Continuous soil attribute modeling and mapping: Caret

Soil Security Laboratory

2018

1 Advanced Work: Model fitting with Caret Package

It becomes quickly apparent that there are many variants of prediction functions that could be used for DSM. As was observed, each of the models used have their relative advantages and disadvantages. Each also has their own specific parameterisations and quirks for fitting. Sometimes for the various parameters that are used for model training are chosen without any sort of optimisation, even due consideration sometimes. Sometimes we might be confronted with many possible model structures to use, it is often difficult to make a choice what to use, and just default with a model we know well or have used often without considering alternatives. This is where the `caret` R package `http://topepo.github.io/caret/index.html` comes into its own in terms of efficiency and streamlining the workflow for fitting models and optimising some of those parameter variables. As the dedicated website indicates (`http://topepo.github.io/caret/index.html`), the `caret` package (short for Classification And REgression Training) is a set of functions that attempt to streamline the process for creating predictive models. As we have seen, there are many different modeling functions in R. Some have different syntax for model training and/or prediction. The `caret` package provides a uniform interface to the various functions themselves, as well as a way to standardize common tasks (such parameter tuning and variable importance). There are currently nearly 400 model functions that the `caret` package interfaces with.

First get the data and perform the covariate data intersection

```
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), ]
```

To begin, we first need to load the package into R:

```
library(caret)
```

The workhorse of the `caret` package is the `train` function. We can specify the model to be fitted in two ways:

```
# 1.
fit <- train(form = pH60_100cm ~ AACN + Landsat_Band1 + Elevation + Hillshading +
    Mid_Slope_Positon + MRVBF + NDVI + TWI, data = DSM_data, method = "lm")

# or 2.
fit <- train(x = DSM_data[, c(6, 7, 8, 9, 11, 12, 13, 14)], y = DSM_data$pH60_100cm,
    method = "lm")
```

Using the `summary(fit)` command brings up the model parameter estimates, while the object `fit` also contains a summary of some useful model goodness of fit diagnostics such as the RMSE and R^2 statistics. You can control how model validation is done where options include simple goodness of fit calibration, k-fold cross-validation, and leave-one-out cross-validation. This option is controlled using the parameter `trControl` in the `train` function. The example below illustrates a 5-fold cross validation of a linear regression model with 10 repetitions.

```
fit <- train(x = DSM_data[, c(6, 7, 8, 9, 11, 12, 13, 14)], y = DSM_data$pH60_100cm,
    method = "lm", trControl = trainControl(method = "repeatedcv", number = 5,
        repeats = 10))

fit
```

```
## Linear Regression
##
## 506 samples
##   8 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 10 times)
## Summary of sample sizes: 404, 405, 405, 405, 405, 404, ...
## Resampling results:
##
##   RMSE      Rsquared
##   1.186179  0.2312369
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

There are a lot of potential models that you could consider too for DSM. Check out `http://topepo.github.io/caret/modelList.html` or print them as below:

```
list_of_models <- modelLookup()
head(list_of_models)
```

```
##      model parameter          label forReg forClass probModel
## 1      ada      iter         #Trees  FALSE     TRUE      TRUE
## 2      ada  maxdepth Max Tree Depth  FALSE     TRUE      TRUE
## 3      ada        nu  Learning Rate  FALSE     TRUE      TRUE
## 4   AdaBag    mfinal         #Trees  FALSE     TRUE      TRUE
## 5   AdaBag  maxdepth Max Tree Depth  FALSE     TRUE      TRUE
## 9 adaboost     nIter         #Trees  FALSE     TRUE      TRUE
```

```
# The number of models caret interfaces with
nrow(list_of_models)
```

```
## [1] 452
```

You can choose which model to use in the `train` function with the `method` option. You will note that the fitting of the Cubist and Random Forest models below automatically attempt to optimise some of the fitting parameters, for example the `mtry` parameter for Random Forest. To look at what parameters can optimised for each model in caret we can use the `modelLookup` function.

```
# Cubist model
modelLookup(model = "cubist")
```

```
##    model  parameter       label forReg forClass probModel
## 1 cubist committees #Committees   TRUE    FALSE     FALSE
## 2 cubist  neighbors  #Instances   TRUE    FALSE     FALSE
```

```
fit_cubist <- train(x = DSM_data[, c(6, 7, 8, 9, 11, 12, 13, 14)],
y = DSM_data$pH60_100cm,
    method = "cubist", trControl = trainControl(method = "cv", number = 5))

# random forest model
modelLookup(model = "rf")
```

```
##   model parameter                         label forReg forClass probModel
## 1    rf      mtry #Randomly Selected Predictors   TRUE     TRUE      TRUE
```

```
fit_rf <- train(x = DSM_data[, c(6, 7, 8, 9, 11, 12, 13, 14)], y = DSM_data$pH60_100cm,
    method = "rf", trControl = trainControl(method = "cv", number = 5))
```

Using the fitted model, predictions can be achieved with the `predict` function:

```
# Cubist model
pred_cubist <- predict(fit_cubist, DSM_data)

# To raster data
pred_cubistMap <- predict(hunterCovariates_sub, fit_cubist)
```

There is plenty of other added functionality of the `caret` package. In addition to the detailed resources mentioned above, it always pays to look over the help files that are associated with each function.

References