest

Random forest (RF) is a widely used algorithm for data science applications in a broad array of scientific domains. A RF model is an ensemble of decision trees organized as set of structured classifiers or trees and can be used for both regression and classification purposes (Breiman 2001). The advantage of RF over its much simpler data-mining counterpart, the CART (Classification and Regression Tree) model, is its ensemble approach. Rather than a single tree in the case of CART, the RF model has many trees, where each is constructed using different perturbations of the data – in terms of both calibration cases and explanatory variables. During RF model development of the ensemble trees, two-thirds of cases are sampled (bootstrap sampling with replacement) and are used to grow a regression tree, and the other one-third is used to perform a cross-validation in parallel with the calibration step. These samples are called out-of-bag samples (oob samples) which are used to obtain an estimate of the model performance. For regression the final prediction is the average of the individual tree or classifier outputs, whereas in classification the trees vote by majority on the correct classification (mode). You might note the similarity of this cross-validation procedure in the earlier described section about the bootstrap PLSR model.

Random forest has three tuning parameters: mtry, number of trees and minimum node size. The first parameter mtry is the number of input variables that are randomly selected for each bootstrap, which can range from 1 to n (sample size).

The second parameter is the number of trees, which must be sufficiently large for the oob error stabilization. In general, 500 trees are sufficient, but if a large number of trees are chosen, the results will not differ significantly, but more time will be necessary to model fitting. The last tuning parameter is the minimum node size, which determines the minimum size of nodes which no split will be attempted; the default value is 5 for regression and 1 for classification.

The RF model is a natural candidate for soil spectral inference work because of the high dimension status of the soil spectral wavelengths, for example, with vis-NIR or MIR data. Ideally it would be used in situations where the number of spectra is also large, as this model has a tendency to overfit, particularly if the tunable parameters previously mentioned are not optimized. The issue here is about generalization when the model is extended to new data, which is not a strong feature of the RF model. Naturally its wide use in data science has also resulted in some researchers experimenting with this model approach for soil spectral data with good results, e.g. Santana et al. (2018) and Hobley et al. (2017) as some relatively recent examples.

In this example, we use the `randomForest` package.

```
# load the required package
require(randomForest)

# prepare the data, the column name cannot be numeric, add 'spec.' in front
datCSub <- data.frame(TotalCarbon = datC$TotalCarbon, datC$spcAMovav)
colnames(datCSub) <- c("TotalCarbon", paste0("spec.", colnames(datC$spcAMovav)))


# run random forest algorithm
soilCRFModel <- randomForest(TotalCarbon ~ .,
                             data = datCSub,
                             ntree = 1000,
                             mtry = 10,
                             importance = TRUE,
                             na.action = na.omit)

# summary of the model
soilCRFModel
```

```
##
## Call:
##  randomForest(formula = TotalCarbon ~ ., data = datCSub, ntree = 1000,
                                        mtry = 10, importance = TRUE, na.action = na.omit)
##                Type of random forest: regression
##                      Number of trees: 1000
## No. of variables tried at each split: 10
##
##           Mean of squared residuals: 0.4582793
##                     % Var explained: 79.18
```

The `randomForest` function saves the important variables that were used to explain the variance of the soil property. For this, we specified `importance = TRUE`, and we can now plot the important variables of the `soilCRFModel` model. Two methods are proposed, the change in terms of MSE or in terms of node purity. The reader is redirected to statistical learning book (e.g. Friedman et al. 2001) for more information on these methods. Figure 9.18 shows the most important variables sorted in order of importance.

```
# plot the most important bands of the spectra
varImpPlot(soilCRFModel,
           main = " ")
```

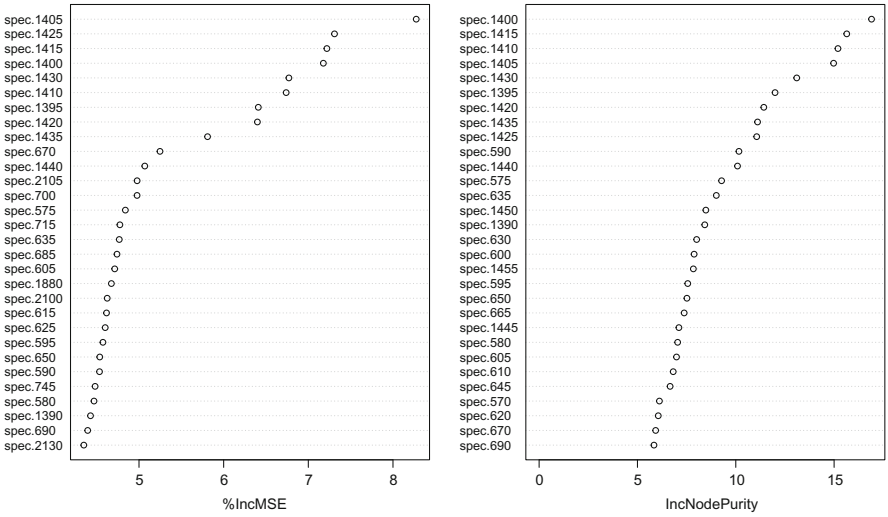

Fig. 9.18 Important variables used by the random forest model to explain the variability of the total carbon. The two methods are the mean decrease in accuracy (left) and mean decrease in node impurity (right). The first variables (bands) are the most important ones

Using the fitted RF model, we can generate predictions on the validation set and validate the predictions (Fig. 9.19).

```
# predict on the calibration data
soilCRFPred <- predict(soilCRFModel, datCSub)

# prepare the validation data
datVSub <- data.frame(TotalCarbon = datV$TotalCarbon, datV$spcAMovav)
colnames(datVSub) <- c("TotalCarbon", paste0("spec.", colnames(datV$spcAMovav)))

# predict on the validation data
soilVRFPred <- predict(soilCRFModel, datVSub)

par(mfrow = c(1, 2))

# plot calibration
plot(datC$TotalCarbon, soilCRFPred,
     xlab = "Observed",
     ylab = "Predicted",
     xlim = c(0, 12),
     ylim = c(0, 12),
     pch = 16)
abline(0, 1)

# plot validation
plot(datV$TotalCarbon, soilVRFPred,
```

```
     xlab = "Observed",
     ylab = "Predicted",
     xlim = c(0, 12),
     ylim = c(0, 12),
     pch = 16)
abline(0, 1)
```

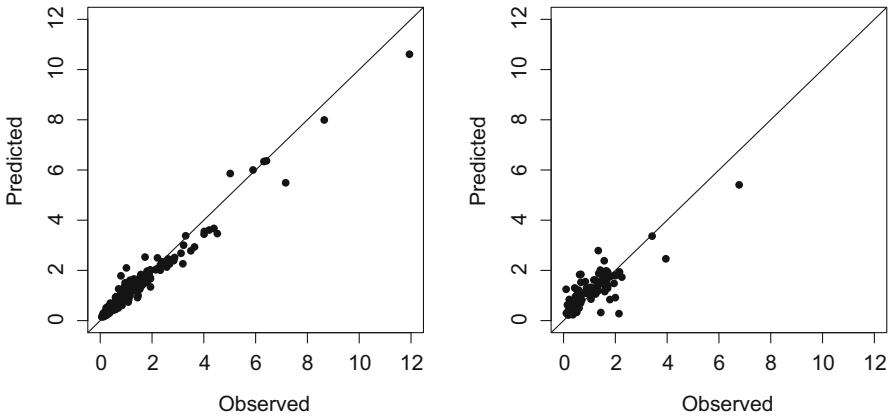

Fig. 9.19 Scatterplot of observed against predicted values of the total carbon. Predictions are made using a random forest model

Derive some accuracy measures, for the example using the `eval` function and specifying the argument `obj = "quant"` for continuous variables:

```
# accuracy measures for calibration
soilspec::eval(datC$TotalCarbon, soilCRFPred, obj = "quant")
```

```
##    ME RMSE   r2   R2 rhoC  RPD RPIQ
## 1   0  0.3 0.94 0.96 0.98 5.03 3.89
```

and for validation.

```
# accuracy measures for validation
soilspec::eval(datV$TotalCarbon, soilVRFPred, obj = "quant")
```

```
##     ME RMSE   r2   R2 rhoC RPD RPIQ
## 1 0.1 0.52 0.57 0.69 0.81 1.8 2.33
```

9.2.5 Memory-Based Learning

Memory-based learning (MBL) is a local calibration algorithm presented in Ramirez-Lopez et al. (2013a) and implemented in the package `resemble`. MBL

has been developed to deal with complex, often continental or global, soil (infrared) spectral datasets. Instead of building a single (global) function to link the soil property and the spectra (as in PLSR or RF), the spectra are split into a number of subsets sharing similar spectral characteristics. In this sense, this technique is closely related to Chap. 7 because it makes use of distance metrics in the principal component space to select the closest points for building a local model. Subsequently, MBL can deal with complex non-linear relationships between the spectra and a particular soil property.
MBL works in the following stages:

1. Build a p-dimensional space of the spectra where p is the number of principal components.
2. For each point of the validation dataset, select its k-nearest neighbours from the calibration sample.
3. Fit a local model for each point of the validation dataset using its nearest neighbours spectra found in the calibration dataset. Several models are available for the fit.

Some aspects are therefore particularly important in any MBL algorithm. One must choose the similarity/dissimilarity metric used to select the closest point in the spectra space (see also Chap. 7). A second important point is the number of neighbours that one wants to use in the fitting process. This number must be sufficiently large to build a realistic model, but increasing too much the number of points might also decrease prediction accuracy. The package `resemble` provides options to optimize the number of neighbours.

In the function `mbl`, the user must then decide:

- the similarity metric `diss_method`, in this example the Mahalanobis distance in the PC space (`diss_method = "pca"`).
- the choice of the optimal number of PCs under the argument `pc_selection`; in this example, the PCs are selected based on the cumulative amount of variance explained.
- the model for fitting, in our example the weighted average PLS model (`method = "local_fit_wapls(min_pls_c = 4, max_pls_c = 17)"`).
- the validation method used, in this example the leave-nearest-neighbour-out cross-validation (`validation_type = "NNv"` in the `mbl_control` argument).

We further decide a sequence of nearest neighbours `k` to test (in order to find the optimal number of neighbours in local model fitting). Here we test a sequence from 20 up to 120 in steps of 10.

```
# define the sequence of neighbours
k2t <- seq(from = 20, to = 120, by = 10)
```

We can now perform the model fitting using MBL and the `mbl` function. We use the object `control` which contains the parameters of the `mbl` function. We make use of a regression function called weighted average partial least square (`wapls1`). It uses multiple models generated by multiple PLS components (i.e. between a minimum and a maximum number of PLS components). At each local partition, the final predicted value is a weighted average of all the predicted values generated by the multiple PLS models. See the package documentation for more details.

```
# load the required package
require(resemble)

# maximum cumulative variance explained to be retained by the PCs
maxexplvar <- 0.99

# run the mbl algorithm
mblResults1 <- mbl(Xr = datC$spcAMovav,
                   Yr = datC$TotalCarbon,
                   # we assume we do not know the total carbon content of the testing
                     dataset
                   Yu = NULL,
                   Xu = datV$spcAMovav,
                   diss_method = "pca",
                   control = mbl_control(validation_type = "NNv"),
                   diss_usage = "none",
                   k = k2t,
                   # define the number of minimum and maximum components for "wapls1"
                   method = local_fit_wapls(min_pls_c = 4, max_pls_c = 17),
                   pc_selection = list("cumvar", maxexplvar),
                   scale = FALSE, center = TRUE)
```

We can summarize the information derived and plot the results.

```
# print a summary of the model
mblResults1
```

```
##
## Call:
##
## mbl(Xr = datC$spcAMovav, Yr = datC$TotalCarbon, Xu = datV$spcAMovav,
##     Yu = NULL, k = k2t, method = local_fit_wapls(min_pls_c = 4,
##         max_pls_c = 17), diss_method = "pca", diss_usage = "none",
##     pc_selection = list("cumvar", maxexplvar), control = mbl_control(validation_
                                                                  type = "NNv"),
##     center = TRUE, scale = FALSE)
##
## ---------------------------------------------------------
##
##  Total number of observations predicted: 98
## ---------------------------------------------------------
##
##  Nearest neighbor validation statistics
##
##       k  rmse st_rmse    r2
##  1:  20 0.530  0.0833 0.784
##  2:  30 0.444  0.0699 0.843
##  3:  40 0.523  0.0823 0.846
##  4:  50 0.542  0.0853 0.850
##  5:  60 0.527  0.0829 0.848
##  6:  70 0.556  0.0874 0.829
##  7:  80 0.549  0.0863 0.828
##  8:  90 0.568  0.0894 0.834
##  9: 100 0.549  0.0863 0.850
```

```
## 10: 110 0.550  0.0864 0.848
## 11: 120 0.535  0.0841 0.851
## ____________________________________________________________
```

```
# plot the RMSE against number of neighbours
matplot(mblResults1$validation_results$nearest_neighbor_validation$k,
        mblResults1$validation_results$nearest_neighbor_validation$rmse,
        type = "b",
        xlab = "K-neighbours",
        ylab = "RMSE",
        pch = 1,
        col = "dodgerblue")
```

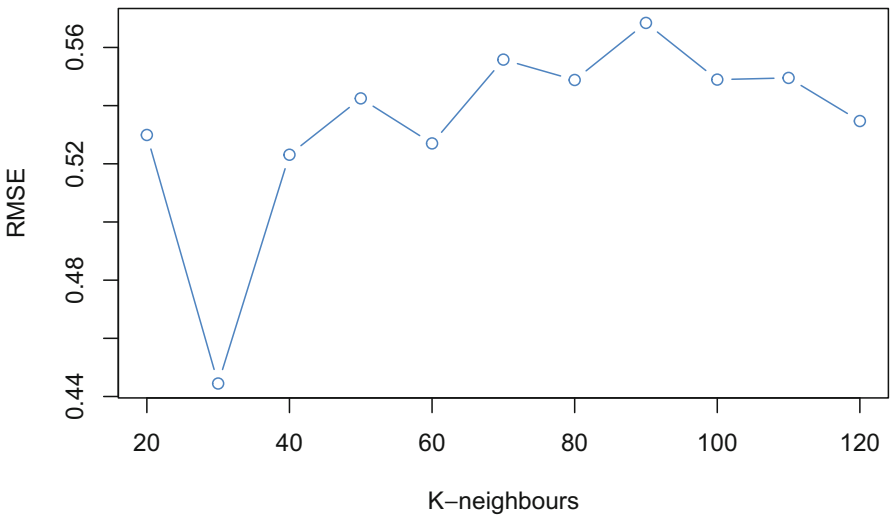

Fig. 9.20 Values of the root mean square error against number of neighbours in the memory-based learning algorithm

Figure 9.20 shows the number of neighbours against the RMSE. The optimal number of neighbours that minimizes the RMSE is 30.

```
# rmse values
rmseMBL <- mblResults1$validation_results$nearest_neighbor_validation$rmse

# minimum rmse value
minRmseMBL <- min(mblResults1$validation_results$nearest_neighbor_validation$rmse)

# number of neighbours values
neighNumber <- mblResults1$validation_results$nearest_neighbor_validation

# select the optimal number of neighbours
optNn <- neighNumber[rmseMBL == minRmseMBL,]$k
```

Instead of assuming that the total carbon content in the validation set is unknown (the `NULL` in the `Yu` argument), we can use the actual total carbon content values

of this dataset. Note that they are not used at all during computations and they are only used for validation purposes (i.e. comparing what it was predicted to the actual values).

```
# run the mbl algorithm specifying the Yu argument
mblResults1Val <- mbl(Xr = datC$spcAMovav,
                      Yr = datC$TotalCarbon,
                      Yu = datV$TotalCarbon,
                      Xu = datV$spcAMovav,
                      diss_method = "pca",
                      control = mbl_control(validation_type = "NNv"),
                      diss_usage = "none",
                      k = optNn,
                      # define the number of minimum and maximum components for "wapls1"
                      method = local_fit_wapls(min_pls_c = 4, max_pls_c = 17),
                      pc_selection = list("cumvar", maxexplvar),
                      scale = FALSE, center = TRUE)
```

We can plot the predicted and observed values of the total carbon as done before for the other calibration algorithms (Fig. 9.21).

```
# plot validation
plot(datV$TotalCarbon, mblResults1Val$results$k_30$pred,
     xlab = "Observed",
     ylab = "Predicted",
     xlim = c(0, 12),
     ylim = c(0, 12),
     pch = 16)
abline(0, 1)
```

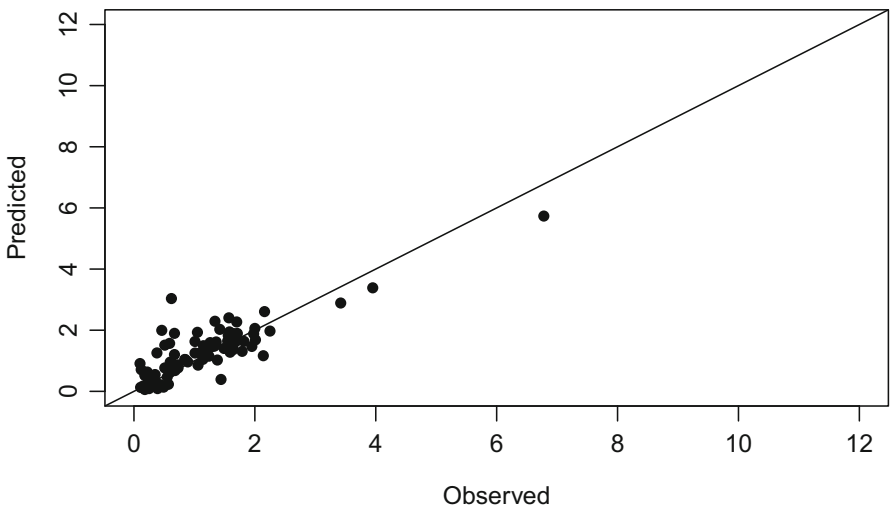

Fig. 9.21 Scatterplot of observed against predicted values of the total carbon. The predictions are made by MBL using a weighted local PLS model

Derive some accuracy measures, for the example using the `eval` function and specifying the argument `obj = "quant"` for continuous variables:

```
# accuracy measures for validation
soilspec::eval(datV$TotalCarbon, mblResults1Val$results$k_30$pred, obj = "quant")
```

```
##      ME RMSE   r2   R2 rhoC  RPD RPIQ
## 1 0.13  0.5 0.65 0.71 0.85 1.86 2.42
```

9.3 Models for Categorical Variables

Soil properties such as organic carbon or clay are continuous. Some other soil properties or attributes are categorical, that is, they are assigned to a finite number of groups. Discrete values can also be considered as categorical if their number is limited. Examples of categorical soil properties are soil texture classes, mineral categories or soil types.

For prediction of categorical soil properties, the models need to be adapted. Since predicting categorical properties with spectroscopy is relatively rare in soil science, we will present two models, which are adaptations of models already presented in the previous section on modelling continuous properties.

As an example, we use the continuous values of clay, silt and sand provided in the book-associated Geeves dataset and convert it to soil texture classes. The dataset originates from Australia (see also Chap. 3), hence the use of the Australian soil texture triangle (Northcote 1971; National Committee on Soil and Terrain (Australia) and CSIRO Publishing 2009) to define the class names. The Australian soil texture triangle has 11 classes. We show below how to convert the soil clay, silt and sand content to soil texture classes using the `soiltexture` R package.

We start by plotting the Australian soil texture triangle and the points from our dataset (Fig. 9.22).

```
# load the require package
require(soiltexture)

# we create the tables
datCsub <- data.frame(CLAY = datC$clay,
                      SILT = datC$silt,
                      SAND = datC$sand)
datVsub <- data.frame(CLAY = datV$clay,
                      SILT = datV$silt,
                      SAND = datV$sand)

# we normalize the data so that the sum of clay, silt and sand is 100
datCsub <- TT.normalise.sum(tri.data = datCsub)
datVsub <- TT.normalise.sum(tri.data = datVsub)

# plot the soil texture triangle for the calibration data
TextPlot <- TT.plot(class.sys = "AU.TT",
                    tri.data = datCsub,
                    main = " ",
                    pch = 16,
```

```
                    cex.axis = 0.7,
                    cex.lab = 0.7)

# add to the soil texture triangle the validation data
TT.points(tri.data = datVsub,
          geo = TextPlot,
          col = "red",
          pch = 16)
```

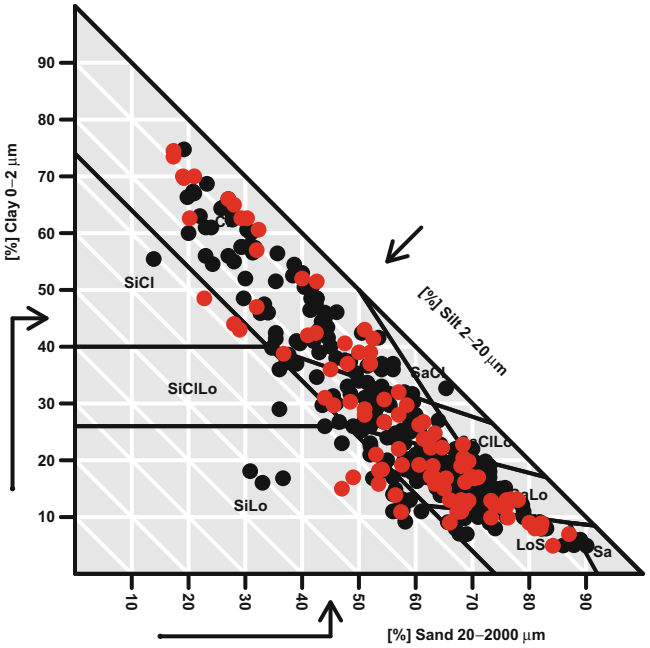

Fig. 9.22 Australian soil texture triangle with location of the calibration (black dots) and validation (red dots) soil texture classes within the triangle

We can assign the name of the Australian soil texture class to the calibration and validation datasets. By default, the points that are at the border of two classes are assigned the name of the two classes. For the demonstration, we remove these points. In most cases, however, it would be more judicious to make further diagnostics on the soil samples and to assign a class to these samples.

```
# make soil texture classes for the calibration dataset
datC$textclass <- TT.points.in.classes(tri.data = datCsub,
                                       class.sys = "AU.TT",
                                       text.tol = 1,
                                       PiC.type = "t",
                                       collapse    = "_")
```

```
# The points that fall into two classes are removed.
datC <- datC[-grep("_", datC$textclass),]

# make soil texture classes for the validation dataset
datV$textclass <- TT.points.in.classes(tri.data = datVsub,
                                       class.sys = "AU.TT",
                                       text.tol = 1,
                                       PiC.type = "t",
                                       collapse    = "_")

# The points that fall into two classes are removed.
datV <- datV[-grep("_", datV$textclass),]

# show the derived categories of soil classes in the calibration dataset
unique(datC$textclass)
```

```
##  [1] "Cl"     "Lo"     "SiLo"   "SaLo"   "ClLo"   "LoSa"   "SiClLo" "SaClLo"
##  [9] "SiCl"   "SaCl"
```

Now we will show how to use two models for predicting soil texture classes.

9.3.1 *Partial Least Squares Discriminant Analysis*

Partial least squares discriminant analysis (PLS-DA, Barker and Rayens 2003) is a linear classification model which builds on the conventional PLS regression. PLS-DA is a PLS regression between two matrices. The first is the matrix $\mathbf{X}$ of size $n \times b$ containing the spectra, where n is the sample size and b is the number of spectral bands. The second is a dummy matrix $\mathbf{Y}$ of size $n \times c$ where c is the total number of classes (e.g. soil texture classes). The columns in Y represent a class membership, that is, a value of 1 or 0 is applied depending on whether the soil sample belongs to the class. The PLS-DA model is then calibrated like PLS, by a linear regression between the rotated scores of the $\mathbf{X}$ and $\mathbf{Y}$ matrices principal components. The calibrated model predicts a value between 0 and 1, which is assigned to the closest class by a softmax function. A discussion on PLS-DA is further provided by Brereton and Lloyd (2014).

In this example, we use the `caret` package which provides functionalities for cross-validation.

```
# load the require package
require(caret)

# create a subset of the calibration dataset
datCSub <- data.frame(soiltext = datC$textclass, datC$spcAMovav)
colnames(datCSub) <- c("textclass", paste0("spec.", colnames(datC$spcAMovav)))

# we do a k-fold cross validation repeated three times
ctrl <- trainControl(method = "repeatedcv",
                     repeats = 3)

# set the seed
set.seed(123)

# build the PLS-DA model using the caret package
soilCPlsdaModel <- train(textclass ~ .,
```

```
                        data = datCSub,
                        method = "pls",
                        preProc = c("center", "scale"),
                        metric = c("Accuracy"),
                        tuneLength = 30,
                        trControl = ctrl)
```

We can display a summary of the fitted PLS-DA model.

```
# summary of the fitted model
soilCPlsdaModel
```

```
## Partial Least Squares
##
## 290 samples
## 421 predictors
##  10 classes: 'Cl', 'ClLo', 'Lo', 'LoSa', 'SaCl', 'SaClLo', 'SaLo', 'SiCl', 'SiClLo',
                            'SiLo'
##
## Pre-processing: centered (421), scaled (421)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 260, 258, 260, 261, 263, 263, ...
## Resampling results across tuning parameters:
##
##   ncomp  Accuracy   Kappa
##    1     0.5534921  0.3461652
##    2     0.5593630  0.3558506
##    3     0.5593189  0.3567091
##    4     0.5650486  0.3656058
##    5     0.5650045  0.3654928
##    6     0.5650045  0.3660005
##    7     0.5708395  0.3736145
##    8     0.5744451  0.3835748
##    9     0.5823474  0.3970096
##   10     0.5640763  0.3746830
##   11     0.5738831  0.3908864
##   12     0.5758560  0.3937251
##   13     0.5850061  0.4117775
##   14     0.5668808  0.3889140
##   15     0.5726905  0.3977221
##   16     0.5853583  0.4178228
##   17     0.5868221  0.4213265
##   18     0.5960500  0.4360405
##   19     0.6052095  0.4509891
##   20     0.6046920  0.4543307
##   21     0.6234867  0.4825286
##   22     0.6270725  0.4894305
##   23     0.6304389  0.4948148
##   24     0.6418556  0.5108486
##   25     0.6484922  0.5213268
##   26     0.6416819  0.5124769
##   27     0.6437369  0.5150439
##   28     0.6414408  0.5116607
##   29     0.6357714  0.5035562
##   30     0.6295964  0.4961511
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was ncomp = 25.
```

The `train` function of the `caret` package automatically selects the optimal number of principal components based on the overall accuracy evaluated by the cross-validation subset left out (Fig. 9.23).

```
# plot the cross-validation results
plot(soilCPlsdaModel)
```

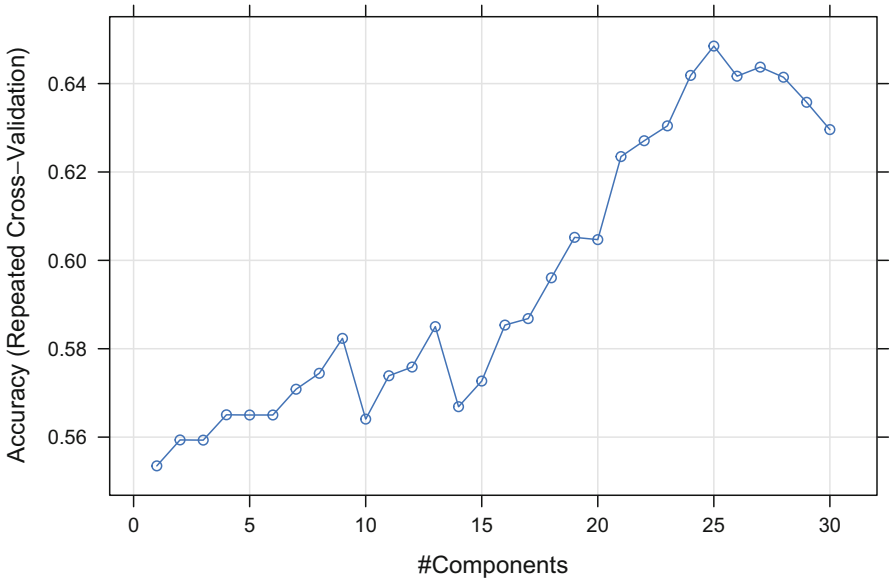

Fig. 9.23 Number of components used in the partial least squares discriminant analysis model against accuracy computed from a repeated cross-validation

The `train` function selected 25 components as the optimal number. The fitted PLS-DA model can now be applied to unseen data.

```
# create a validation dataset
datVSub <- data.frame(soiltext = datV$textclass, datV$spcAMovav)
colnames(datVSub) <- c("textclass", paste0("spec.", colnames(datV$spcAMovav)))

# show the probability of each class
head(predict(soilCPlsdaModel,
             newdata = datVSub,
             type="prob"))
```

```
##          Cl       ClLo         Lo       LoSa       SaCl     SaClLo       SaLo
## 1 0.1201323 0.16463553 0.07922130 0.09377720 0.08920005 0.08531693 0.08447906
## 2 0.2054752 0.10797734 0.06762277 0.10687099 0.08748491 0.08216171 0.07374156
## 3 0.1930218 0.09038021 0.08228958 0.09765407 0.08766723 0.08802700 0.09041075
## 4 0.1163598 0.07402499 0.11089104 0.14420159 0.08809790 0.08977505 0.10009583
## 5 0.2927797 0.05545218 0.09967716 0.07337192 0.08037484 0.08205305 0.07691914
## 6 0.1025771 0.06678866 0.23378266 0.08995787 0.08500366 0.08440278 0.08189733
##         SiCl     SiClLo       SiLo
## 1 0.09069487 0.09012774 0.10241505
## 2 0.08595896 0.08518364 0.09752289
## 3 0.08966126 0.09750844 0.08337970
## 4 0.08502138 0.08641169 0.10512070
## 5 0.08054953 0.07722688 0.08159563
## 6 0.08272078 0.08220722 0.09066194
```

Here we can see that the output is a number between 0 and 1. As mentioned previously, the PLS-DA model returns a probability-like value of the assignment to a class. For example, the fifth soil sample is more likely to belong to the class `Cl` (probability of 0.20) than it is to the other classes (probability below 0.1 in all other classes). To obtain the final results, the softmax function is applied on the probability values to return the single value, that is, the membership to a class. This is done by default in the `caret` package.

We can now visualize the prediction and validate against measured value of the classes.

```
# predict the classes on the validation dataset using the fitted PLS-DA model
plsClasses <- predict(soilCPlsdaModel, newdata = datVSub)

# create a confusion matrix of the predicted versus observed soil texture classes
confMat <- confusionMatrix(datVSub$textclass, plsClasses)

# show confusion matrix computed on the validation dataset
confMat$table
```

```
##            Reference
## Prediction Cl ClLo Lo LoSa SaCl SaClLo SaLo SiCl SiClLo SiLo
##     Cl     21    1  2    0    0      0    0    0      0    0
##     ClLo    6    5  7    0    0      0    0    0      0    0
##     Lo      0    2 24    0    0      0    0    0      0    0
##     LoSa    0    0  5    7    0      0    0    0      0    0
##     SaCl    0    0  0    0    0      0    0    0      0    0
##     SaClLo  0    1  0    0    0      0    0    0      0    0
##     SaLo    0    1  3    2    0      0    0    0      0    0
##     SiCl    2    0  1    0    0      0    0    0      0    0
##     SiClLo  0    0  0    0    0      0    0    0      0    0
##     SiLo    0    0  6    1    0      0    0    0      0    0
```

```
# show accuracy and Cohen's kappa
confMat$overall[1:2]
```

```
##  Accuracy     Kappa
## 0.5876289 0.4584787
```

The confusion matrix `confMat` shows that there is on average a good agreement between predicted and measured soil texture classes. For example, 22 of the reference `Cl` values are correctly classified, and 11 are assigned to a different class, 8 of which are to the most similar class (`ClLo`). The overall accuracy and Cohen's kappa values show that the predictions are accurate.

9.3.2 Random Forest

Random forest applied to categorical variables is similar to the same model applied to continuous variables. Note that parameter tuning is not performed in this example and that, for example, mtry is held constant. The reader will find a large number of examples on how to make parameter tuning in the package vignette. Parameter

tuning will be a key element of the modelling, in particular to avoid overfitting of the model or to avoid an excessive number of trees without compromising on prediction accuracy.

```
# we start by defining the control parameters for the caret function
# we do a k-fold cross validation repeated three times
trainControl <- trainControl(method = "repeatedcv",
                             number = 10,
                             repeats = 3)

# define the random forest parameter mtry (to its default)
mtry <- sqrt(ncol(datCSub))

# hold the mtry parameter constant (not parameter tuning)
tunegrid <- expand.grid(.mtry = mtry)

# set the seed
set.seed(123)

# build the random forest model using the caret package
soilCRFModel <- train(textclass ~ .,
                      data = datCSub,
                      method = "rf",
                      metric = c("Accuracy"),
                      tuneGrid = tunegrid,
                      trControl = trainControl,
                      verbose = FALSE)

# print a summary of the model
print(soilCRFModel)
```

```
## Random Forest
##
## 290 samples
## 421 predictors
##  10 classes: 'Cl', 'ClLo', 'Lo', 'LoSa', 'SaCl', 'SaClLo', 'SaLo', 'SiCl', 'SiClLo',
                'SiLo'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 260, 258, 260, 261, 263, 263, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.6269545  0.5055345
##
## Tuning parameter 'mtry' was held constant at a value of 20.54264
```

Let us now apply the calibrated random forest model to the validation data and display the overall accuracy and Cohen's kappa statistics. Note that the same statistics can be obtained by using the book-associated package `soilspec` with the `eval` function.

```
# predict the classes on the validation dataset using the fitted random forest model
RFClasses <- predict(soilCRFModel,
                     newdata = datVSub)

# create a confusion matrix of the predicted versus observed soil texture classes
confMat <- confusionMatrix(datVSub$textclass, RFClasses)

# show confusion matrix computed on the validation dataset
confMat$table
```

```
##            Reference
## Prediction Cl ClLo Lo LoSa SaCl SaClLo SaLo SiCl SiClLo SiLo
##      Cl    22    2  0    0    0       0    0    0       0    0
##      ClLo   8    8  2    0    0       0    0    0       0    0
##      Lo     0    1 20    4    0       0    1    0       0    0
##      LoSa   0    0  0   12    0       0    0    0       0    0
##      SaCl   0    0  0    0    0       0    0    0       0    0
##      SaClLo 0    1  0    0    0       0    0    0       0    0
##      SaLo   0    0  6    0    0       0    0    0       0    0
##      SiCl   3    0  0    0    0       0    0    0       0    0
##      SiClLo 0    0  0    0    0       0    0    0       0    0
##      SiLo   0    0  4    1    0       0    0    0       0    2
```

```
# show accuracy and Cohen's kappa
confMat$overall[1:2]
```

```
##  Accuracy     Kappa
## 0.6597938 0.5641933
```

9.4 Soil Spectral Inference Systems

Another way to estimate soil properties is to combine the spectra with pedotransfer functions (PTF, Bouma 1989), in a soil spectral inference system (McBratney et al. 2006). Several soil physical, chemical and biological properties cannot be estimated directly from the spectra, as it done in Sects. 9.2 and 9.3. The reasons are that i) some soil properties do not have a clear spectral response in the spectrum and that ii) the development of calibration functions of a soil property from soil spectral libraries is not always possible due, for example, to budget constraints. McBratney et al. (2006) thus proposed a two-step approach to estimate a large range of soil properties by combining spectroscopy and PTFs, as follows:

- Step 1, calibration: a multivariate model is built between the spectra and the measured values of some basic soil properties (that have been shown to demonstrate a spectral response in the spectral region of interest). In the infrared, the basic soil properties are, for example, clay, silt, sand, organic carbon, pH and cation exchange capacity. This step can be implemented using one of the methods presented in Sects. 9.2 or 9.3. When a model is calibrated, it can be used to predict soil properties of a soil sample where only the spectrum is available.
- Step 2, inference: the basic soil properties estimated in Step 1 are used as input in a PTF to predict a set of different soil properties, such as the permanent soil wilting point or field capacity. The PTF can be found in the literature or derived using a large soil database. Ideally, a PTF developed on similar soil is used to derive the soil properties. A number of PTFs have been published, for example, by McBratney et al. (2002) or Pachepsky and Rawls (2004). The PTFs can be concatenated into a network structure to create an inference system that estimates the target soil property or properties and also propagates the uncertainty of the estimate.

So one of the main features of soil spectral inference systems is the quantification and propagation of uncertainty. Model uncertainty, for example, can be quantified using an ensemble model by bootstrap and aggregating, for which an example using partial least squares regression is provided in Sect. 9.2.2.

Since Step 1 is already presented in Sects. 9.2 or 9.3, we do not repeat it here and make a simple example of Step 2. Note also that uncertainty quantification and propagation is discussed elsewhere in the literature (e.g. in Tranter et al. 2010, Van der Klooster et al. 2011 or Brodský et al. 2013).

Take the following PTF from Rab et al. (2011) to estimate the volumetric field capacity (FC, m^3/m^3, %) as a function of basic soil properties. We chose this PTF because it has been derived using data from a case study in Australia, for an area in which the soils are similar to those of the Geeves dataset. The Geeves dataset is provided by the book-associated `soilspec` package. The PTF makes use of the soil clay and silt content.

$$Field\ capacity = 7.759 + 0.7165 \times clay + 0.9708 \times silt - (0.01729 \times clay) \times silt \tag{9.16}$$

We can use the Geeves dataset provided in the `soilspec` package. It contains values of the soil clay and silt content.

```
# load the required package
require(soilspec)

# load the data
data("datsoilspc")

# show the available soil properties
colnames(datsoilspc)
```

```
## [1] "clay"        "silt"        "sand"        "TotalCarbon" "spc"
```

Note here the soil properties are available in our dataset but that in many cases they can be estimated using the spectra (Step 1) or using a soil spectral library (Viscarra-Rossel et al. 2008). We can now derive the field capacity using the basic soil properties clay and silt (Fig. 9.24).

```
# estimate field capacity using a PTF
FC <-  7.759 +
  0.7165*datsoilspc["clay"] + 0.9708*datsoilspc["silt"]  -
  (0.01729*datsoilspc["clay"] )*datsoilspc["silt"]

# plot the distribution of the estimated field capacity
boxplot(FC,
        ylab = "Field capacity")
```

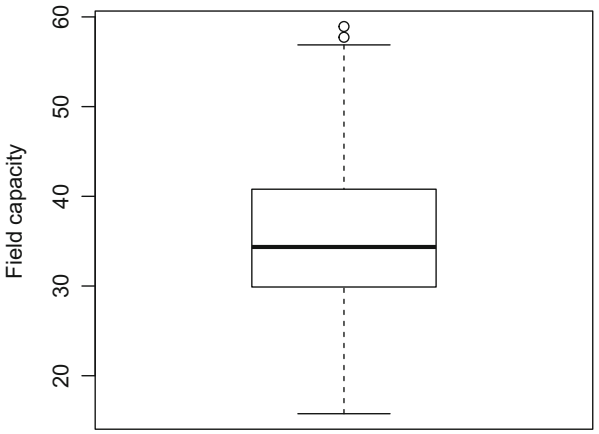

Fig. 9.24 Boxplot of the estimated values of the field capacity (in m^3/m^3, %) of the Geeves dataset

References

Barker M, Rayens W (2003) Partial least squares for discrimination. J Chemometr J Chemometrics Soc 17:166–173

Batten GD (1998) Plant analysis using near infrared reflectance spectroscopy: the potential and the limitations. Aust J Exp Agric 38:697–706

Bellon-Maurel V, Fernandez-Ahumada E, Palagos B, Roger J-M, McBratney AB (2010) Critical review of chemometric indicators commonly used for assessing the quality of the prediction of soil attributes by NIR spectroscopy. TrAC Trends Anal Chem 29:1073–1081

Bouma J (1989) Using soil survey data for quantitative land evaluation. In: Advances in soil science. Springer, Berlin, pp 177–213

Breiman L (2001) Random forests. Mach Learn 45:5–32

Breiman L (1996) Bagging predictors. Mach Learn 24:123–140

Brereton RG, Lloyd GR (2014) Partial least squares discriminant analysis: taking the magic away. J Chemometr 28:213–225

Brodský L, Vasat R, Klement A, Zadorova T, Jaksik O (2013) Uncertainty propagation in VNIR reflectance spectroscopy soil organic carbon mapping. Geoderma 199:54–63

Brus DJ, Kempen B, Heuvelink GBM (2011) Sampling for validation of digital soil maps. Eur J Soil Sci 62:394–407

Chang C-W, Laird DA, Mausbach MJ, Hurburgh CR (2001) Near-infrared reflectance spectroscopy–principal components regression analyses of soil properties. Soil Sci Soc Am J 65:480–490

Cohen J (1960) A coefficient of agreement for nominal scales. Educ Psychol Meas 20:37–46

Esbensen KH, Geladi P, Larsen A (2014) The RPD myth. . . . NIR News 25:24–28

Friedman J, Hastie T, Tibshirani R (2001) The elements of statistical learning. Springer series in statistics, New York

Geeves GW, Cresswell HP, Murphy BW, Gessler PI, Chartres CJ, Little IP, Bowman GM (1994) Physical, chemical and morphological properties of soils in the wheat-belt of southern NSW and northern Victoria. NSW Department of Conservation; Land Management/CSIRO Division of Soils Occasional Report, CSIRO

Hobley EU, Brereton AJLEG, Wilson B (2017) Soil charcoal prediction using attenuated total reflectance mid-infrared spectroscopy. Soil Res 55:86–92

Janssen PHM, Heuberger PSC (1995) Calibration of process-oriented models. Ecol Model 83:55–66

Kuhn M, Weston S, Keefer C, Coulter N (2012) Cubist models for regression. R package Vignette R package version 00 18

Lawrence I, Lin K (1989) A concordance correlation coefficient to evaluate reproducibility. Biometrics 45:255–268

McBratney AB, Minasny B, Cattle SR, Vervoort RW (2002) From pedotransfer functions to soil inference systems. Geoderma 109:41–73

McBratney AB, Minasny B, Viscarra-Rossel RA (2006) Spectral soil analysis and inference systems: A powerful combination for solving the soil data crisis. Geoderma 136:272–278

Mevik B-H, Segtnan VH, Næs T (2004) Ensemble methods and partial least squares regression. J Chemometr J Chemometr Soc 18:498–507

Minasny B, McBratney AB (2013) Why you don't need to use RPD. Pedometron 33:14–15

Minasny B, McBratney AB (2008) Regression rules as a tool for predicting soil properties from infrared reflectance spectroscopy. Chemom Intell Lab Syst 94:72–79

Nash JE, Sutcliffe JV (1970) River flow forecasting through conceptual models part I—A discussion of principles. J Hydrol 10:282–290

National Committee on Soil and Terrain (Australia) and CSIRO Publishing (2009) Australian soil and land survey field handbook. CSIRO Publishing

Ng W, Minasny B, Malone BP, Sarathjith MC, Das BS (2019) Optimizing wavelength selection by using informative vectors for parsimonious infrared spectra modelling. Comput Electron Agric 158:201–210

Northcote KH (1971) Factual key for the recognition of Australian soils

Nussbaum M, Walthert L, Fraefel M, Greiner L, Papritz A (2017) Mapping of soil properties at high resolution in Switzerland using boosted geoadditive models. Soil 3:191–210

Pachepsky Y, Rawls WJ (2004) Development of pedotransfer functions in soil hydrology. Elsevier, Amsterdam

Rab MA, Chandra S, Fisher PD, Robinson NJ, Kitching M, Aumann CD, Imhof M (2011) Modelling and prediction of soil water contents at field capacity and permanent wilting point of dryland cropping soils. Soil Res 49:389–407

Ramirez-Lopez L, Behrens T, Schmidt K, Stevens A, Demattê JAM, Scholten T (2013a) The spectrum-based learner: a new local approach for modeling soil vis–NIR spectra of complex datasets. Geoderma 195:268–279

Santana FB de, Souza AM de, Poppi RJ (2018) Visible and near infrared spectroscopy coupled to random forest to quantify some soil quality parameters. Spectrochim Acta Part A Mol Biomol Spectrosc 191:454–462

Tranter G, Minasny B, McBratney AB (2010) Estimating pedotransfer function prediction limits using fuzzy k-means with extragrades. Soil Sci Soc Am J 74:1967–1975

Van der Klooster E, Van Egmond FM, Sonneveld MPW (2011) Mapping soil clay contents in Dutch marine districts using gamma-ray spectrometry. Eur J Soil Sci 62:743–753

Viscarra-Rossel RA, Jeon YS, Odeh IOA, McBratney AB (2008) Using a legacy soil sample to develop a mid-IR spectral library. Soil Res 46:1–16

Wadoux AMJ-C, Brus DJ, Heuvelink GBM (2018) Accounting for non-stationary variance in geostatistical mapping of soil properties. Geoderma 324:138–147

Wehrens R, Mevik B-H (2007) The pls package: principal component and partial least squares regression in R. J Stat Softw 18:1–24

Williams PC, Thompson BN (1978) Influence of whole meal granularity on analysis of HRS wheat for protein and moisture by near infrared reflectance spectroscopy (NRS). Cereal Chem 55:1014–1037

Wold S, Johansson E, Cocchi M (1993) PLS: partial least squares projections to latent structures. In: 3D qsar in drug design: theory, methods and applications. Kluwer ESCOM Science, Dordrecht, pp 523–550

Wold S, Sjöström M, Eriksson L (2001) PLS-regression: a basic tool of chemometrics. Chemom Intell Lab Syst 58:109–130