# Random Forest modelling for DSM

*Bas Kempen*

*Version 1.1, 23 May 2018*





This tutorial shows how to fit a random forest model for modelling the soil organic carbon content in Macedonia using a large soil sample data set and a stack gridded of covariate layers. The random forest model is subsequently applied to map the soil organic carbon content across Macedonia. Finally the tutorial shows how to visualize and export the map.

## 0 Setting up an R session

Before starting scripting, it is recommended to empty the computer memory, clean up the workspace and load the required libraries.

```
# empty memory and workspace
gc()
rm(list=ls())

# load libraries
require(sp)
require(raster)
require(rgdal)
require(randomForest)
require(leaflet)
require(plotKML)
require(htmlwidgets)
```

Also, we need to set a random seed to make our results reproducible. Remember, the random forest algorithm randomly boostraps sample data and randomly selects covariates for splitting. This means that each time you fit a random forest model on your data, the results will be (slightly) different. To make results reproducible we need to set a so-called random seed with the **set.seed()** function. The number used for this can be any number. In this way we ensure that we always get the same result when we run this code.

```r
# set random seed
set.seed(20180523)
```

# 1 Fitting a Random Forest model

## 1.1 Read and process the input data

We start by reading the input data: the soil sample data with associated covariate values (the regression matrix) and the gridded covariate stack. These data have already been prepared and are available with this tutorial. These can be used right away for modelling. In this tutorial we will fit a random forest model fo the soil organic carbon content (SOC).

```r
# read profile and horizon data
load("regressionMatrix.rda")
load("covariateStack.rda")
```

Before we start modelling we must check if there are sampling sites with missing covariate values. We can use the **`complete.cases()`** function for this purpose.

```r
# check completeness
summary(complete.cases(rm))
```

The output shows that 9 sampling sites have incomplete covariate data. We will remove these from the dataset. And check the SOC values.

```r
# check if incomplete observations
rm[!complete.cases(rm),]

# exclude incomplete observations
rm <- rm[complete.cases(rm),]

# summary stats
summary(rm$SOC)
```

We see that there are seveal sampling sites with a topsoil SOC content of 0%. Here we assume that 0 represents `NoData` and will remove these sampling sites from the dataset. (But normally this should be checked with the data provider first!)

```r
# check if incomplete observations
rm <- rm[rm$SOC!=0,]
```

Our input dat is now ready to be used.

We use the **randomForest()** function of the **randomForest** package for fitting a random forest model. This function requires the response variable (the soil property we want to model) to be stored in a separate vector, while the covariate values at the sampling sites have to be stored in a separate `data.frame` or `matrix`. By default, the model will use all covariates for stored in the covariate `data.frame`.

```r
# check the column name of the soil property we want to model
names(rm) # it is 'SOC'
```

```
##  [1] "ProfID"     "SOC"        "X_coord"    "Y_coord"    "B02CHE3"
##  [6] "B04CHE3"    "B07CHE3"    "B13CHE3"    "B14CHE3"    "BARL10"
## [11] "CMCF5avg"   "CRDMRG5"    "CRUMRG5"    "CRVMRG5"    "DEMENV5"
## [16] "DV2MRG5"    "DVMMRG5"    "ENTENV3"    "ESMOD5avg"  "EVEENV3"
## [21] "EXMOD5avg"  "F01USG5"    "F02USG5"    "F04USG5"    "F05USG5"
```

```
## [26] "F06USG5"    "F07USG5"    "IMOD4avg"   "MANMCF5"    "MAXENV3"
## [31] "MMOD4avg"   "MRNMRG5"    "NEGMRG5"    "NIRL00"     "NIRL14"
## [36] "NMOD3avg"   "NMSD3avg"   "PCHE3avg"   "POSMRG5"    "PRSCHE3"
## [41] "RANENV3"    "REDL00"     "REDL14"     "SESA4avg"   "SLPMRG5"
## [46] "SW1L00"     "SW1L14"     "SW2L00"     "SW2L14"     "TMDMOD3"
## [51] "TMNMOD3"    "TPIMRG5"    "TWIMRG5"    "VBFMRG5"    "VDPMRG5"
## [56] "VW1MOD1avg" "LCEE1010"   "LCEE1060"
```

```
# copy soil property values to a new vector
d <- rm$SOC

# and detrmine in which columns the covariate values are stored (column numbers)
# copy the covariates to a new data.frame
covar <- rm[,5:58]
```

## 1.2 Fitting a random forest model

Check the help file of the `randomForest` function by typing `?randomForest` in the console. The `Default S3 method` shows the arguments that of function.

The only two mandatory arguments are `x` and `y`, in which `x` is a `data.frame` storing the covariate values (the predictors) at the sampling sites, and `y` is a vector with the observed data. We proceed by fitting a random forest model in its most simple format.

Model fitting is controlled by two important parameters: **`ntree`** and **`mtry`**. It is not mandatory to specify these. `R` will use default values. Try to find out what these default values are using the help file.

We will not fit the random forest model on the Macedonia soil sample data. Note that fitting the model takses a little while so be patient!

```
# fit the model
rf <- randomForest(x = covar, y = d)
```

## 1.3 Inspecting model output and variable importance

Let's take a look at the model output.

```
# inspect the output
str(rf, max.level=2)
```

Inspect the output and try to understand what all the output arguments mean. Again, check the help file of the `randomForest` function and look under heading **`Value`**, which specifies the output arguments, to help interpretation. What are the values of the **`ntree`** and **`mtry`** parameters, do these match the default values? We refer to Strobl, Malley, and Tutz (2009) for more details on random forest modelling.

The output shows that the individual trees are stored. To view a single tree use the **`getTree()`** function.

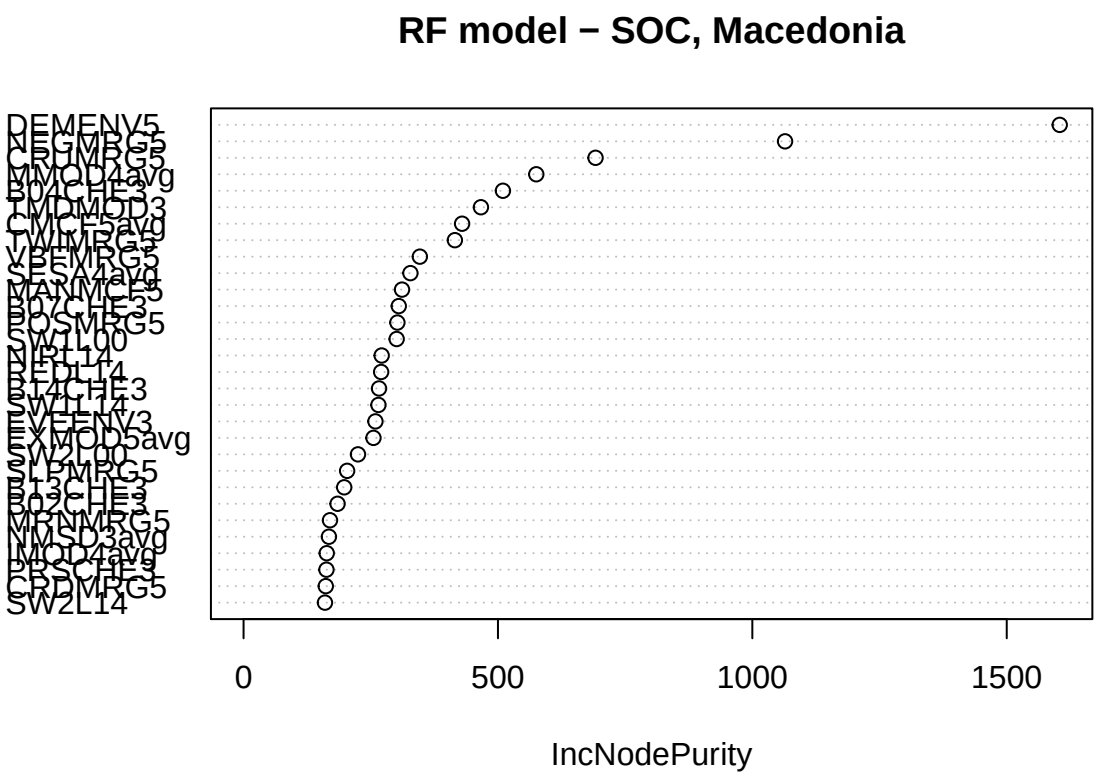
```
# inspect the top of the first tree in the forest.
head(getTree(rf, k=1, labelVar = TRUE))
```

```
##   left daughter right daughter split var split point status prediction
## 1             2              3   DEMENV5  1008.50000     -3   1.870454
## 2             4              5   NEGMRG5  1577.50000     -3   1.558551
## 3             6              7   B07CHE3    40.85866     -3   3.847219
## 4             8              9    SW1L14    75.50000     -3   1.482840
## 5            10             11 EXMOD5avg  4058.75000     -3   6.107660
## 6            12             13   EVEENV3  9751.50000     -3   4.976961
```

The output is a `data.frame` that shows how each node of the tree is split, which variable is used for splitting and the split point, and it gives a predction for each node.

Now plot the variable importance.

```
varImpPlot(rf, main = "RF model - SOC, Macedonia")
```

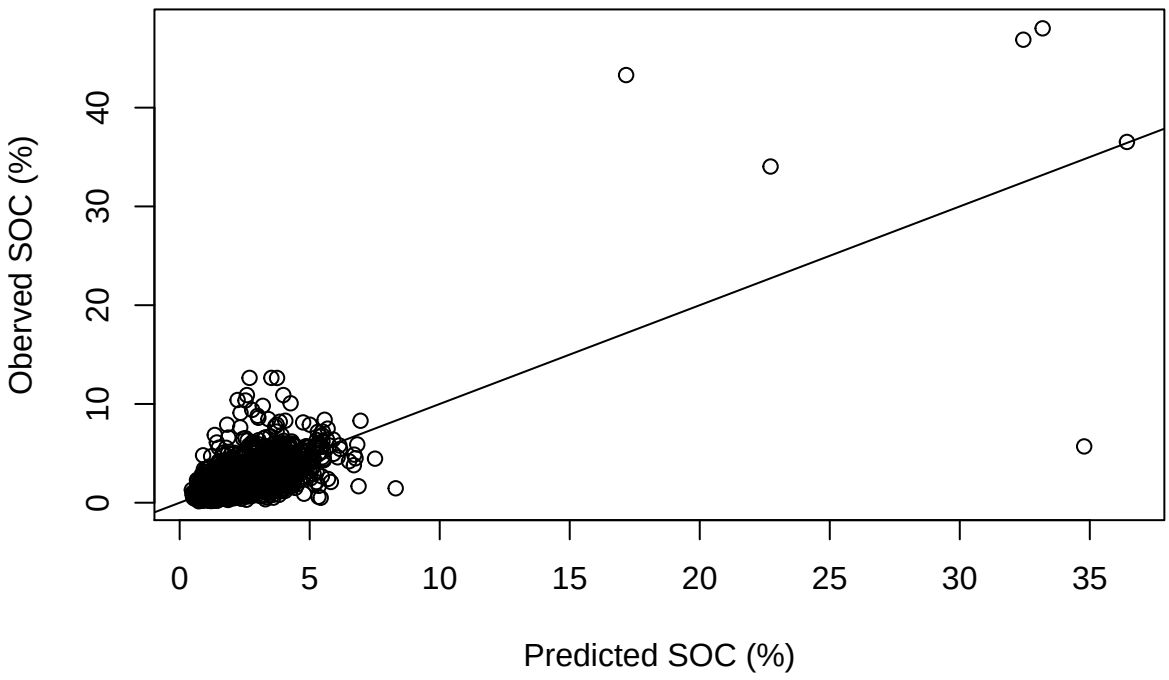



The plot shows that the most important variables (i.e. the strongest predictors) were elevation (DEMENV5) and the negative topographic openess of the landscape (NEGMRG5).

## 2 Accessing goodness-of-fit

Once we have fitted the model, we would like to assess how well the model fits the data. The random forest model output comes with some easily accessible goodness-of-fit statistics. A good way to start is to plot the (Out-Of-Bag; OOB) predicted value versus the observed value (stored in attribute `y`). Let's start by making a scatterplot that shows the observed versus the predicted value and add the 1:1 line to this plot.

```
plot(rf$predicted, rf$y, xlab="Predicted SOC (%)", ylab="Oberved SOC (%)")
abline(0,1)
```




The plot shows that there is some correlation between observed and predicted values, but it also shows that the large SOC contents (indicating organic soils) are not well predicted: there is a strong deviation from the 1:1 line. The model fit could be approved by modelling the log-transformed SOC content, or by fitting a separate model for organic soils (of course there should be enough data available to fit such model, which is not the case here).

The random forest model output contains the OOB **mean squared error (MSE)**, which is a measure of accuracy, and the $\boldsymbol{R^2}$, which indicates which fraction of the variation in the data is explained by the model (it is a variance reduction measure). These can be easily accessed. From the MSE, the **root mean squared error (RMSE)** can be computed, which gives an indication of the error the model makes on average.

```
# MSE
round(rf$mse[rf$ntree],1)
```

```
## [1] 1.7
```

```
# r-squared
round(rf$rsq[rf$ntree],2)
```

```
## [1] 0.6
```

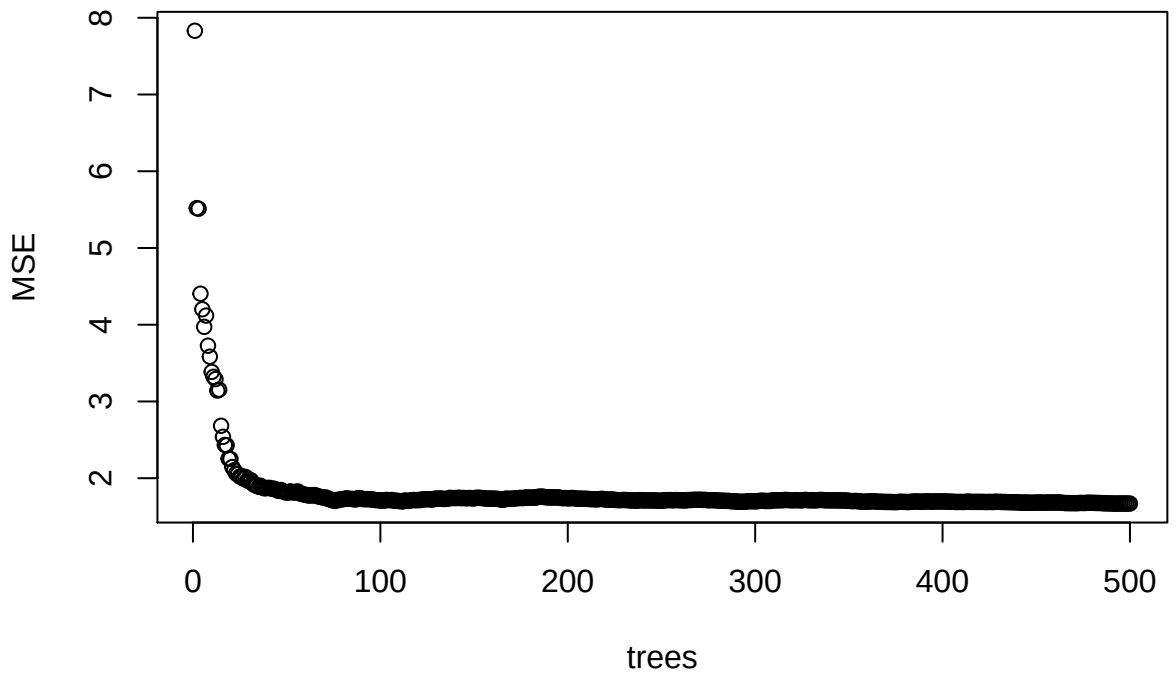
```
# RMSE
round(sqrt(rf$mse[rf$ntree]),1)
```

```
## [1] 1.3
```

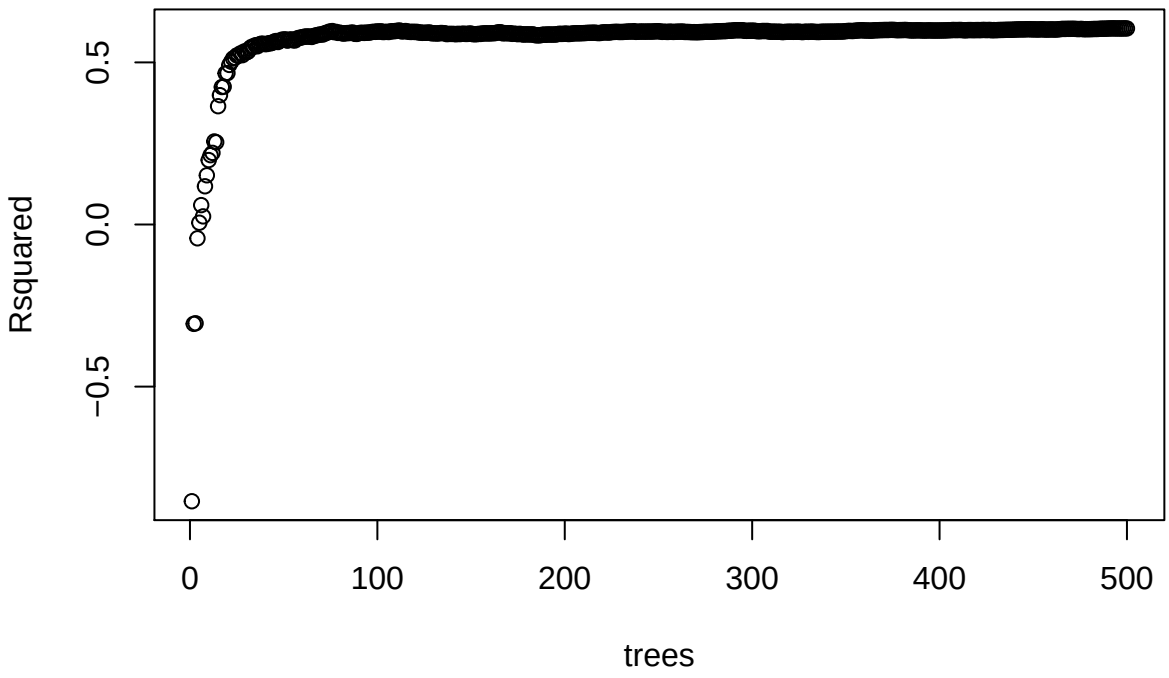
Note that the model stores the MSE and $R^2$ cumulatively for each tree in the forest. Thus to obtain these statistics for the random forest model in general we must use the values calculated from all trees. This means here we have to access the $500^{\text{th}}$ value, hence the use of `rf$ntree` which returns the number of trees in the forest. Here 500.

We can plot the cumulative MSE and $R^2$ values.

```
plot(rf$mse, xlab = "trees", ylab = "MSE")
```




```
plot(rf$rsq, xlab = "trees", ylab = "Rsquared")
```




These plots show that it takes a while for the model to stabalize. One must ensure that the forest is large enough to get stable results. A rule-of-thumb is to have at least 500 trees in a forest.

# 3 Spatial Prediction

## 3.1 Predict using the covariate stack

The generic `predict` function can be used to predict at the nodes of the Macedonia grid with the random forests model. The grid should be offered to the `predict` function as a `data.frame`. Again we must ensure there are no `NoData` values at the prediction locations.

```
# exclude incomplete observations
r <- r[complete.cases(r),]
```

Now predict and inspect output, e.g. check if we do not predict negative values.

```
# predict
p.temp <- predict(rf, newdata = r)

# inspect
str(p.temp)
summary(p.temp)
```

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

### 3.2 Compile predictions

The predict function only returns a vector with the predicted values. These should be combined with the coordinates in order to be able to make a map.

```
# check the names of the columns storing the coordinates
names(r)

# compile predictions
p <- data.frame(x = r$s1, y = r$s2, SOC = p.temp)
head(p)
```

Now we convert the `data.frame` to a `SpatialPixelsDataFrame` (**sp** package) and then to a `RasterLayer` (**raster** package) to generate a map. The map is projected in a local projection Macedonia State Coordinate System zone 7. The `EPSG` code for this coordinate system is 6316.

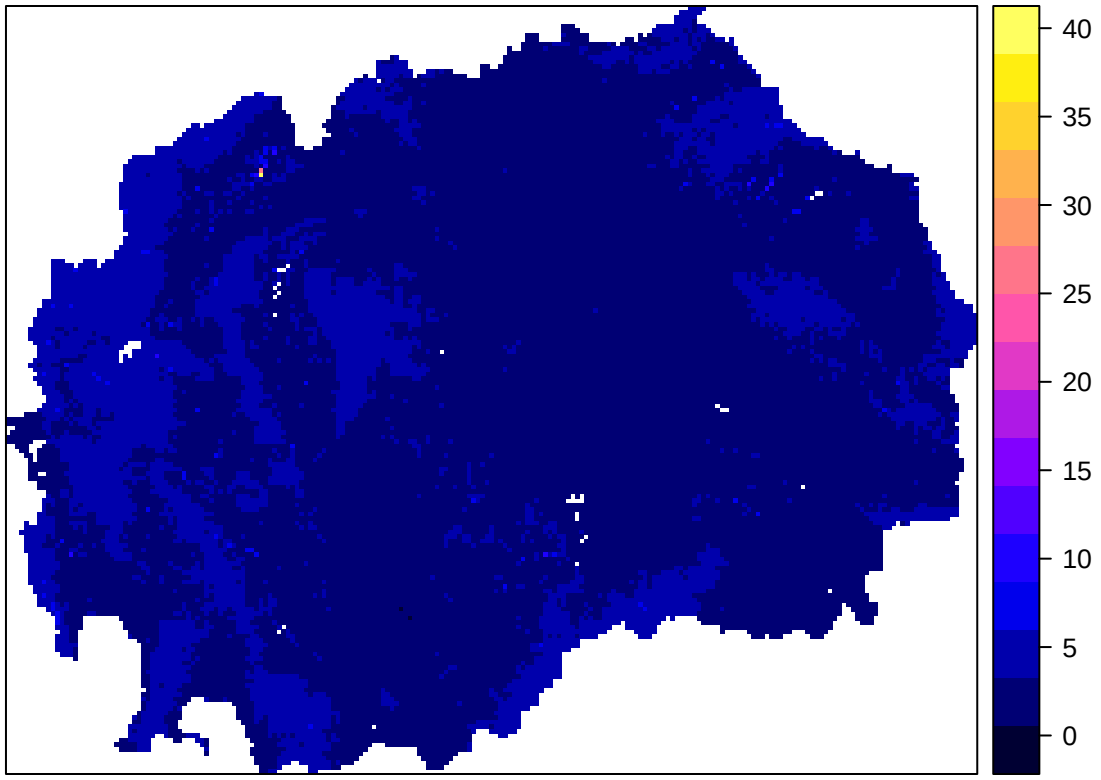
```
# make spatial and project
gridded(p) <- ~x+y
proj4string(p) <- CRS("+init=epsg:6316")

# convert to a RasteLayer object.
p.r <- raster(p)
```

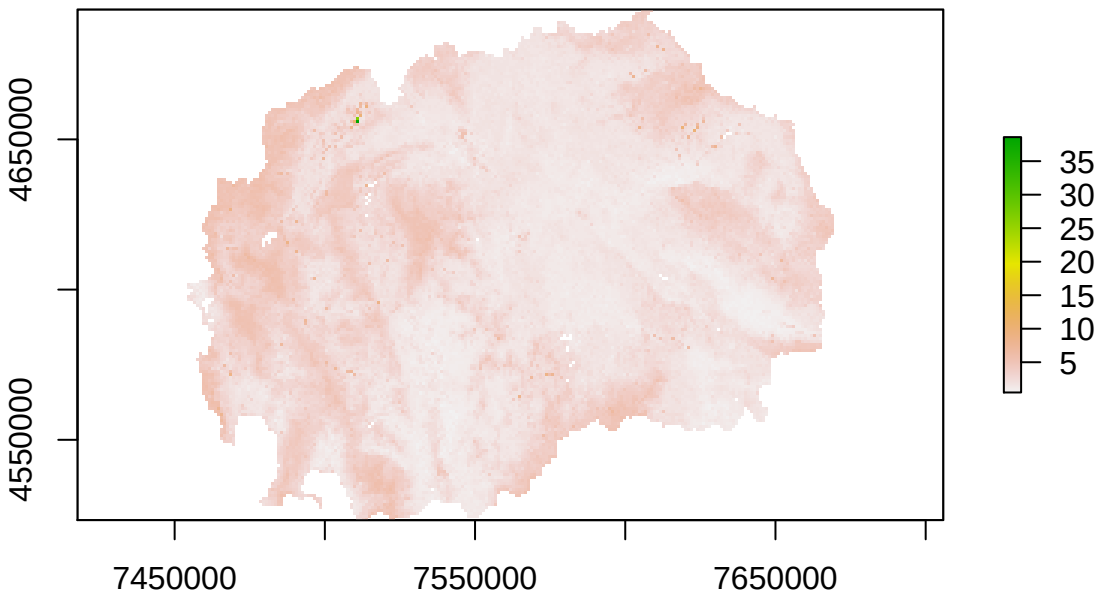
### 3.3 SOC content maps

Now that we have organized our SOC predictions in a spatial object we can generate maps.

```
# plot with the sp package
spplot(p, zcol = "SOC")
```

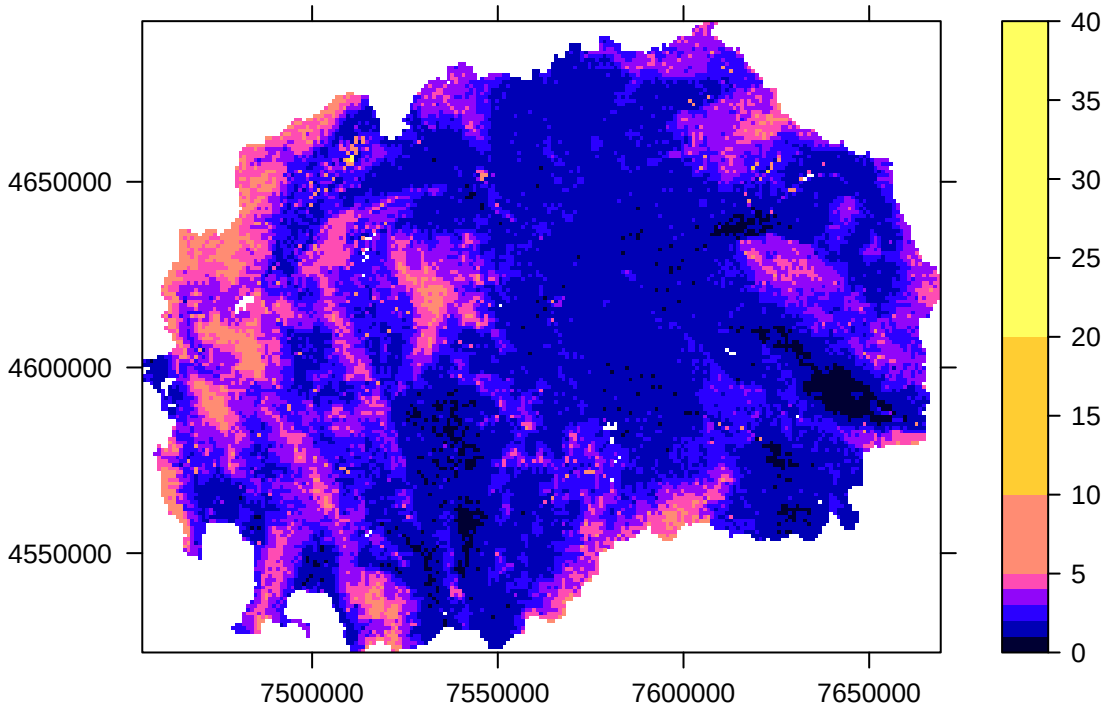



```
# plot with the  raster package
plot(p.r)
```

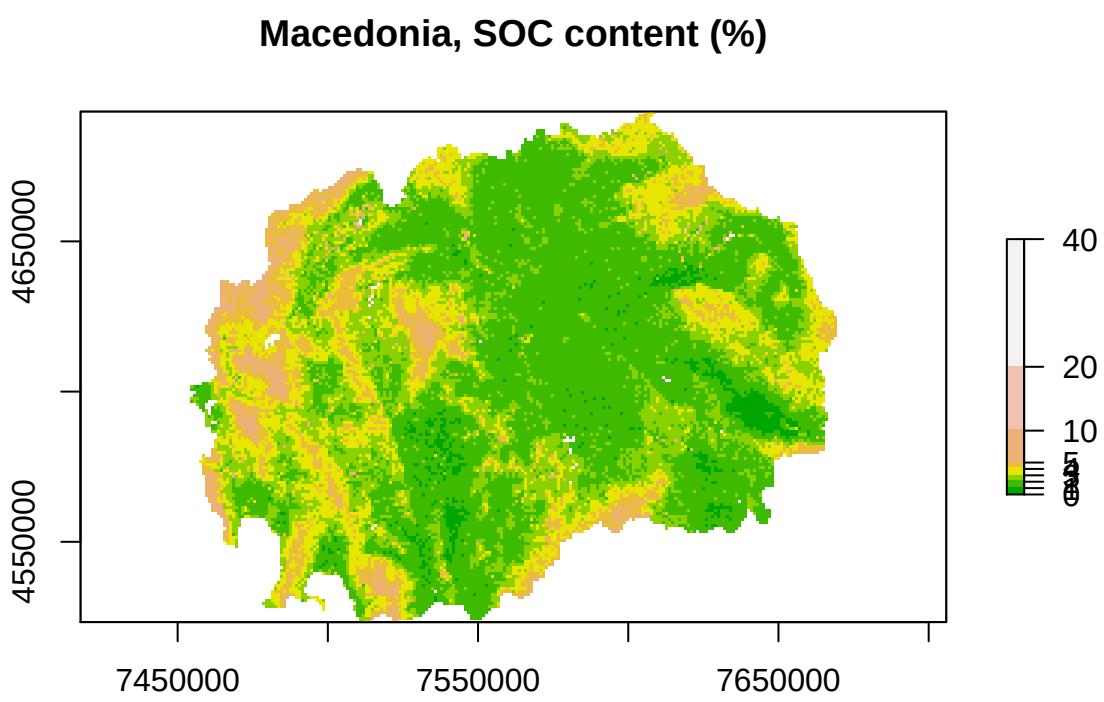



There is not much contract in the plots, so we can try to polish the plots a bit by defining legend class breaks.

```
# sp package
spplot(p, zcol = "SOC", at = c(0,1,2,3,4,5,10,20,40), scales = list(draw=TRUE), main = "Macedonia, SOC
```




```
# or alternatively
plot(p.r, breaks = c(0,1,2,3,4,5,10,20,40), col = terrain.colors(8), main = "Macedonia, SOC content (%)
```




### 3.4 Export the map

We can export the map to a GeoTiff file that you can open and further process in a GIS. We export to a newly defined output folder.

```
# create output folder
dir.create(paste0(getwd(),"/out"))

# save as GeoTIFF
writeGDAL(p["SOC"], fname = "./out/SOC.tif", drivername = "GTiff", type = "Float32")
```

### 3.5 Creating interactive maps

Let's now take a look on how to create some interactive maps with the **leaflet** package. We will start with the point data. The first thing to do is to reproject the data to the WGS84 coordinate system using the **spTransform()** function.

```
# copy regression matrix to a new object
d.ll <- rm

# convert to spatial object amd set projection
coordinates(d.ll) <- ~X_coord+Y_coord
proj4string(d.ll) <- CRS("+init=epsg:6316")
```

```
# reproject to WGS84
d.ll <- spTransform(d.ll, CRSobj = CRS("+init=epsg:4326"))
```

Then we will create a `leaflet` map. We start simple and then add extra functionality such as labels for the markers and a background map.

```
# create a basic map
leaflet() %>%
  addTiles() %>%
  addMarkers(data = d.ll)
```

```
# creat pop-ups for markers (displaying the SOC content)
my_pops <- paste0(
   "<strong>Site: </strong>",
   d.ll$ProfID,
   '<br>
   <strong> SOC content (%): </strong>',
   round(d.ll$SOC,1)
)

# create interactive map
leaflet() %>%
   addProviderTiles("Esri.WorldImagery") %>%
   addMarkers(data = d.ll, popup = my_pops)
```

Click on a marker in the map!

We can further enhance the map. For instance by using different markers.

```
# define colour ramps for markers
pal1 <- colorQuantile("YlOrBr", domain = rm$SOC)
pal2 <- colorNumeric(SAGA_pal[[1]], domain = d.ll$SOC, na.color = "transparent")

# markers based on SOC quantiles
(ll1 <- leaflet() %>%
          addProviderTiles("Esri.WorldImagery") %>%
          addCircleMarkers(data = d.ll, color = ~pal1(SOC), popup = my_pops) %>%
          addLegend("bottomright", pal = pal1, values = d.ll$SOC,
                    title = "SOC content quantiles",
                    opacity = 0.8)
)
```

```
# markers based on SOC content
(ll2 <- leaflet() %>%
          addProviderTiles("Esri.WorldImagery") %>%
          addCircleMarkers(data = d.ll, color = ~pal2(SOC), popup = my_pops) %>%
          addLegend("bottomright", pal = pal2, values = d.ll$SOC,
                     title = "SOC content (%)",
                     opacity = 0.8)
)
```

The interactive map can be saved as an HTML widget.

```
# save as HTML widget
saveWidget(ll1, file = paste0(getwd(),"/out/SOC_content.html",sep=""))
```

Now let's make an interactive map for raster data. As input for the **`leaflet()`** function we have to use a grid of class **`RasterLayer`**.

```
# define plot aesthetics: header and color scale
header <- "SOC content (%)"
pal <- colorNumeric(SAGA_pal[[1]], values(p.r), na.color = "transparent")

# create leaflet
(ll <- leaflet() %>%
        addProviderTiles("Esri.WorldImagery") %>%
        addRasterImage(p.r, colors=pal, opacity=0.5) %>%
        addLegend(pal=pal, values=values(p.r), title=header)
)
```

The map does not show much contrast. We can enhance contrast by changing the color scale. For instance by using SOC content classes.

```
# define color scale
pal <- colorBin(SAGA_pal[[1]], values(p.r), bins = c(0,1,2,3,4,5,10,20,40), na.color = "transparent")

# create leaflet
(ll <- leaflet() %>%
        addProviderTiles("Esri.WorldImagery") %>%
        addRasterImage(p.r, colors=pal, opacity=0.5) %>%
        addLegend(pal=pal, values=values(p.r), title=header)
)
```

```
# save as widget
saveWidget(ll, file = paste0(getwd(),"/out/SOC_content_RF.html",sep=""))
```

You can also save your plots as a png (or jpeg) or pdf file.

```
png(filename = "./out/SOC_Macedonia.png", width = 600, height = 400, pointsize = 12)
spplot(p, zcol = "SOC", at = c(0,1,2,3,4,5,10,20,40), scales = list(draw=TRUE), main = "Macedonia, SOC
dev.off()
```

```
pdf(file = "./out/SOC_Macedonia.pdf", width = 600, height = 400)
spplot(p, zcol = "SOC", at = c(0,1,2,3,4,5,10,20,40), scales = list(draw=TRUE), main = "Macedonia, SOC
dev.off()
```

That's it. Don't forget to save your outputs!

```
# map
save(p, p.r, file="rfSOCmap.rda")

# model
save(rm,rf,file="randomForestModel.rda")

# environment
save.image("randomForest.rda")
```

## Acknowledgements

This tutorial was developed with support from the Cereal Systems Initiative for South Asia (CSISA), Bill and Melinda Gates Foundation and CIMMYT.

The soil data used in this tutorial is from the Macedonian Soil Information System (Vrscaj et al. 2017), and provided by the Secretariat of the Global Soil Partnership. The raw data was preprocessed into an `R data.frame` that is provided with this tutorial.

The covariate layers were all derived from freely available global biophysical data and have been made available by ISRIC for all territories in the world at 1km resolution from ftp://isric.org (user: gsp, password: gspisric).

# References

Strobl, C., J. Malley, and G. Tutz. 2009. "An introduction to recursive partitioning: Rationale, application, and characteristics of classification and regression trees, bagging, and random forests." *Psychological Methods* 14 (4): 323–48. doi:10.1037/a0016973.

Vrscaj, B., L. Poggio, D. Muaketov, and R. Vargas. 2017. "Utilizing the Legacy Soil Data of Macedonia: The Creation of the Macedonian Soil Information System and its use for digital soil mapping and assessment applications. Abstract Book of Pedometrics 2017, Wageningen, 26 June - 1 July 2017." http://www.pedometrics2017.org/s/Abstract-Book-Pedometrics-2017.pdf.