# Continuous soil attribute modeling and mapping: Decision Trees

Soil Security Laboratory

2018

## 1 Decision Trees

Linear regression is a global model, where there is a single predictive formula holding over the entire data space. With a linear model we therefore make some assumptions about how our target variable relates to the covariates. These may often hold, however, it is models that allow one the flexibility of modelling non-linearity that are increasingly popular in the DSM community. One of these model structures are classification and regression trees (CART). These models are a non-parametric decision tree learning techniques that produces either classification or regression trees. In this section we will concentrate on regression trees because our target variable is numeric i.e. a continuous variable. Later we will look at classification trees for categorical variables. Decision trees (either regression or classification) are formed by a collection of rules based on variables in the modeling data set:

- Rules based on variables' values are selected to get the best split to differentiate observations based on the dependent variable.
- Once a rule is selected and splits a node into two, the same process is applied to each subsequent node (i.e. it is a recursive procedure).
- Splitting stops when CART detects no further gain can be made, or some pre-set stopping rules are met. Alternatively, the data are split as much as possible and then the tree is later pruned.

Each branch of the tree ends in a terminal node. Each observation falls into one and exactly one terminal node, and each terminal node is uniquely defined by a set of rules. For a regression tree, the terminal node is a single value, or could be a regression model (which is the case for Cubist models which we will look at later). Implementation of regression trees in `R` is provided both through the `rpart` and `party` packages. We will use the `rpart` package and its `rpart` function. However, the `party` package through the `ctree` function offers more functionality, and implements the partitioning in a more statistical robust fashion. Both functions however can handle both continuous and categorical predictor variables.

First lets get the data.

```
library(ithir)
library(raster)
library(rgdal)
library(sp)

# point data
data(HV_subsoilpH)

# Start afresh round pH data to 2 decimal places
HV_subsoilpH$pH60_100cm <- round(HV_subsoilpH$pH60_100cm, 2)

# remove already intersected data
HV_subsoilpH <- HV_subsoilpH[, 1:3]

# add an id column
HV_subsoilpH$id <- seq(1, nrow(HV_subsoilpH), by = 1)

# re-arrange order of columns
HV_subsoilpH <- HV_subsoilpH[, c(4, 1, 2, 3)]

# Change names of coordinate columns
names(HV_subsoilpH)[2:3] <- c("x", "y")


# grids (covariate raster)
data(hunterCovariates_sub)
```

Perform the covariate intersection.

```
coordinates(HV_subsoilpH) <- ~x + y

# extract
DSM_data <- extract(hunterCovariates_sub, HV_subsoilpH, sp = 1, method = "simple")
DSM_data <- as.data.frame(DSM_data)
str(DSM_data)
```

```
## 'data.frame': 506 obs. of  15 variables:
##  $ id                      : num  1 2 3 4 5 6 7 8 9 10 ...
##  $ x                       : num  340386 340345 340559 340483 340734 ...
##  $ y                       : num  6368690 6368491 6369168 6368740 6368964 ...
##  $ pH60_100cm              : num  4.47 5.42 6.26 8.03 8.86 7.28 4.95 5.61 5.39 3.44 ...
##  $ Terrain_Ruggedness_Index: num  1.34 1.42 1.64 1.04 1.27 ...
##  $ AACN                    : num  1.619 0.281 2.301 1.74 3.114 ...
##  $ Landsat_Band1           : num  57 47 59 52 62 53 47 52 53 63 ...
##  $ Elevation               : num  103.1 103.7 99.9 101.9 99.8 ...
##  $ Hillshading             : num  1.849 1.428 0.934 1.517 1.652 ...
##  $ Light_insolation        : num  1689 1701 1722 1688 1735 ...
##  $ Mid_Slope_Positon       : num  0.876 0.914 0.844 0.848 0.833 ...
##  $ MRVBF                   : num  3.85 3.31 3.66 3.92 3.89 ...
##  $ NDVI                    : num  -0.143 -0.386 -0.197 -0.14 -0.15 ...
```

```
##  $ TWI                       : num  17.5 18.2 18.8 18 17.8 ...
##  $ Slope                     : num  1.79 1.42 1.01 1.49 1.83 ...
```

Often it is handy to check to see whether there are missing values both in the target variable and of the covariates. It is possible that a point location does not fit within the extent of the available covariates. In these cases the data should be excluded. A quick way to assess whether there are missing or NA values in the data is to use the `complete.cases` function.

```
which(!complete.cases(DSM_data))
```

```
## integer(0)
```

```
DSM_data <- DSM_data[complete.cases(DSM_data), ]
```

Fitting a decision tree in `R` is quite similar to that for linear models:

```
library(rpart)
```

```
## Warning:  package 'rpart' was built under R version 3.2.5
```

```
set.seed(123)
training <- sample(nrow(DSM_data), 0.7 * nrow(DSM_data))
hv.RT.Exp <- rpart(pH60_100cm ~ AACN + Landsat_Band1 + Elevation + Hillshading +
    Mid_Slope_Positon + MRVBF + NDVI + TWI, data = DSM_data[training, ],
    control = rpart.control(minsplit = 50))
```

It is worthwhile to look at the help file for `rpart` particularly those aspects regarding the `rpart.control` parameters which control the `rpart` fit. Often it is helpful to just play around with the parameters to get a sense of what does what. Here for the `minsplit` parameter within `rpart.control` we are specifiying that we want at least 50 observations in a node in order for a split to be attempted.

Detailed results of the model fit can be provided via the `summary` and `printcp` functions.

```
summary(hv.RT.Exp)
```

The `summary` output provides detailed information of the data splitting as well as information as to the relative importance of the covariates.

```
printcp(hv.RT.Exp)
```

The `printcp` function provides the useful output of indicating which covariates were included in the final model. For the visually inclined, a plot of the tree assists a lot to interpret the model diagnostics and assessing the important covariates too(Figure 1 ).

```
plot(hv.RT.Exp)
text(hv.RT.Exp)
```

As before, we can use the `goof` function to test the performance of the model fit both internally and externally.

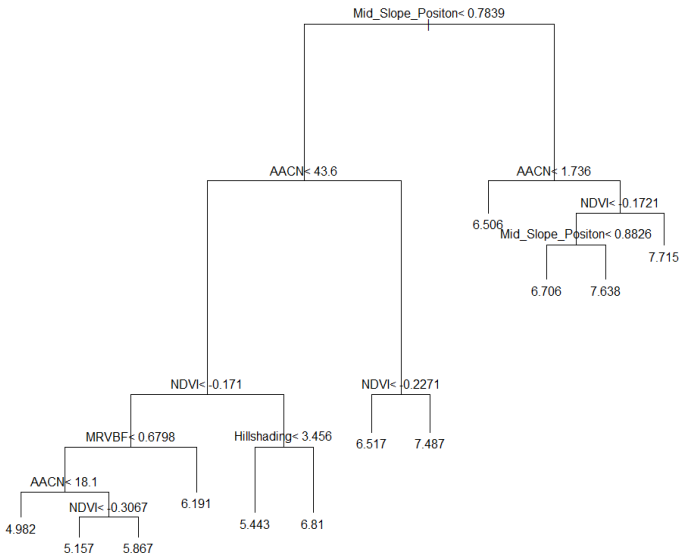


Figure 1: Decision tree of Hunter Valley soil pH (60-100cm).

```
# Internal validation
RT.pred.C <- predict(hv.RT.Exp, DSM_data[training, ])
goof(observed = DSM_data$pH60_100cm[training], predicted = RT.pred.C)
```

```
##          R2 concordance      MSE     RMSE          bias
## 1 0.3627782   0.5328417 1.154466 1.074461 -8.881784e-16
```

```
# External validation
RT.pred.V <- predict(hv.RT.Exp, DSM_data[-training, ])
goof(observed = DSM_data$pH60_100cm[-training], predicted = RT.pred.V)
```

```
##           R2 concordance      MSE     RMSE       bias
## 1 0.05587206   0.2131035 1.858148 1.363139 0.03135219
```

The decision tree model performance is not too dissimilar to the MLR model. Looking at the xy-plot from the external validation (Figure 2) and the decision tree (Figure 1), it becomes clear that a potential issue is apparent. This is: there are only a finite number of possible outcomes in terms of the predictions.

This finite property becomes obviously apparent once we make a map by applying the fitted model to the covariates (using the `raster` predict function and `hunterCovariates_sub` object). (Figure 3).

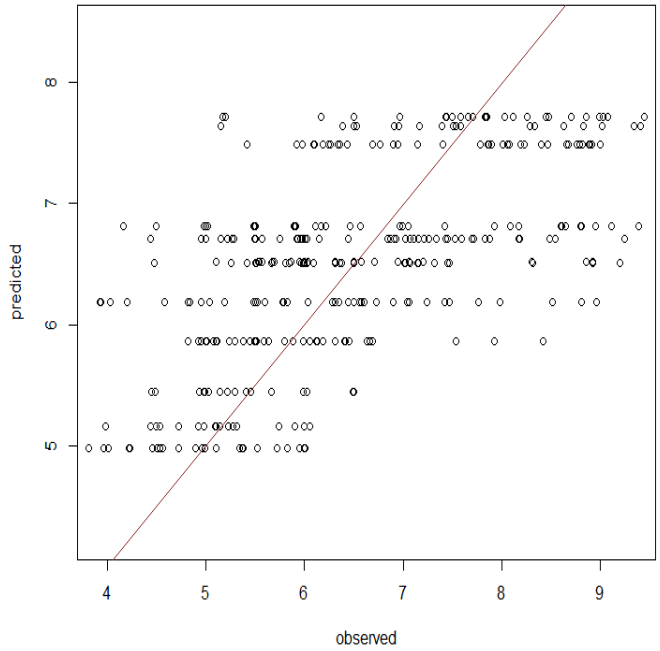


Figure 2: Decision tree xy-plot plot of predicted soil pH (60-100cm) (validation data set).

```
map.RT.r1 <- predict(hunterCovariates_sub, hv.RT.Exp, "soilpH_60_100_RT.tif",
    format = "GTiff", datatype = "FLT4S", overwrite = TRUE)

plot(map.RT.r1, main = "Decision tree predicted Hunter Valley soil pH (60-100cm")
```

# References

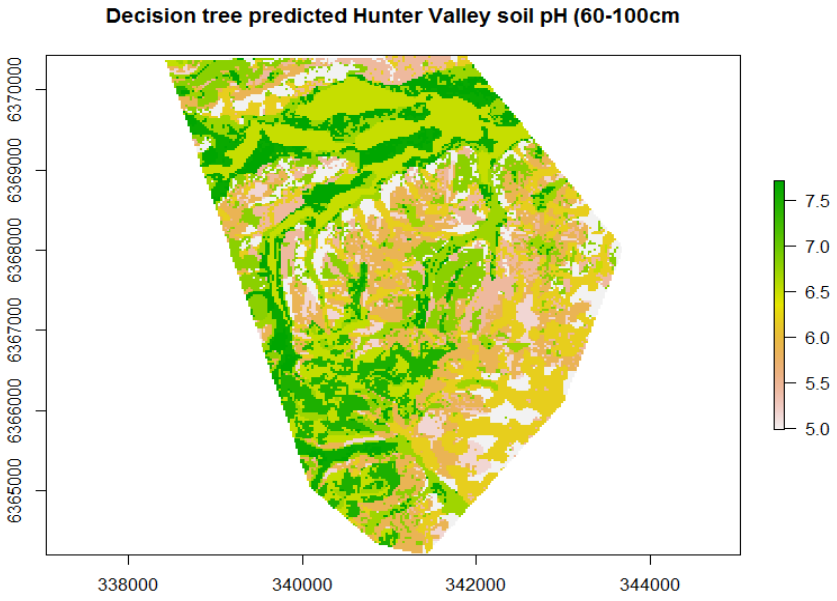


Figure 3: Decision tree predicted Hunter Valley soil pH (60-100cm).