Tutorial
Regression modelling for digital soil mapping

B. Kempen
ISRIC - World Soil Information

March 20, 2017

Contents

This tutorial presents an introduction to regression modelling for digital soil mapping using the open-source R project for statistical computing [14]. It focusses on linear regression and random forests modelling and aims to introduce the reader to methods for model selection, checking model assumptions, model fitting and evaluation, using the Edgeroi dataset from Australia [6].

This tutorial shows how to:

1. fit a linear regression model and assess model diagnostics;
2. fit and interpret a random forests model;
3. extend the regression models with a kriging component.

The tutorial contains example code and code that you have to complete or programme yourself. In addition there are several questions regarding the methods and output. Answers to these questions and code solutions can be found at the end of this tutorial.

Several R packages are used, including `GSIF` that contains tools and procedures to handle soil data and produce gridded soil maps, `sp` for spatial data handling [4, 13], `gstat` for geostatistical modelling and prediction [11, 12], `rgdal` for spatial data import, export and geometric transformation [3], `aqp` that contains algorithms to work with soil data in R [1, 2], `randomForest` for random forests modelling [9, 15], `plyr` for efficient data processing [17], and packages `plotKML`, `leaflet`, `htmlwidgets`, `RColourBrewer`, `RSAGA` and `ggplot2` [16] for visualization and plotting.

These must loaded before their first use, as is show in the code.

> **Note:** The code in this document was tested with R version 3.3.3 (2017-03-06) and packages from that version or later running on x86_64-w64-mingw32 (64-bit).
>
> This document was created by means of 'literate programming' [7]. This means that the text and graphical output you see here was written as a NoWeb file (.Rnw), including both R code and regular LaTeX source, and then run through the `knitr` package Version: 1.15.1 [18] on R and automatically generated and incorporated into LaTeX. Then the LaTeX document was compiled into the PDF version you are now reading. The R code (file `RegressionModelling.R`, supplied with this document) was generated by `knitr` from the same source document. If you run this R code, or copy code from this document, your output may be slightly different on different R-versions and on different operating platforms.

1 Initialization

TASK 1: Load R-packages. •

We start by loading the required R-packages and by setting a random seed to make the results reproducible in case a random process is used.

```
> # load libraries
> require(GSIF)
> require(plotKML)
> require(sp)
> require(gstat)
> require(rgdal)
> require(aqp)
> require(randomForest)
> require(plyr)
> require(ggplot2)
> require(e1071)
> require(leaflet)
> library(htmlwidgets)
> library(RColorBrewer)
> library(RSAGA)
>
> # set random seed
> set.seed(170317)
```

TASK 2: Define geographic projection •

The Edgeroi soil data are georeferenced in the GRS80 (decimal degrees) system, the Edgeroi covariate data in the metric GDA94 system. We create two objects that store the referencing definitions. These objects will be used later to georeference the data. The expressions for the coordinate systems can be obtained from spatialreference.org (use the references in `proj4` format).

```
> grs80 <- "+proj=longlat +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +no_defs"
> wgs84 <- "+proj=longlat +datum=WGS84  +no_defs +ellps=WGS84  +towgs84=0,0,0"
> gda94 <- "+proj=utm +zone=55 +south +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs
```

2 Soil data exploration and preprocessing

TASK 3: Load and explore Edgeroi soil data •

The Edgeroi soil dataset is included in the `GSIF` package. We will start by loading this dataset.

```
> data(edgeroi)
```

Let's explore this dataset.

```
> str(edgeroi)

List of 2
 $ sites   :'data.frame': 359 obs. of  5 variables:
  ..$ SOURCEID: Factor w/ 359 levels "199_CAN_CP111_1",..: 1 2 3 4 5 6 7 8 9 10 ...
  ..$ LONGDA94: num [1:359] 150 150 150 150 150 ...
  ..$ LATGDA94: num [1:359] -30.1 -30.1 -30.2 -30.2 -30.2 ...
  ..$ TAXGAUC : Factor w/ 18 levels "A","BC","BE",..: 5 3 5 3 5 5 5 5 5 2 ...
  ..$ NOTEOBS : chr [1:359] "FREQUENT SLICKENSIDES >120CM
 $ horizons:'data.frame': 2338 obs. of  10 variables:
  ..$ SOURCEID: Factor w/ 359 levels "199_CAN_CP111_1",..: 1 1 1 1 1 1 1 1 1 1 ...
  ..$ LSQINT  : int [1:2338] 1 10 11 12 13 14 2 3 4 5 ...
  ..$ HZDUSD  : Factor w/ 6 levels "A","AC","B","C",..: NA NA NA NA NA NA NA NA NA NA ...
  ..$ UHDICM  : num [1:2338] 0 90 100 110 120 ...
  ..$ LHDICM  : num [1:2338] 10 100 110 120 130 ...
  ..$ CLYPPT  : num [1:2338] 36 NA NA NA NA NA NA NA NA NA ...
  ..$ SNDPPT  : num [1:2338] 43 NA NA NA NA NA NA NA NA NA ...
  ..$ SLTPPT  : num [1:2338] 18 NA NA NA NA NA NA NA NA NA ...
  ..$ PHIHO5  : num [1:2338] 6.6 8.3 8 8.3 8.5 ...
  ..$ ORCDRC  : num [1:2338] NA NA NA NA NA NA NA NA NA NA ...
```

```
> head(edgeroi[[1]][, 1:4])

           SOURCEID LONGDA94  LATGDA94 TAXGAUC
145 199_CAN_CP111_1 149.6845 -30.14842      GC
146 199_CAN_CP112_1 149.6178 -30.13176      BE
149 199_CAN_CP115_1 149.6511 -30.19842      GC
150 199_CAN_CP116_1 149.6178 -30.21509      BE
184 199_CAN_CP183_1 149.5159 -30.19759      GC
185 199_CAN_CP188_1 149.6270 -30.14314      GC

> head(edgeroi[[2]])

            SOURCEID LSQINT HZDUSD UHDICM LHDICM CLYPPT SNDPPT SLTPPT
1051 199_CAN_CP111_1      1   <NA>      0     10     36     43     18
1052 199_CAN_CP111_1     10   <NA>     90    100     NA     NA     NA
1053 199_CAN_CP111_1     11   <NA>    100    110     NA     NA     NA
1054 199_CAN_CP111_1     12   <NA>    110    120     NA     NA     NA
1055 199_CAN_CP111_1     13   <NA>    120    130     NA     NA     NA
1056 199_CAN_CP111_1     14   <NA>    130    140     NA     NA     NA
     PHIHO5 ORCDRC
1051    6.6     NA
1052    8.3     NA
1053    8.0     NA
1054    8.3     NA
1055    8.5     NA
1056    8.6     NA
```

You see that the the R object `edgeroi` contains two lists: one containing the site data (e.g. coordinates, soil classification) and one containing the profile descriptions.

TASK 4 : Derive depth-specific soil property values •

We will model and map the clay content for the Edgeroi case study area for the 5–15-cm layer (GlobalSoilMap standard depth 2). The first thing we have to do is to extract clay data for this layer from the Edgeroi soil profile descriptions. There are several ways to do this. Here we will fit a mass-preserving spline [10] to the data and determine the clay content for standard depth 2 from the fitted spline. A mass-preserving spline can be fit with the `mpspline` function of the `GSIF` package. In order to do this, we have to process the soil profiles first. The `mpsplne` function must be applied to an object with class `SoilProfileCollection`, which is a class that belongs to the `aqp` package.

Create a `SoilProfileCollection` object.

```
> # copy the horizon information to a new object
> edgeroi.spc <- edgeroi[[2]]
>
> # convert to SoilProfileCollection class
> depths(edgeroi.spc) <- SOURCEID ~ UHDICM + LHDICM
>
> # inspect result
> str(edgeroi.spc)

Formal class 'SoilProfileCollection' [package "aqp"] with 7 slots
  ..@ idcol     : chr "SOURCEID"
  ..@ depthcols : chr [1:2] "UHDICM" "LHDICM"
  ..@ metadata  :'data.frame': 1 obs. of  1 variable:
  .. ..$ depth_units: chr "cm"
  ..@ horizons  :'data.frame': 2338 obs. of  10 variables:
  .. ..$ SOURCEID: chr [1:2338] "199_CAN_CP111_1" "199_CAN_CP111_1" "199_CAN_CP111_1" "199_
```

```
  .. ..$ LSQINT  : int [1:2338] 1 2 3 4 5 6 7 8 9 10 ...
  .. ..$ HZDUSD  : Factor w/ 6 levels "A","AC","B","C",..: NA NA NA NA NA NA NA NA NA NA ..
  .. ..$ UHDICM  : num [1:2338] 0 10 20 30 40 ...
  .. ..$ LHDICM  : num [1:2338] 10 20 30 40 50 ...
  .. ..$ CLYPPT  : num [1:2338] 36 NA NA NA NA NA NA NA NA NA ...
  .. ..$ SNDPPT  : num [1:2338] 43 NA NA NA NA NA NA NA NA NA ...
  .. ..$ SLTPPT  : num [1:2338] 18 NA NA NA NA NA NA NA NA NA ...
  .. ..$ PHIHO5  : num [1:2338] 6.6 7.8 8.5 8.9 9 ...
  .. ..$ ORCDRC  : num [1:2338] NA NA NA NA NA NA NA NA NA NA ...
  ..@ site      :'data.frame': 359 obs. of  1 variable:
  .. ..$ SOURCEID: chr [1:359] "199_CAN_CP111_1" "199_CAN_CP112_1" "199_CAN_CP115_1" "199_C
  ..@ sp        :Formal class 'SpatialPoints' [package "sp"] with 3 slots
  .. .. ..@ coords     : num [1, 1] 0
  .. .. ..@ bbox       : logi [1, 1] NA
  .. .. ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
  .. .. .. .. ..@ projargs: chr NA
  ..@ diagnostic:'data.frame': 0 obs. of  0 variables
```

You can see that the `SoilProfileCollection` object has seven slots. One slot is called `site` that contains the profile identifiers. We will use the `join` function of the `plyr` package to join all site information that is contained in the `site` slot of the `edgeroi` object. We will then add a projection to the `Edgeroi SoilProfileCollection` object. Projecting the data can be done with the `CRS` function of the `sp` package.

```
> # add site data to SoilProfileCollection object
> site(edgeroi.spc) <- join(x = edgeroi.spc@site, y = edgeroi[[1]], by = "SOURCEID")
>
> # add coordinates to SoilProfileCollection object
> coordinates(edgeroi.spc) <- ~LONGDA94 + LATGDA94
> proj4string(edgeroi.spc) <- CRS(grs80)
>
> # inspect the result
> str(edgeroi.spc)

Formal class 'SoilProfileCollection' [package "aqp"] with 7 slots
  ..@ idcol     : chr "SOURCEID"
  ..@ depthcols : chr [1:2] "UHDICM" "LHDICM"
  ..@ metadata  :'data.frame': 1 obs. of  1 variable:
  .. ..$ depth_units: chr "cm"
  ..@ horizons  :'data.frame': 2338 obs. of  10 variables:
  .. ..$ SOURCEID: chr [1:2338] "199_CAN_CP111_1" "199_CAN_CP111_1" "199_CAN_CP111_1" "199_
  .. ..$ LSQINT  : int [1:2338] 1 2 3 4 5 6 7 8 9 10 ...
  .. ..$ HZDUSD  : Factor w/ 6 levels "A","AC","B","C",..: NA NA NA NA NA NA NA NA NA NA ..
  .. ..$ UHDICM  : num [1:2338] 0 10 20 30 40 ...
  .. ..$ LHDICM  : num [1:2338] 10 20 30 40 50 ...
  .. ..$ CLYPPT  : num [1:2338] 36 NA NA NA NA NA NA NA NA NA ...
  .. ..$ SNDPPT  : num [1:2338] 43 NA NA NA NA NA NA NA NA NA ...
  .. ..$ SLTPPT  : num [1:2338] 18 NA NA NA NA NA NA NA NA NA ...
  .. ..$ PHIHO5  : num [1:2338] 6.6 7.8 8.5 8.9 9 ...
  .. ..$ ORCDRC  : num [1:2338] NA NA NA NA NA NA NA NA NA NA ...
  ..@ site      :'data.frame': 359 obs. of  3 variables:
  .. ..$ SOURCEID: chr [1:359] "199_CAN_CP111_1" "199_CAN_CP112_1" "199_CAN_CP115_1" "199_C
  .. ..$ TAXGAUC : Factor w/ 18 levels "A","BC","BE",..: 5 3 5 3 5 5 5 5 5 2 ...
  .. ..$ NOTEOBS : chr [1:359] "FREQUENT SLICKENSIDES >120CM
  ..@ sp        :Formal class 'SpatialPoints' [package "sp"] with 3 slots
  .. .. ..@ coords     : num [1:359, 1:2] 150 150 150 150 150 ...
  .. .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. .. ..$ : chr [1:359] "1" "2" "3" "4" ...
  .. .. .. .. ..$ : chr [1:2] "LONGDA94" "LATGDA94"
  .. .. ..@ bbox       : num [1:2, 1:2] 149.5 -30.3 150 -30
  .. .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. .. ..$ : chr [1:2] "LONGDA94" "LATGDA94"
  .. .. .. .. ..$ : chr [1:2] "min" "max"
  .. .. ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
  .. .. .. .. ..@ projargs: chr "+proj=longlat +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +no_defs
  ..@ diagnostic:'data.frame': 0 obs. of  0 variables
```

When you inspect the object you will see that the `sp` slot now contains spatial data and a projection.

We are now ready to fit mass-preserving splines to the soil profile data.

```
> mps <- mpspline(edgeroi.spc, var.name = "CLYPPT", vlow = 0, vhigh = 100)
```

Note that for a few profiles it is not possible to fit a spline. This is because at least two horizons are required for fitting.

Inspect output.

```
> str(mps)

List of 4
 $ idcol     : num [1:359] 0 0 0 0 0 ...
 $ var.fitted: num [1:359, 1:14] NA NA NA NA 49 ...
 $ var.std   :'data.frame': 359 obs. of  7 variables:
  ..$ 0-5 cm    : num [1:359] 36 55 71 54 48.9 ...
  ..$ 5-15 cm   : num [1:359] 36 55 71 NA 49.6 ...
  ..$ 15-30 cm  : num [1:359] NA NA NA NA 51.5 ...
  ..$ 30-60 cm  : num [1:359] NA NA NA NA 52.6 ...
  ..$ 60-100 cm : num [1:359] NA NA NA NA 50.8 ...
  ..$ 100-200 cm: num [1:359] NA NA NA NA NA NA NA NA NA ...
  ..$ soil depth: num [1:359] 10 10 10 6 100 ...
 $ var.1cm   : num [1:200, 1:359] 36 36 36 36 36 36 36 36 36 36 ...
```

As you can see, the `mpspline` function automatically gives estimates for the GlobalSoilMap.net standard dephts. The values for the 5–15-cm layer can be obtained from these. In addition, the function gives the spline-estimated vlaues of the target variable for 1 cm increments for a depth up to 2 m.

NOTE: because of a bug in the current version of `GSIF` package (0.5.3) the profile identifiers are lost when these contain characters, as is the case here. We have to reassing the identifiers by copying these from teh `edgeroi.spc` object.

```
> mps$idcol <- edgeroi.spc@site$SOURCEID
```

We will store the clay content of standard depth 2 in a new `data.frame` and add the coordinates.

```
> d <- data.frame(id = mps$idcol, x = edgeroi.spc@sp@coords[, 1], y = edgeroi.spc@sp@coords
+     2], z = mps$var.std[, 2])
```

Let's round the clay content to 1 decimal.

```
> d$z <- round(d$z, digits = 1)
```

TASK 5 : Load and extract covariate data. •

Edgeroi covariate data have been compiled in a `data.frame` that is stored in the `edgeroi.grids.rda` file that accompanies this tutorial. The covariates have been extracted from 100-m and 250-m resolution grids. You can find more information about the Edgeroi covariates on the GSIF package documentation. Load the covariate data.

```
> load("edgeroi.grids.rda")
```

We can now convert point data to a `SpatialPointsDataFrame` object.

Also, we have to define the projection. The Edgeroi data object contains coordinates that are georeferenced in decimal degrees (based on the GRS80 ellipsoid). We need to 'tell' this to R. With the proj4string function one can get or set the coordinate reference system (CRS).

```
> # create spatial objects
> coordinates(d) <- ~x + y
>
> # get the projection
> proj4string(d)  # no projection set

[1] NA

> # set projection
> proj4string(d) <- CRS(grs80)
> proj4string(d)

[1] "+proj=longlat +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +no_defs"
```

Spatial analysis however, requires a metric projection. We therefore have to reproject the data from GRS80 decimal degrees to the metric GDA94. We use the `spTransform` function of the `sp` package for this purpose.

```
> # reproject to GDA94 system
> d <- spTransform(d, CRSobj = CRS(gda94))
```

Convert the Edgeroi grid to a `SpatialPixelsDataFrame` object, and set a projection. Note that the grid data is already in the metric GDA94 projection so we do not have to reproject.

```
> # convert grid data to spatial object
> gridded(edgeroi.grids) <- ~x + y
> proj4string(edgeroi.grids) <- CRS(gda94)
```

We extract covariate values from the grids at the data points using a spatial overlay opertation. Then we append the covariate data to the `data.frame` that contains the point data.

```
> # overlay
> dum <- over(x = d, y = edgeroi.grids)
>
> # append covariate data to point data
> d@data <- cbind(d@data, dum)
>
> # clean-up
> rm(dum)
```

Before we can start modelling, we have to do a bit more processing of the data. To avoid problems during spatial analysis we must make sure that there are no observations with identical coordinates, and remove all observations that do not have covariate data.

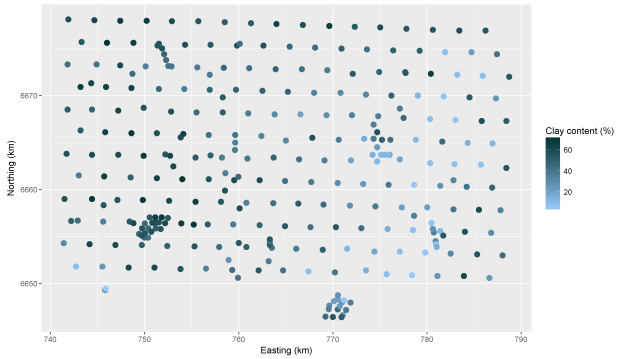
```
> # identify points with similar coordinates (gives problems with spatial
> # modelling)
> zerodist(d, zero = 0, unique.ID = FALSE)
>
> # remove duplicates
> d <- remove.duplicates(d, zero = 0, remove.second = TRUE)
>
> # convert to data.frame
> d <- as(d, Class = "data.frame")
>
> # remove observations without covariate data, clay data or coordinates
> d <- na.omit(d)
```

3 Linear regression modelling

TASK 6: Explore and plot data. •

Plot the clay content with the `ggplot2` package.

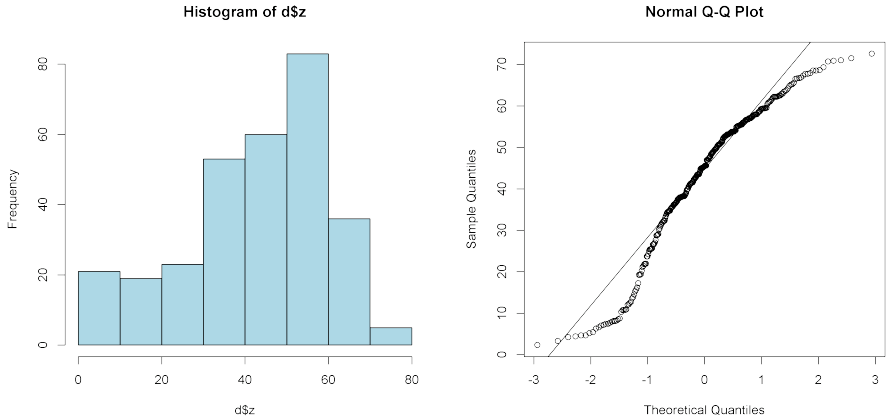
```
> # create ggplot object
> gg.z <- ggplot() + geom_point(data = d, mapping = aes(x = x * 0.001, y = y *
+     0.001, colour = z), size = 3) + scale_colour_continuous(name = "Clay content (%)",
+     low = "#99CCFF", high = "#003333") + scale_x_continuous(name = "Easting (km)") +
+     scale_y_continuous(name = "Northing (km)") + coord_equal(ratio = 1)
>
> # plot
> gg.z
```



The pattern in the plot indicates that there is spatial correlation in the data: there are regions with relatively high values and regions with relatively low values.

Look at some summary statistics, skewness coefficient, histogram and qq-plot.

```
> # summary statistics
> summary(d$z)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   2.30   33.70   45.45   42.75   55.90   72.60

> # skewness statistics (e1071 package)
> skewness(d$z)

[1] -0.627055

> # histogram
> hist(d$z, col = "lightblue")
>
> # qqplot
> qqnorm(d$z)
> qqline(d$z)
```



Q1 : *Based on these statisics and plot, should we transform the data before fitting a linear regression model?* *Jump to A1* •

TASK 7 : Fit a linear regression model. •

We begin our modelling with defining a trend model. The trend model is an object of class `formula`. The formula has the general structure: 'target variable ~ covariates'. Here our target variable (clay) is stored in attribute `z`. The ~ means 'is a function of'. We assign the trend model to object 'trend'. The advantage of creating an object that stores the trend models is that you do not have to write out trend model each time you need it. You just use the object name. You can also create a formula by pasting the variable names together in two steps, which can be convenient if you have many covariates. First, we paste the covariate names together. We obtain the names with the `names` function of the data object and select the covariate columns by number (first check which columns store the covariates). Second, paste 'z ~' to the covariate string.

```
> # define trend model
> trend <- z ~ MVBSRT6 + TI1LAN6 + TI2LAN6 + PCKGAD6 + RUTGAD6 + PCTGAD6 +
+     DEMSRT5 + TWISRT5 + PMTGEO5 + EV1MOD5 + EV2MOD5 + EV3MOD5
> class(trend)

[1] "formula"

> # or alternatively and more generic
> names(d)

 [1] "id"       "z"        "MVBSRT6"  "TI1LAN6"  "TI2LAN6"  "PCKGAD6"
 [7] "RUTGAD6"  "PCTGAD6"  "DEMSRT5"  "TWISRT5"  "PMTGEO5"  "EV1MOD5"
[13] "EV2MOD5"  "EV3MOD5"  "x"        "y"

> trend <- as.formula(paste("z~", paste(names(d)[3:14], collapse = "+")))
> trend

z ~ MVBSRT6 + TI1LAN6 + TI2LAN6 + PCKGAD6 + RUTGAD6 + PCTGAD6 +
    DEMSRT5 + TWISRT5 + PMTGEO5 + EV1MOD5 + EV2MOD5 + EV3MOD5
```

Now we can fit the linear regression modelling.

```
> # fit lm model
> z.lm <- lm(trend, data = d)
>
> # inspect model
> summary(z.lm)


Call:
lm(formula = trend, data = d)

Residuals:
     Min       1Q   Median       3Q      Max
-29.0443  -7.8035   0.8322   6.4996  31.2668

Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)   -15.35910   29.07371  -0.528 0.597718
MVBSRT6         5.90990    0.95061   6.217 1.81e-09 ***
TI1LAN6         0.03674    0.02181   1.684 0.093233 .
TI2LAN6        -0.04747    0.02697  -1.760 0.079461 .
PCKGAD6         3.75471    1.78444   2.104 0.036249 *
RUTGAD6       -45.84053   18.74868  -2.445 0.015095 *
PCTGAD6        -2.44021    0.63274  -3.857 0.000142 ***
DEMSRT5         0.05378    0.03312   1.624 0.105498
TWISRT5         0.14777    1.16547   0.127 0.899194
PMTGEO5Qrs     37.57998    6.97816   5.385 1.52e-07 ***
PMTGEO5Qrt/Jp  28.46376    7.27337   3.913 0.000114 ***
PMTGEO5Qrt/Rn  60.62062   10.91998   5.551 6.52e-08 ***
PMTGEO5Qrt/Tv  51.56366    7.98798   6.455 4.68e-10 ***
PMTGEO5Tv      50.32176    8.49459   5.924 9.10e-09 ***
EV1MOD5        -0.06732    0.25603  -0.263 0.792800
EV2MOD5        -0.92425    0.30102  -3.070 0.002345 **
EV3MOD5        -1.35173    0.34418  -3.927 0.000108 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 11.14 on 283 degrees of freedom
Multiple R-squared:  0.606,Adjusted R-squared:  0.5838
F-statistic: 27.21 on 16 and 283 DF,  p-value: < 2.2e-16
```

Q2 : *Are all covariates significant? What is the Akaike Information Criterion (AIC) of this model (hint: use the AIC() function)?* *Jump to A2*

•

Select a linear regression model with stepwise regression.

```
> # stepwise selection of covariates
> z.slm <- step(z.lm)
```

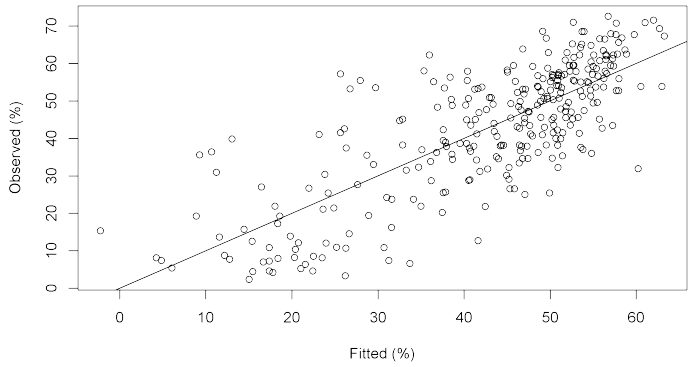
Inspect the model.

```
> # inspect model
> summary(z.slm)


Call:
lm(formula = z ~ MVBSRT6 + TI1LAN6 + TI2LAN6 + PCKGAD6 + RUTGAD6 +
    PCTGAD6 + DEMSRT5 + PMTGEO5 + EV2MOD5 + EV3MOD5, data = d)

Residuals:
    Min      1Q  Median      3Q     Max
-28.942  -7.720   0.779   6.627  31.538

Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)   -11.69117   13.02890  -0.897  0.37030
```

```
MVBSRT6          5.86268    0.93164   6.293 1.17e-09 ***
TI1LAN6          0.03587    0.02155   1.665  0.09708 .
TI2LAN6         -0.04808    0.02678  -1.795  0.07365 .
PCKGAD6          3.64180    1.73193   2.103  0.03636 *
RUTGAD6        -46.63049   18.33520  -2.543  0.01151 *
PCTGAD6         -2.48299    0.61032  -4.068 6.13e-05 ***
DEMSRT5          0.05218    0.03179   1.641  0.10182
PMTGEO5Qrs      37.93587    6.80173   5.577 5.67e-08 ***
PMTGEO5Qrt/Jp   28.59485    7.21908   3.961 9.44e-05 ***
PMTGEO5Qrt/Rn   60.70085   10.82783   5.606 4.89e-08 ***
PMTGEO5Qrt/Tv   51.68369    7.92785   6.519 3.20e-10 ***
PMTGEO5Tv       50.28767    8.38463   5.998 6.05e-09 ***
EV2MOD5         -0.93435    0.29685  -3.148  0.00182 **
EV3MOD5         -1.35080    0.34010  -3.972 9.04e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 11.11 on 285 degrees of freedom
Multiple R-squared:  0.6059,Adjusted R-squared:  0.5865
F-statistic:  31.3 on 14 and 285 DF,  p-value: < 2.2e-16
```

Plot fitted versus observed values.

```
> plot(z.slm$fitted.values, d$z, xlab = "Fitted (%)", ylab = "Observed (%)")
> abline(0, 1)
```

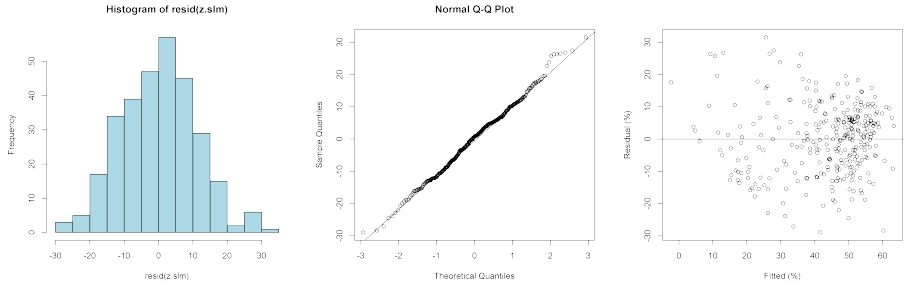


Q3 : *What is the AIC of this model? Which model should we choose for further analysis, this model or the full model? How much of the variation in the data is explained by the model?* *Jump to A3* •

TASK 8 : Examine model residuals. •

```
> # summary statistics
> summary(resid(z.slm))

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-28.940  -7.720   0.779   0.000   6.627  31.540

> # skewness statistics (e1071 package)
> skewness(resid(z.slm))

[1] 0.05021202

> # histogram
> hist(resid(z.slm), col = "lightblue")
```

```
>
> # qqplot
> qqnorm(resid(z.slm))
> qqline(resid(z.slm))
>
> # residual plot
> plot(z.slm$fitted.values, resid(z.slm), xlab = "Fitted (%)", ylab = "Residual (%)")
> abline(0, 0)
```



Q4 : *Take a look at the residual diagnostics. Should we transform the clay content before fitting a regression model? Which assumption we did not evaluate?* *Jump to A4* •

TASK 9 : Uncertainty assessment. •

To start, we will compute the uncertainty associated to the regression model fit. We use the se.fit argument in the predict.lm function to obtain the standard errors of the fit and the residual standard deviation.

```
> # predict with standard errors
> p <- predict.lm(z.slm, data = d, se.fit = TRUE)
> str(p)

List of 4
 $ fit           : Named num [1:300] 54.8 53.6 52.7 55.5 53.4 ...
  ..- attr(*, "names")= chr [1:300] "1" "2" "3" "5" ...
 $ se.fit        : num [1:300] 1.53 1.48 1.53 1.71 2.02 ...
 $ df            : int 285
 $ residual.scale: num 11.1
```

Next, we will compute the 95% confidence interval of the fitted value (i.e. the model mean or expected value).

```
> # 95% confidence interval of the mean (expected value)
> p$ci.l <- p$fit - 1.96 * p$se.fit
> p$ci.l <- unname(p$ci.l)  # lower limit
> p$ci.u <- p$fit + 1.96 * p$se.fit
> p$ci.u <- unname(p$ci.u)  # upper limit
> str(p)

List of 6
 $ fit           : Named num [1:300] 54.8 53.6 52.7 55.5 53.4 ...
  ..- attr(*, "names")= chr [1:300] "1" "2" "3" "5" ...
 $ se.fit        : num [1:300] 1.53 1.48 1.53 1.71 2.02 ...
 $ df            : int 285
 $ residual.scale: num 11.1
 $ ci.l          : num [1:300] 51.8 50.7 49.7 52.1 49.5 ...
 $ ci.u          : num [1:300] 57.8 56.5 55.7 58.8 57.4 ...
```

Last, we will compute the 90% prediction interval from the prediction stan-

dard deviation. To do this we need to compute the prediction error standard deviation from the standard error of the fit and the residual standard deviation (note we do these calculation with variances and then take the square root to obtain the standard deviation).

```
> # compute the prediction standard deviation
> u <- sqrt(p$se.fit^2 + p$residual.scale^2)
> head(round(u, 3))

[1] 11.210 11.204 11.210 11.237 11.288 11.343

> # 90% prediction interval
> p$pi.l <- p$fit - 1.645 * u
> p$pi.l <- unname(p$pi.l)
> p$pi.u <- p$fit + 1.645 * u
> p$pi.u <- unname(p$pi.u)
> str(p)

List of 8
 $ fit           : Named num [1:300] 54.8 53.6 52.7 55.5 53.4 ...
  ..- attr(*, "names")= chr [1:300] "1" "2" "3" "5" ...
 $ se.fit        : num [1:300] 1.53 1.48 1.53 1.71 2.02 ...
 $ df            : int 285
 $ residual.scale: num 11.1
 $ ci.l          : num [1:300] 51.8 50.7 49.7 52.1 49.5 ...
 $ ci.u          : num [1:300] 57.8 56.5 55.7 58.8 57.4 ...
 $ pi.l          : num [1:300] 36.4 35.1 34.3 37 34.8 ...
 $ pi.u          : num [1:300] 73.3 72 71.1 74 72 ...
```

TASK 10 : Fit a variogram to the residuals. •

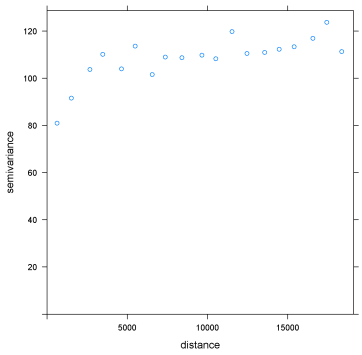
We will continue with the second part of the regression-kriging framework: spatial modelling and interpolation of the residuals.

We start by creating an object of class `formula` that stores the trend model. We include covariates that were selected with the stepwise procedure.

```
> # define selected trend model (based on stepwise selection)
> trend.sw <- z ~ MVBSRT6 + TI1LAN6 + TI2LAN6 + PCKGAD6 + RUTGAD6 + PCTGAD6 +
+     DEMSRT5 + PMTGEO5 + EV2MOD5 + EV3MOD5
```

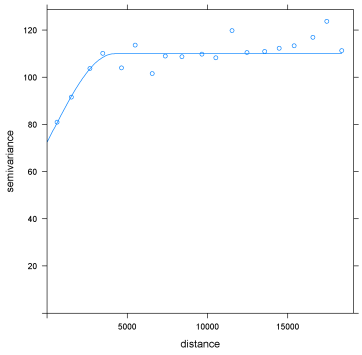
Before we can calculate the sample variogram with have to convert the data object to class `SpatialPointsDataFrame`, which is a class for storing spatial point data. with the `CRS` function we can define the coordinate system. The sample variogram is computed with the `variogram` function.

```
> # convert data.frame to a spatial object
> coordinates(d) <- ~x + y
>
> # set coordinate projection
> proj4string(d) <- CRS(gda94)
>
> # compute sample variogram from the trend residuals
> v <- variogram(trend.sw, data = d, width = 1000)
> plot(v)
```



Finally, we define a variogram model that contains the initial parameters for variogram parameters. With the `variogram` function we estimate the variogram model from the sample variogram.

```
> # define variogram model: initial values for psill, range, nugget based
> # on sample variogram
> vm <- vgm(psill = 30, model = "Sph", range = 5000, nugget = 80)
>
> # fit variogram model to the experimental variogram
> vmf <- fit.variogram(v, model = vm)
>
> # plot variograms
> plot(v, vmf)
```



TASK 11 : Spatial prediction: regression-kriging •

Now that we have obtained the variogram model, we can proceed with predicting the clay content at the nodes of a regular grid that covers the Edgeroi area. In this tutorial, two prediction methods will be illustrated: regression-kriging (separate prediction of trend and residuals) and kriging with an external drift (simultaneous prediction of trend and residuals).

Start with the trend predictions.

```
> p.tr <- predict(z.slm, newdata = edgeroi.grids)
```

Proceed with kriging of the residuals using the `krige` function of the `gstat` package. By setting the argument `debug.level` to -1, we can keep track of the progress. First, we will copy the residual values to the `data.frame` that contains the point data.

```
> d$resid <- resid(z.slm)
```

Krige the residuals.

```
> rk <- krige(resid ~ 1, locations = d, newdata = edgeroi.grids, model = vmf,
+     debug.level = -1)
```

Inspect the results. Try to understand the `SpatialPixelsDataFrame` object.

```
> str(rk)

Formal class 'SpatialPixelsDataFrame' [package "sp"] with 7 slots
  ..@ data       :'data.frame': 151975 obs. of  2 variables:
  .. ..$ var1.pred: num [1:151975] 1.55 1.62 1.7 1.77 1.85 ...
  .. ..$ var1.var : num [1:151975] 101.3 100.4 99.6 98.7 97.8 ...
  ..@ coords.nrs : int [1:2] 1 2
  ..@ grid       :Formal class 'GridTopology' [package "sp"] with 3 slots
  .. .. ..@ cellcentre.offset: Named num [1:2] 741450 6646150
  .. .. .. ..- attr(*, "names")= chr [1:2] "x" "y"
  .. .. ..@ cellsize         : Named num [1:2] 100 100
  .. .. .. ..- attr(*, "names")= chr [1:2] "x" "y"
  .. .. ..@ cells.dim        : Named int [1:2] 475 320
  .. .. .. ..- attr(*, "names")= chr [1:2] "x" "y"
  ..@ grid.index : int [1:151975] 1 2 3 4 5 6 7 8 9 10 ...
  ..@ coords     : num [1:151975, 1:2] 741450 741550 741650 741750 741850 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:151975] "1" "2" "3" "4" ...
  .. .. ..$ : chr [1:2] "x" "y"
  ..@ bbox       : num [1:2, 1:2] 741450 6646150 788850 6678050
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2] "x" "y"
  .. .. ..$ : chr [1:2] "min" "max"
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
  .. .. ..@ projargs: chr "+proj=utm +zone=55 +south +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +u
```

Append the trend predictions stored in object `p.tr` to the object that holds the kriged residuals, and then add these to the predicted residuals.

```
> rk$trend <- p.tr
> rk$pred <- rk$trend + rk$var1.pred
```

Compute the 90% prediction interval.

```
> rk$lower <- rk$pred - 1.645 * sqrt(rk$var1.var)
> rk$upper <- rk$pred + 1.645 * sqrt(rk$var1.var)
>
> # summarize predictions
> summary(rk@data)

   var1.pred           var1.var          trend             pred
 Min.   :-9.69986   Min.   : 81.75   Min.   :-51.84   Min.   :-52.93
 1st Qu.:-2.03068   1st Qu.:100.04   1st Qu.: 37.40   1st Qu.: 36.51
 Median : 0.05696   Median :101.80   Median : 46.27   Median : 45.61
 Mean   :-0.04732   Mean   :101.64   Mean   : 42.82   Mean   : 42.77
 3rd Qu.: 1.94265   3rd Qu.:102.66   3rd Qu.: 52.57   3rd Qu.: 52.98
 Max.   :11.86253   Max.   :110.54   Max.   : 74.95   Max.   : 75.26
     lower             upper
 Min.   :-69.44   Min.   :-36.41
 1st Qu.: 19.89   1st Qu.: 53.10
 Median : 29.04   Median : 62.19
 Mean   : 26.19   Mean   : 59.35
 3rd Qu.: 36.44   3rd Qu.: 69.56
 Max.   : 58.59   Max.   : 91.92

> # summary regression-kriging standard deviation
> summary(sqrt(rk$var1.var))

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
```

```
  9.042  10.000  10.090  10.080  10.130  10.510
```

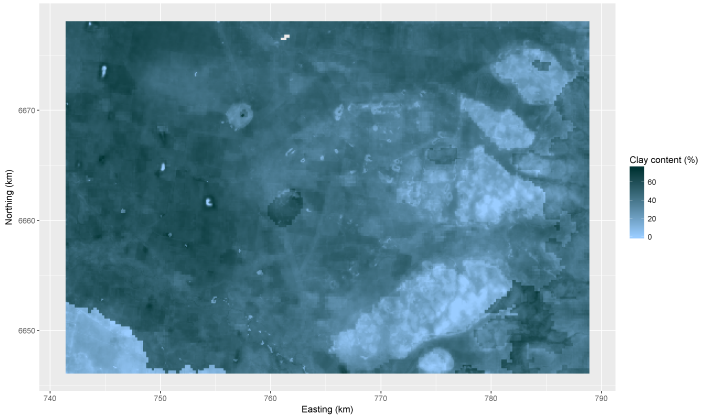
Q5 : *Compare the regression-kriging standard deviationwith the residual standard deviation of the linear model. Can you explain the difference?*
Jump to A5 •

Some predictions are outside the physical range [0,100]. Set predictions outside this range to 0 or 100. Note that this is a pragmatic solution.

```
> # correct predictions
> rk$pred <- ifelse(test = rk$pred < 0, yes = 0, no = rk$pred)
> rk$lower <- ifelse(test = rk$lower < 0, yes = 0, no = rk$lower)
> rk$upper <- ifelse(test = rk$upper < 0, yes = 0, no = rk$upper)
```

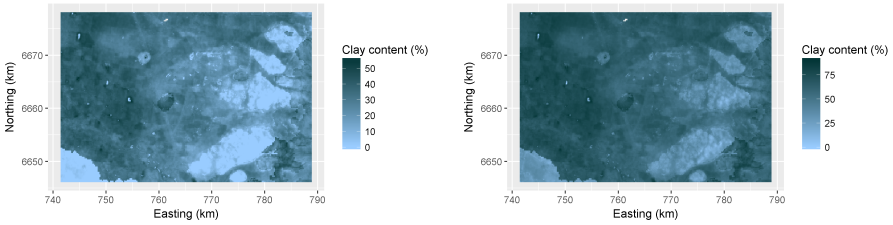
Plot the predictions.

```
> # convert to data.frame
> rk <- as(rk, Class = "data.frame")
>
> # create ggplot object
> gg.rk <- ggplot() + geom_raster(data = rk, mapping = aes(x = x * 0.001, y = y *
+     0.001, fill = pred)) + scale_fill_continuous(name = "Clay content (%)",
+     low = "#99CCFF", high = "#003333") + scale_x_continuous(name = "Easting (km)") +
+     scale_y_continuous(name = "Northing (km)") + coord_equal(ratio = 1)
>
> # plot
> gg.rk
```



Plot the lower and upper boundary of the 90% prediction interval.

```
> # create ggplot object
> gg.lower <- ggplot() + geom_raster(data = rk, mapping = aes(x = x * 0.001,
+     y = y * 0.001, fill = lower)) + scale_fill_continuous(name = "Clay content (%)",
+     low = "#99CCFF", high = "#003333") + scale_x_continuous(name = "Easting (km)") +
+     scale_y_continuous(name = "Northing (km)") + coord_equal(ratio = 1)
>
> # create ggplot object
> gg.upper <- ggplot() + geom_raster(data = rk, mapping = aes(x = x * 0.001,
+     y = y * 0.001, fill = upper)) + scale_fill_continuous(name = "Clay content (%)",
+     low = "#99CCFF", high = "#003333") + scale_x_continuous(name = "Easting (km)") +
+     scale_y_continuous(name = "Northing (km)") + coord_equal(ratio = 1)
>
> # plot
> gg.lower
> gg.upper
```



TASK 12 : Spatial prediction: kriging with external drift •

On contrary to (linear) regression-kriging, kriging with an external drift predicts the trend and residuals simultaneously. It takes into account spatial correlation between observations when estimating the trend coefficients, and the uncertainty associated to these predictions is included in the kriging variance. It is a statistically more elegant approach than regression-kriging.

It is easy to implement kriging with an external drift in `gstat`. The only difference with the regression-kriging code is that the `resid~1` function has to be replaced with the trend model.

```
> ked <- krige(trend.sw, locations = d, newdata = edgeroi.grids, model = vmf,
+     debug.level = -1)

[using universal kriging]

  1% done
  4% done
  7% done
 12% done
 17% done
 22% done
 27% done
 31% done
 35% done
 36% done
 38% done
 40% done
 42% done
 45% done
 48% done
 49% done
 51% done
 54% done
 56% done
 59% done
 63% done
 68% done
 73% done
 78% done
 83% done
 88% done
 93% done
 98% done
100% done
```

Compute the 90% prediction interval, and make sure that the predictions are in the [0,100] range.

```
> # 90% prediction interval
> ked$lower <- ked$var1.pred - 1.645 * sqrt(ked$var1.var)
> ked$upper <- ked$var1.pred + 1.645 * sqrt(ked$var1.var)
```

```
>
> # correct predictions
> ked$var1.pred <- ifelse(test = ked$var1.pred < 0, yes = 0, no = ked$var1.pred)
> ked$lower <- ifelse(test = ked$lower < 0, yes = 0, no = ked$lower)
> ked$upper <- ifelse(test = ked$upper < 0, yes = 0, no = ked$upper)
```

Inspect the output of both prediction methods.

```
> # regression-kriging
> summary(rk[, -c(1:2)])

   var1.pred           var1.var          trend             pred
 Min.   :-9.69986   Min.   : 81.75   Min.   :-51.84   Min.   : 0.00
 1st Qu.:-2.03068   1st Qu.:100.04   1st Qu.: 37.40   1st Qu.:36.51
 Median : 0.05696   Median :101.80   Median : 46.27   Median :45.61
 Mean   :-0.04732   Mean   :101.64   Mean   : 42.82   Mean   :42.78
 3rd Qu.: 1.94265   3rd Qu.:102.66   3rd Qu.: 52.57   3rd Qu.:52.98
 Max.   :11.86253   Max.   :110.54   Max.   : 74.95   Max.   :75.26
     lower            upper
 Min.   : 0.00   Min.   : 0.00
 1st Qu.:19.89   1st Qu.:53.10
 Median :29.04   Median :62.19
 Mean   :26.68   Mean   :59.35
 3rd Qu.:36.44   3rd Qu.:69.56
 Max.   :58.59   Max.   :91.92

> round(head(rk[, -c(1:2)]), 2)

  var1.pred var1.var trend  pred lower upper
1      1.55   101.25 55.12 56.67 40.12 73.22
2      1.62   100.44 54.42 56.04 39.56 72.53
3      1.70    99.58 52.54 54.23 37.82 70.65
4      1.77    98.68 51.74 53.52 37.17 69.86
5      1.85    97.78 51.25 53.10 36.84 69.37
6      1.88    97.51 51.69 53.57 37.33 69.82

> # kriging with an external drift
> summary(ked@data)

   var1.pred         var1.var           lower            upper
 Min.   : 0.00   Min.   :  82.05   Min.   : 0.00   Min.   : 5.943
 1st Qu.:36.91   1st Qu.: 102.06   1st Qu.:19.70   1st Qu.:53.947
 Median :45.57   Median : 104.19   Median :28.65   Median :62.598
 Mean   :42.73   Mean   : 107.37   Mean   :26.27   Mean   :59.728
 3rd Qu.:52.70   3rd Qu.: 107.72   3rd Qu.:35.87   3rd Qu.:69.573
 Max.   :74.09   Max.   :1959.66   Max.   :56.51   Max.   :91.670

> round(head(ked@data), 2)

  var1.pred var1.var lower upper
1     56.39   103.23 39.67 73.10
2     55.78   102.21 39.15 72.41
3     54.11   100.91 37.58 70.63
4     53.46    99.66 37.04 69.88
5     53.04    98.79 36.69 69.39
6     53.54    98.80 37.19 69.89
```

Q6 : *Why do the predictions differ? Can you explain the difference between the RK variance and the KED variance?* *Jump to A6*

•

4 Random forests modelling

This section will illustrate how to fit a random forests model and interpret the results.

TASK 13 : Fit a random forests model •

We start by specifying the model input. The response variable (soil property of interest) has to be stored in a separate vector; the covariate values at the data points have to be stored in a separate `data.frame` or `matrix`. It is not necessary to specify a trend formula, the function will use all variables stored in the covariate `data.frame`. Check the help file on the `randomForest` function of the `randomForest` package on how to fit a random forests model.

```
> # convert data object to data.frame
> d <- as(d, Class = "data.frame")
>
> # copy soil property values to a new vector
> z <- d$z
>
> # copy the covariates to a new data.frame
> names(d)

 [1] "id"      "z"       "MVBSRT6" "TI1LAN6" "TI2LAN6" "PCKGAD6"
 [7] "RUTGAD6" "PCTGAD6" "DEMSRT5" "TWISRT5" "PMTGEO5" "EV1MOD5"
[13] "EV2MOD5" "EV3MOD5" "x"       "y"       "resid"

> covar <- d[, 3:14]
```

Fitting a (default) random forests model is very easy. You can modify the number of trees (default = 500) with the `ntree` argument.

```
> rf <- randomForest(x = covar, y = z)
```

Inspect the output.

```
> str(rf, max.level = 2)

List of 17
 $ call           : language randomForest(x = covar, y = z)
 $ type           : chr "regression"
 $ predicted      : Named num [1:300] 52.4 58.4 55.1 48.3 61.5 ...
  ..- attr(*, "names")= chr [1:300] "1" "2" "3" "5" ...
 $ mse            : num [1:500] 290 225 208 189 170 ...
 $ rsq            : num [1:500] 0.0235 0.2436 0.3014 0.366 0.4281 ...
 $ oob.times      : int [1:300] 176 190 183 183 185 157 179 209 196 197 ...
 $ importance     : num [1:12, 1] 18742 3758 3908 9054 2255 ...
  ..- attr(*, "dimnames")=List of 2
 $ importanceSD   : NULL
 $ localImportance: NULL
 $ proximity      : NULL
 $ ntree          : num 500
 $ mtry           : num 4
 $ forest         :List of 11
  ..$ ndbigtree    : int [1:500] 185 209 203 217 203 187 209 205 207 191 ...
  ..$ nodestatus   : int [1:225, 1:500] -3 -3 -3 -3 -3 -1 -3 -3 -3 -1 ...
  ..$ leftDaughter : int [1:225, 1:500] 2 4 6 8 10 0 12 14 16 0 ...
  ..$ rightDaughter: int [1:225, 1:500] 3 5 7 9 11 0 13 15 17 0 ...
  ..$ nodepred     : num [1:225, 1:500] 44.7 29.2 51.2 44.7 24.2 ...
  ..$ bestvar      : int [1:225, 1:500] 1 11 8 8 1 0 1 12 6 0 ...
  ..$ xbestsplit   : num [1:225, 1:500] 3.565 0.015 19.05 20.1 0.13 ...
  ..$ ncat         : Named int [1:12] 1 1 1 1 1 1 1 1 6 1 ...
  .. ..- attr(*, "names")= chr [1:12] "MVBSRT6" "TI1LAN6" "TI2LAN6" "PCKGAD6" ...
```

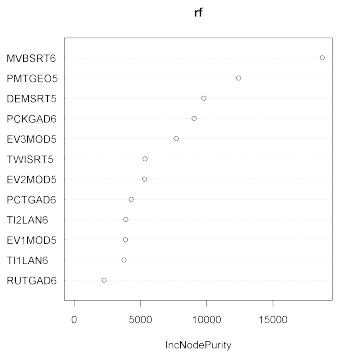
```
  ..$ nrnodes       : int 225
  ..$ ntree         : num 500
  ..$ xlevels       :List of 12
 $ coefs           : NULL
 $ y               : num [1:300] 36 55 71 49.6 62.2 62 58.5 62.5 55.5 37.8 ...
 $ test            : NULL
 $ inbag           : NULL
 - attr(*, "class")= chr "randomForest"
```

You can see that the individual trees are stored. To view a single tree use the `getTree` of the `randomForest` package.

Q7 : *How many trees were fitted? Try to find out how many covariates were randomly selected to be evaluate for splitting a node.* Hint*: This is determined by the `mtry` argument, use `?randomForest` to find out what the default setting is. What is the random forests prediction at the first observation site? How many times was the first observation used to calibrate a tree?* *Jump to A7* •

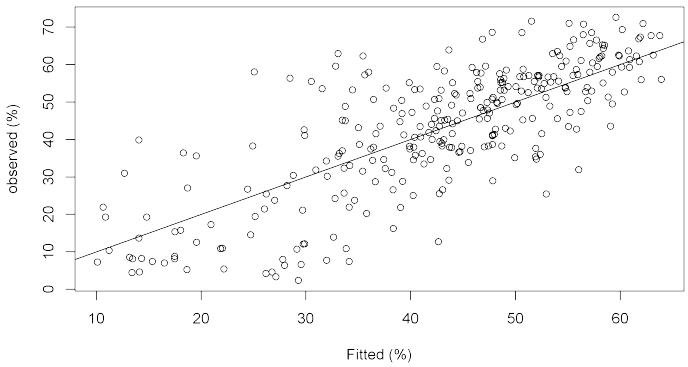
Plot the variable importance.

```
> varImpPlot(rf)
```



Plot predicted versus observed value and compute the R^2 value.

```
> # correlation plot
> plot(rf$predicted, rf$y, xlab = "Fitted (%)", ylab = "observed (%)")
> abline(0, 1)
>
> # r2 value
> cor(rf$predicted, rf$y)^2

[1] 0.600839
```



Q8 : *What can you tell about the performance of the random forest model compared to the linear regression model for predicting topsoil clay content?*
Jump to A8 •

TASK 14 : Predict with the random forests model •

The generic `predict` function can be used to predict at the nodes of the Edgeroi grid with the random forests model. The grid should be offered to the `predict` function as a `data.frame`.

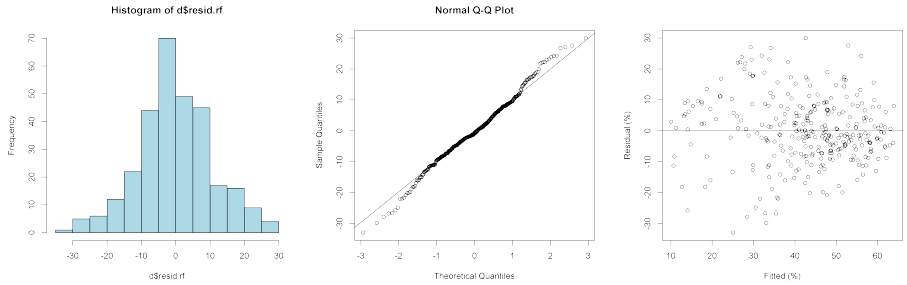
```
> edgeroi.grids <- as(edgeroi.grids, Class = "data.frame")
> p.rf <- predict(rf, newdata = edgeroi.grids)
```

It is also possible to obtain the predictions for each of the trees in the forest by adding the `predict.all = TRUE` argument to the `predict` function.

TASK 15 : Fit variogram to random forests residuals •

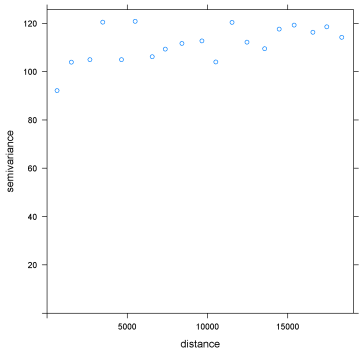
Compute the model residuals and check their distribution.

```
> # compute residual
> d$resid.rf <- rf$predicted - rf$y
>
> # check residual distribution
> hist(d$resid.rf, col = "lightblue")
> qqnorm(d$resid.rf)
> qqline(d$resid.rf)
>
> # residual plot
> plot(rf$predicted, d$resid.rf, xlab = "Fitted (%)", ylab = "Residual (%)")
> abline(0, 0)
```



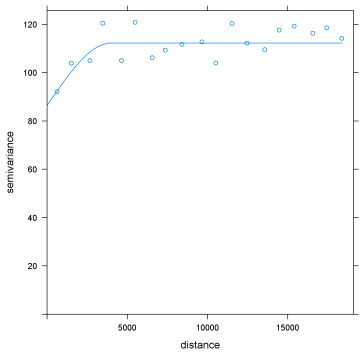
Calculate residual sample variogram.

```
> # convert data.frame to a spatial object
> coordinates(d) <- ~x + y
>
> # set coordinate projection
> proj4string(d) <- CRS(gda94)
>
> # compute sample variogram.
> v.rf <- variogram(d$resid.rf ~ 1, data = d, width = 1000)
> plot(v.rf)
```



Fit a variogram model to the sample variogram.

```
> # define variogram model: initial values for psill, range, nugget based
> # on sample variogram
> vm <- vgm(psill = 30, model = "Sph", range = 5000, nugget = 80)
>
> # fit variogram model to the experimental variogram
> vmf.rf <- fit.variogram(v.rf, model = vm)
>
> # plot variograms
> plot(v.rf, vmf.rf)
```




```
> # inspect variogram parameters
> vmf
```

```
  model    psill    range
1   Nug 72.40077    0.000
2   Sph 37.53745 4171.178

> vmf.rf

  model    psill   range
1   Nug 86.39179    0.00
2   Sph 25.84965 3990.42
```

Q9 : *Compare the variogram parameters of the linear regression residuals and the random forest residuals. What can you conclude?* *Jump to A9* •

TASK 16 : Kriging the random forests residuals •

Like for the linear regression model, we can add a kriging component to the random forests modelling. This is another form or regression-kriging, in this case non-linear or tree-based regression kriging. We can follow the same procedure as for the linear model.

```
> # convert grid data to spatial object and set projection
> gridded(edgeroi.grids) <- ~x + y
> proj4string(edgeroi.grids) <- CRS(gda94)
>
> # krige the residuals
> rf.k <- krige(resid.rf ~ 1, locations = d, newdata = edgeroi.grids, model = vmf.rf,
+     debug.level = -1)
```

Append the trend predictions to the object that holds the kriged residuals. Inspect the results.

```
> rf.k$trend <- p.rf
> str(rf.k@data)

'data.frame': 151975 obs. of  3 variables:
 $ var1.pred: num  -1.18 -1.22 -1.27 -1.32 -1.36 ...
 $ var1.var : num  108 108 108 107 107 ...
 $ trend    : num  46.9 47.5 48.4 54.8 54.9 ...
```

Add the trend predictions to the predicted residuals.

```
> rf.k$pred <- rf.k$trend + rf.k$var1.pred
```

Compute the 90% prediction interval.

```
> rf.k$lower <- rf.k$pred - 1.645 * sqrt(rf.k$var1.var)
> rf.k$upper <- rf.k$pred + 1.645 * sqrt(rf.k$var1.var)
```

Check for predictions outside physical range [0,100]. Note that the random forests model does not predict outside the physical range. Only the lower boundary of the 90% prediction interval has some negative values.

```
> summary(rf.k@data)

   var1.pred          var1.var          trend            pred
 Min.   :-7.1038   Min.   : 95.51   Min.   : 8.358   Min.   : 7.819
 1st Qu.:-1.4960   1st Qu.:107.84   1st Qu.:35.918   1st Qu.:35.623
 Median :-0.3022   Median :108.77   Median :43.902   Median :43.660
 Mean   :-0.2334   Mean   :108.56   Mean   :42.554   Mean   :42.321
 3rd Qu.: 0.9487   3rd Qu.:109.22   3rd Qu.:51.113   3rd Qu.:50.762
 Max.   : 7.1011   Max.   :112.77   Max.   :67.080   Max.   :66.914
     lower            upper
```

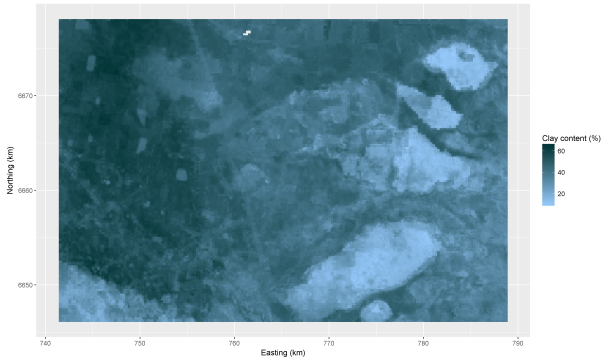
```
 Min.   :-9.347   Min.   :24.79
 1st Qu.:18.418   1st Qu.:52.82
 Median :26.541   Median :60.79
 Mean   :25.182   Mean   :59.46
 3rd Qu.:33.647   3rd Qu.:67.87
 Max.   :49.863   Max.   :83.96
```

Save the map as a GeoTIFF in a new folder that we will create to store output.

```
> # create output folder
> dir.create("./out")
>
> # save as GeoTIFF
> writeGDAL(rf.k["pred"], fname = "./out/Clay_RFK.tif", drivername = "GTiff",
+     type = "Float32")
```

Plot the random forests-kriging map.

```
> # convert to data.frame
> rf.k <- as(rf.k, Class = "data.frame")
>
> # create ggplot object
> gg.rfk <- ggplot() + geom_raster(data = rf.k, mapping = aes(x = x * 0.001,
+     y = y * 0.001, fill = pred)) + scale_fill_continuous(name = "Clay content (%)",
+     low = "#99CCFF", high = "#003333") + scale_x_continuous(name = "Easting (km)") +
+     scale_y_continuous(name = "Northing (km)") + coord_equal(ratio = 1)
>
> # plot
> gg.rfk
```



Q10 : *The map appears to be different from the linear-regression krigin map. How can we best test which of the two maps is the most accurate?*
Jump to A10 •

5 Visualizing spatial data in Google Earth and interactive maps

You can display your spatial data in Google Earth by saving these as 'kml' files. This can be done with the plotKML package. plotKML requires installation of some other softwares. Detailed instructions can be found in the plotKML tutorial. The `kml` function writes the data to a kml file. You can also use `plotKML` for this purpose. The difference is that plotKML directly opens the kml in Google Earth.

Note: when converting a raster object to kml, the `kml` function produces two PNG files ('.png' and '_legend.png') that are used for visualization in

Goolge Earth. When several KML files are generated, as such below, these PNG files are overwritten each time a new KML file is generated. This means that only the visualization of the last KML file of the sequence is stored. When one wants to keep all kml files and their associated legends it is adviced to copy these to a different folder before running the next KML command. Here we show how to copy files from within R. (Of course, the files can also be directly generated in the new folders.)

```
> # create folders to store KML files
> dir.create("./out/kml_rk")
> dir.create("./out/kml_rf")
>
> # define shape of the points in the visualization
> shape <- "http://maps.google.com/mapfiles/kml/pal2/icon18.png"
>
> # export point data
> kml(d, file.name = "./out/edgeroiData.kml", colour = z, balloon = TRUE, shape = shape,
+     colour_scale = SAGA_pal[[1]], points_names = round(d$z, 1))
>
> # export regression-krigin results
> gridded(rk) <- ~x + y
> proj4string(rk) <- CRS(gda94)
> kml(rk["pred"], file.name = "clay_RK.kml", colour_scale = SAGA_pal[[1]])
>
> # copy kml files to separate directory
> system(paste("xcopy", "clay_RK.kml", shortPathName(normalizePath("./out/kml_rk"))))
> system(paste("xcopy", "*.png", shortPathName(normalizePath("./out/kml_rk"))))
>
> # export random forest model results
> gridded(rf.k) <- ~x + y
> proj4string(rf.k) <- CRS(gda94)
> kml(rf.k["pred"], file.name = "clay_RF.kml", colour_scale = SAGA_pal[[1]])
>
> # copy kml files to separate directory
> system(paste("xcopy", "clay_RF.kml", shortPathName(normalizePath("./out/kml_rf"))))
> system(paste("xcopy", "*.png", shortPathName(normalizePath("./out/kml_rf"))))
>
> # delete KML files from working directory
> unlink(c("clay_RK.kml", "clay_RF.kml", ".png", "_legend.png"))
```

Let's now take a look on how to create some interactive maps with the `leaflet` package. We will start with the point data. First the data need to be reprojected to the WGS84 coordinate system.

```
> # convert data to WGS84 coordinate reference system
> d.ll <- d
> d.ll <- spTransform(d.ll, CRSobj = CRS(wgs84))
```

Then we will create a `leaflet` map. We start simple and then add extra functionality.

```
> # basic map
> addMarkers(addTiles(leaflet()), data = d.ll)
>
> # or, more elegant:
> leaflet() %>% addTiles() %>% addMarkers(data = d.ll)
>
> # creat pop-ups for markers
> my_pops <- paste0("<strong>Site: </strong>", d.ll$id, "<br>
   <strong> Clay content (%): </strong>",
+     round(d.ll$z, 1))
>
> # create interactive map
> leaflet() %>% addProviderTiles("Esri.WorldImagery") %>% addMarkers(data = d.ll,
+     popup = my_pops)
> # now click on a marker in the map!
```

```
>
> ## further marker enhancements define colour ramps
> pal1 <- colorQuantile("YlOrBr", domain = d$z)
> pal2 <- colorNumeric(SAGA_pal[[1]], domain = d.ll$z, na.color = "transparent")
>
> # generate interactive maps
> (ll1 <- leaflet() %>% addProviderTiles("Esri.WorldImagery") %>% addCircleMarkers(data = d
+     color = ~pal1(z), popup = my_pops) %>% addLegend("bottomright", pal = pal1,
+     values = d.ll$z, title = "Clay content (%) quantiles", opacity = 0.8))
>
> # with a different color scale
> (ll2 <- leaflet() %>% addProviderTiles("Esri.WorldImagery") %>% addCircleMarkers(data = d
+     color = ~pal2(z), popup = my_pops) %>% addLegend("bottomright", pal = pal2,
+     values = d.ll$z, title = "Clay content (%)", opacity = 0.8))
>
> # save as HTML widget
> saveWidget(ll1, file = paste0(getwd(), "/out/Edgeroi_clay_content.html",
+     sep = ""))
```

Now let's make an interactive map for raster data.

```
> # read tiff as RasterLayer
> r <- raster("./out/Clay_RFK.tif")
>
> # plot aesthetics
> header <- "Clay content (%)"
> pal <- colorNumeric(SAGA_pal[[1]], values(r), na.color = "transparent")
>
> # create leaflet
> (ll <- leaflet() %>% addProviderTiles("Esri.WorldImagery") %>% addRasterImage(r,
+     colors = pal, opacity = 0.5) %>% addLegend(pal = pal, values = values(r),
+     title = header))
>
> # save as widget
> saveWidget(ll, file = paste0(getwd(), "/out/Clay_content_RFK.html", sep = ""))
```

6 Hands-on exercise: model and map the organic carbon content

This section includes a hands-on exercise that, for a large part, you will have to programme yourself based on what you have learned in this tutorial so far. This section also introduces the log-transform for (right-)skewed data. In the code below, you will have to replace the '...' with code you have to programme yourself.

TASK 17 : Derive carbon values and extract covariate data •

Fit a mass-preserving spline to the carbon profiles of the Edgeroi dataset and compile a new `data.frame` with the carbon data.

```
> # fit spline
> mps <- mpspline(edgeroi.spc, var.name = "ORCDRC", vlow = 0, vhigh = 1000)
>
> # compile a new data.frame with carbon data
> d <- data.frame(id = mps$idcol, x = edgeroi.spc@sp@coords[, 1], y = edgeroi.spc@sp@coords
+     2], z = mps$var.std[, 2])
>
> # set projection and reproject
> coordinates(d) <- ~x + y
> proj4string(d) <- CRS(grs80)
> d <- spTransform(d, CRS(gda94))
>
> # identify points with similar coordinates (gives problems with spatial
> # modelling)
```

```
> zerodist(d, zero = 0, unique.ID = FALSE)
>
> # remove duplicates
> d <- remove.duplicates(d, zero = 0, remove.second = TRUE)
```

Extract covariate data through a spatial overlay.

```
> # overlay
> dum <- over(x = ..., y = ...)
>
> # append covariate data to
> d@data <- cbind(d@data, dum)
>
> # convert to data.frame
> d <- as(d, Class = "data.frame")
>
> # remove observations without covariate data, clay data or coordinates
> d <- na.omit(d)
>
> # clean-up
> rm(dum)
```

TASK 18 : Fit a linear regression model and assess model diagnostics •

Fit a linear regression model to the carbon data. Apply stepwise selection. Inspect the distribution of residuals and decide if a log-transform is required to normalize the residuals and stabalize the variance. You will now programme some code yourself for model selection and inspection of the model residuals.

```
> # define trend model
> names(d)
> trend <- as.formula(paste("z~", paste(names(d)[3:14], collapse = "+")))
>
> # fit lm model
> oc.lm <- ...
>
> # stepwise selection
> oc.slm <- ...
>
> # inspect model
> summary(oc.slm)
```

Inspect the residual diagnostics. Compute the skewness coefficient, plot histograms, QQplot and residual plot.

```
> # skewness statistics
> skewness(...)
>
> # histogram
> hist(...)
>
> # qqplot
> qqnorm(...)
> qqline(...)
>
> # residual plot
> plot(..., ...)
> abline(0, 0)
```

Q11 : *Do the residuals follow the assumptions of having a normal distribution and constant variance? Should we log-transform the carbon data?*

Jump to A11 •

Based on the residual diagnostics, transformation might not be necessary. However, for illustrative purposes we will transform the carbon contents before fitting the linear regression model.

TASK 19 : Fit a linear regression model on log-transformed data •

Transform the carbon data to natural logarithms, this can be done with the `log()` function, and fit a linear regression model.

```
> # log transform
> d$lz <- log(d$z)  # use log1p when the data contains zeros
>
> # define trend model
> names(d)
> trend <- as.formula(paste("lz~", paste(names(d)[3:14], collapse = "+")))
>
> # fit lm model
> loc.lm <- ...
>
> # stepwise selection of covariates
> loc.slm <- ...
>
> # inspect model
> summary(loc.slm)
```

Q12 : *How much of the variation in the data is explained by the model?*
Jump to A12 •

Inspect the residual diagnostics of the log-transformed data. Compute the skewness coefficient, plot histograms, QQplot and residual plot.

```
> # skewness statistics
> skewness(...)
>
> # histogram
> hist(...)
>
> # qqplot
> qqnorm(...)
> qqline(...)
>
> # residual plot
> plot(..., ...)
> abline(0, 0)
```

Q13 : *Do the residuals on log-scale follow the assumptions of having a normal distribution and constant variance?*
Jump to A13 •

```
> # copy residuals to data object
> d$lresid <- ...
>
> # convert data.frame to a spatial object
> coordinates(d) <- ...
>
> # set coordinate projection
> proj4string(d) <- ...
>
> # compute sample variogram (use lresid~1 as function, use width=1000)
```

```
> v.oc <- ...
> plot(v.oc)
>
> # define variogram model (use a spherical model) choose psill and nugget
> # values on basis of the sample variogram
> vm <- ...
>
> # fit variogram model to the experimental variogram
> vmf.oc <- ...
>
> # plot variograms
> plot(v.oc, vmf.oc)
```

Q14 : *Do the residuals show spatial correlation? How should we proceed with the analysis?* *Jump to A14* •

Unfortunately, the residuals do not show any spatial correlation. The fitted variogram is practially flat (pure nugget). Sometimes it helps of to try different values for the `width` and `cutoff` when computing a sample variogram, or to compute a robust variogram by setting the argument `cressie = TRUE`; see the help file for the `variogram` function.

The `fit.variogram` function gave a warning that the function did not converge; another frequently occuring warning is `'singular model'`. It gives these warnings when the function encounters problems fitting the variogram. Often this means that there is no unique analytical solution for the parameter estimates. Despite this warning, the function does give output. In that case the parameter estimates should be heeded with caution since these are unreliable and should therefore preferably not be used for kriging. A solution can be to fix the range or sill parameter based on the sample variogram, to try a different variogram model, or to try a different fit method by setting the `fit.method` argument (e.g. `fit.method=2`) (see the help file).

TASK 20 : Predict the carbon content •

Use the linear regression model to predict the carbon content in g/kg at the nodes of the Edgeroi prediction grid. For back-transformation we need the prediction error variance. This can be obtained by setting the argument `se.fit = TRUE` in the `predict` function.

```
> p.oc <- ...
```

The prediction error variance is made up of two components: the variance of the prediction and the residual variance. Note that the `predict` function gives the standard error of the prediction and residual standard deviation. These should be squared first to obtain variances, and then summed.

```
> # convert to data.frame
> p.oc <- as.data.frame(p.oc)
>
> # square and sum the prediction se (se.fit) and residual sd
> # (residual.scale)
> p.oc$var <- ... + ...
```

Back-transform the predictions to the original scale by adding half of the prediction error variance to the predicted values before taking the exponent (`exp()`). Hint: a similar function is used to back-transform kriging predictions: `exp(kriging prediction + 0.5*kriging variance)`.

```
> p.oc$pred <- ...
```

Inspect the predictions; compare with observed values.

```
> # predicted carbon
> summary(p.oc$pred)
>
> # observed carbon
> summary(d$z)
>
> # predicted carbon (log-scale)
> summary(p.oc$fit)
>
> # observed carbon (log-scale)
> summary(d$lz)
```

The results show that the linear regression model predicts some extreme carbon contents (up to 330 g/kg) whereas the maximum observed content is 35.6 g/kg. This might be caused by extrapolating the linear relationship with the predictors outside the observed predictor feature space. Another reason might be inflated estimates of the prediction error variance.

Inspect the prediction error variance.

```
> # residual variance
> summary(p.oc$residual.scale^2)
>
> # variance of the estimated vale (model fit)
> summary(p.oc$se.fit^2)
>
> # prediction error variance
> summary(p.oc$var)
```

Summary statistics of the prediction error variances show that there are some extreme values (mean = 0.1573455, while the maximum is 2.1945486). As the results show, the extreme values come from the uncertainty associated to the estimated value (se.fit) These might lead to inflated estimates of the predicted value after back-transformation. It is up to the modeller to decide whether the back-transformed values are realistic or not. If these are not realistic, then a pragmatic solution would be to only use the residual variance (`residual.scale`) for back-transformation. Ignoring the variance of the predictions will only have a very small effect on the back-transformed values since this variance is in general much smaller than the residual variance (except for the extreme values of course, but these are where one might to get rid off). The dependence of the back-transformed value on the prediction error variance is one of the disadvantages of using the log-transform.

Finally, compute the 90% prediction interval boundaries on the original scale. This can be simly done by taking the exponent of the lower and upper boundary on the log-scale [8]. The variance is not added to the fitted values.

```
> p.oc$lower <- ...
> p.oc$upper <- ...
```

7 Answers

A1 : *The statistics and plots show that the clay data are mildly skewed (to the left). Note, however, that the* model residuals *should be normally distributed, not the original data. So, before we decide to transform the data we will first check the residuals of the linear regression model fitted to the original data. Furthermore, since the data are skewed to the left, log-transformation will make the distribution worse.* *Return to Q1* •

A2 : *No, 3 covariates are not significant at the 0.1 level. The AIC of the model is 2316.4* *Return to Q2* •

A3 : *The AIC of the model is 2312.5. The AIC is smaller than the AIC of the full model which indicates that this model is preferred over the full model. The model explains 58.7% of the variation (R2 value).* *Return to Q3* •

A4 : *The skewness statistic, histogram and QQplot show that the residuals approximately follow a normal distribution. The plot showing the fitted versus residual values does not give evidence of heterescedacity. Based on these plots and statistics, there is no need to transform the data. The assumption of linearity between the dependent and independent variables has not been checked.* *Return to Q4* •

A5 : *The regression-kriging standard deviation is smaller than the linear model standard deviation. A part of the residual variance is spatially structured that is accounted for with kriging. Furthermore, the linear model residual standard deviation is constant (homoscedacity) while the RK standard deviation is not. It depends on the distance between prediciton location and observation locations. The closer a prediction location is to an observation location, the smaller the RK variance is. Note that the RK variance does not include the uncertainty associated to the trend predictions since the trend and the residuals are modelled separtely. To account for this uncertainty in the RK variance, one should apply 'kriging with external drift'/'universal kriging'.* *Return to Q5* •

A6 : *The predictions slightly differ because the KED trend estimates are different from the RK trend estimates; the former are estimated with generalized least squares, the later with ordinary least squares. The KED variance is on aveage larger than the RK variance because the uncertainty associated to the trend estimates is accounted for by the KED variance.* *Return to Q6* •

A7 : *This forest contains 500 trees, this is the default.*
For this forest, 4 covariates were evaluated for each split; the default in case of regression is the number of covariates divided by 3. Here we have 12 covariates.
The random forests prediction at the first observation site is 52.4%.
The `oob.times` *component indicates for each observation how many times it was used to predict the target value at the observation site, i.e. how many times it was*

'out-of-bag'. For the first observation this was 176 times. This means that this observation was used to calibrate 500-176=324 trees. Note that this value can differ a little bit from what you find, depending on the random seed. *Return to Q7* •

A8 : *The plot and R^2 show that the random forests model performs as well as the linear regression model. We should remark, however, is that the random forests R^2 is computed from the OOB predictions (thus based on cross-validation; observations were not used to fit the model) whereas the linear regression R^2 is based on internal validation (observations were used to fit the model). Internal validation, therefore, often gives a more optimistic indication of model accuracy.* *Return to Q8* •

A9 : *The random forest (RF) model is better able to capture the spatial structure in the data. Both the range of spatial correlation and the [partial sill/sill] ratio of the RF model residuals is smaller than those of the linear model residuals. Random forests [psill/sill] = 0.23; Linear model [psill/sill]= 0.341.* *Return to Q9* •

A10 : *This can best be determined through validation. Validation means putting the predictions to the test by comparing predicted values with observed, independent, data. This means data that is not used to calibrate the prediction models. This can best be done with independent data collected with a probability (random) sampling design. Other validation methods include data-splitting or cross-validation. Brus et al. [5] give an overview of validation methods for soil maps.* *Return to Q10* •

A11 : *The residuals are still mildly skewed. The skewness coefficient should preferably be smaller than 0.5, though this choice is a bit arbitrary. The plot showing the fitted versus residual values does not give a strong evidence for heterescedacity, though the point cloud seems to widen a bit with increasing fitted values. Based on these plots and statistics, log-transformation might not be necessary.* *Return to Q11* •

A12 : *The model explains 29.7% of the variation (R2 value).* *Return to Q12* •

A13 : *The residuals approximate a normal distribution. The histogram and QQ-plot show that one residual is a bit of an outlier. The observations belonging to this residual needs further scrutiny. The residual plot does not give an indication of heteroscedacity.* *Return to Q13* •

A14 : *The variogram is a flat line (pure nugget variogram), this means that there is no spatial correlation in the residuals. This can happen! This means that it is of no use to extend the regression model with a kriging step since kriging will only interpolate random noise. We will predict the carbon content using the trend model only.* *Return to Q14* •

8 Code solutions

Hands-on exercise.

```
> # overlay
> dum <- over(x = d, y = edgeroi.grids)
```

```
> # fit lm model
> oc.lm <- lm(trend, data = d)
>
> # stepwise selection
> oc.slm <- step(oc.lm)
```

```
> # skewness statistics (e1071 package)
> skewness(resid(oc.slm))
>
> # histogram
> hist(resid(oc.slm))
>
> # qqplot
> qqnorm(resid(oc.slm))
> qqline(resid(oc.slm))
>
> # residual plot
> plot(oc.slm$fitted.values, resid(oc.slm), xlab = "Fitted (g/kg)", ylab = "Residual (g/kg)
```

```
> # fit lm model
> loc.lm <- lm(trend, data = d)
>
> # stepwise selection of covariates
> loc.slm <- step(loc.lm)
```

```
> # skewness statistics (e1071 package)
> skewness(resid(loc.slm))
>
> # histogram
> hist(resid(loc.slm))
>
> # qqplot
> qqnorm(resid(loc.slm))
> qqline(resid(loc.slm))
>
> # residual plot
> plot(loc.slm$fitted.values, resid(loc.slm), xlab = "Fitted (g/kg)", ylab = "Residual (g/k
```

```
> # copy residuals to data object
> d$lresid <- resid(loc.slm)
>
> # convert data.frame to a spatial object
> coordinates(d) <- ~x + y
>
> # set coordinate projection
> proj4string(d) <- CRS(gda94)
>
> # compute sample variogram (use lresid~1 as function, use width=1000)
> v.oc <- variogram(lresid ~ 1, data = d, width = 1000)
> plot(v.oc)
>
> # define variogram model (use a spherical model) choose psill and nugget
> # values on basis of the sample variogram
> vm <- vgm(psill = 0.02, model = "Sph", range = 5000, nugget = 0.13)
>
> # fit variogram model to the experimental variogram
> vmf.oc <- fit.variogram(v.oc, model = vm)
```

```
> p.oc <- predict(loc.slm, newdata = edgeroi.grids, se.fit = TRUE)
```

```
> # square and sum the prediction se (se.fit) and residual sd
> # (residual.scale)
> p.oc$var <- p.oc$se.fit^2 + p.oc$residual.scale^2
```

```
> p.oc$pred <- exp(p.oc$fit + 0.5 * p.oc$var)
```

```
> p.oc$lower <- exp(p.oc$fit - 1.645 * sqrt(p.oc$var))
> p.oc$upper <- exp(p.oc$fit + 1.645 * sqrt(p.oc$var))
```

References

[1] D. Beaudette and P. Roudier. *aqp: Algorithms for Quantitative Pedology*, 2015. URL http://r-forge.r-project.org/projects/aqp/. R package version 1.8-6. 1

[2] D. E. Beaudette, P. Roudier, and A. T. O'Geen. Algorithms for quantitative pedology: A toolkit for soil scientists. *Computers & Geosciences*, 52:258 – 268, 2013. doi: 10.1016/j.cageo.2012.10.020. 1

[3] R. Bivand, T. Keitt, and B. Rowlingson. *rgdal: Bindings for the Geospatial Data Abstraction Library*, 2013. URL http://CRAN.R-project.org/package=rgdal. R package version 0.8-6. 1

[4] R. S. Bivand, E. J. Pebesma, and V. Gómez-Rubio. *Applied Spatial Data Analysis with R*. Springer, NY, 2008. URL http://www.asdar-book.org/. 1

[5] D. J. Brus, B. Kempen, and G. B. M. Heuvelink. Sampling for validation of digital soil maps. *European Journal of Soil Science*, 62(3):394–407, 2011. doi: 10.1111/j.1365-2389.2011.01364.x. 31

[6] F. Carré, Alex B. McBratney, and B. Minasny. Estimation and potential improvement of the quality of legacy soil samples for digital soil mapping. *Geoderma*, 141(1-2):1–14, 2007. doi: 10.1016/j.geoderma.2007.01.018. 1

[7] D. E. Knuth. Literate programming. *The Computer Journal*, 27(2): 97–111, 1984. URL http://www.literateprogramming.com/knuthweb.pdf. 1

[8] R.M. Lark and D.J. Lapworth. Quality measures for soil surveys by lognormal kriging. *Geoderma*, 173 - 174(0):231 – 240, 2012. ISSN 0016-7061. doi: 10.1016/j.geoderma.2011.12.008. 29

[9] A. Liaw. *randomForest: Breiman and Cutler's random forests for classification and regression*, 2014. URL http://CRAN.R-project.org/package=randomForest. R package version 4.6-10. 1

[10] B. P. Malone, A. B. McBratney, B. Minasny, and G. M. Laslett. Mapping continuous depth functions of soil carbon storage and available water capacity. *Geoderma*, 154(1-2):138–152, 2009. doi: http://dx.doi.org/10.1016/j.geoderma.2009.10.007. 3

[11] E. Pebesma and B. Graeler. *gstat: Spatial and Spatio-Temporal Geostatistical Modelling, Prediction and Simulation*, 2015. URL http://CRAN.R-project.org/package=gstat. R package version 1.0-22. 1

[12] E. J. Pebesma. Multivariable geostatistics in S: the gstat package. *Computers & Geosciences*, 30:683–691, 2004. 1

[13] E. J. Pebesma and R. S. Bivand. *Classes and methods for spatial data in R. R News 5 (2)*. R News 5 (2), 2005. URL http://cran.r-project.org/doc/Rnews. 1

[14] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015. URL http://www.R-project.org. 1

[15] C. Strobl, J. Malley, and G. Tutz. An introduction to recursive partitioning: Rationale, application, and characteristics of classification and regression trees, bagging, and random forests. *Psychological Methods*, 14(4):323–348, 2009. doi: 10.1037/a0016973. cited By (since 1996) 31. 1

[16] H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Use R. Springer, NY, New York, 2009. URL http://had.co.nz/ggplot2/book. 1

[17] H. Wickham. The Split-Apply-Combine Strategy for Data Analysis. *Journal of Statistical Software*, 40(1):1–29, 2011. URL http://www.jstatsoft.org/v40/i01/. 1

[18] Y. Xie. *knitr: Elegant, flexible and fast dynamic report generation with R*, 2013. URL http://yihui.name/knitr/. R package version 1.1. 1

Index of R Concepts