---

title: "Classification and Regression Trees"

output: learnr::tutorial

runtime: shiny\_prerendered

---

```{r setup, include=FALSE}

#Load packages

library(raster)

library(rpart)

library(dismo)

library(learnr)

# Setup

knitr::opts\_chunk$set(echo = FALSE,

message = FALSE,

warning = FALSE)

gradethis::gradethis\_setup()

tutorial\_options(exercise.blanks = TRUE)

# Load the data

# Land Cover

nlcd <- stack("Data/nlcd-L1\_Sml.tif")

names(nlcd) <- c("nlcd2001", "nlcd2011")

# Only 2011 data

nlcd2011 <- nlcd[[2]] # OR nlcd[["nlcd2011"]]

#table of classes

classdf <- data.frame(classvalue1 = c(1, 2, 3, 4, 5, 7, 8, 9), # Class Number.

nlcdclass = c("Water", "Developed", "Barren", "Forest", "Shrubland", "Herbaceous", "Planted/Cultivated", "Wetlands"), # Class Name.

classcolor = c("#5475A8", "#B50000", "#D2CDC0", "#38814E", "#AF963C", "#D1D182", "#FBF65D", "#C8E6F8") # # Hex codes of colours.

)

# Satellite images

landsat5 <- stack("Data/centralvalley-2011LT5\_Sml.tif")

names(landsat5) <- c('blue', 'green', 'red', 'NIR', 'SWIR1', 'SWIR2')

# Stratified sampling

samp2011 <- sampleStratified(x = nlcd2011,size = 200,na.rm = TRUE, sp = TRUE)

# data for modelling

sampdata <- data.frame(classvalue = samp2011@data$nlcd2011,

extract(x = landsat5,y = samp2011,df = TRUE)[-1])

# CART model

cartMod <- rpart(formula = as.factor(classvalue)~.,data = sampdata,method = 'class',cp = 0.01,minsplit = 5)

# K-fold data

k.foldDta <- kfold(sampdata, k = 5, by = sampdata$classvalue)

# Prediction

CART.pr2011 <- predict(object = landsat5, model = cartMod, type = 'class')

# Kfold Validation for the CART model

CART.Kfold.List <- lapply(1:5, function(k){

train <- sampdata[k.foldDta!= k, ]

test <- sampdata[k.foldDta == k, ]

cartMod.kfold.Mod <- rpart(formula = as.factor(classvalue)~.,data = train, method = 'class', cp = 0.01, minsplit = 5)

pclass <- predict(object = cartMod.kfold.Mod, newdata = test, type = "class")

cbind(Observed = test$classvalue,

Predicted = as.numeric(as.vector(pclass)))

})

CART.Kfold <- do.call("rbind",CART.Kfold.List)

CART.Kfold <- data.frame(CART.Kfold)

colnames(CART.Kfold) <- c('observed', 'predicted')

CART.conmat <- table(CART.Kfold)

```

## Supervised Classification of satellite images

In supervised classification, you have prior knowledge about some of the features of interest (in this case, land-cover types). In the context of satellite images, this prior knowledge can come from, for example, fieldwork, reference spatial data or interpretation of high-resolution imagery (such as available on Google maps). Specific sites in the study area that represent homogeneous examples of these known land-cover types are identified. These areas are commonly referred to as training sites because the spectral properties of these sites are used to train the classification algorithm.

## Reference data

As stated above, supervised classification is based on the principle that you have prior knowledge about some of the features of interest. In this case, you will use the United States National Land Cover Database for the year 2011 (NLCD <https://www.mrlc.gov/>) to identify the spectral signal of predefined land cover classes. The NLCD is a 30m Landsat-based land cover database spanning four epochs (1992, 2001, 2006 and 2011). NLCD 2011 has two pairs of class values and names that correspond to the levels of land use and land cover classification system (see <https://www.mrlc.gov/nlcd11\_leg.php> for the class names in NCLD 2011). The names represent the level of complexity, level I (and the one you will use) being the simplest, with broad land cover categories.

<div class="alert alert-info">

\*\*Your task:\*\*

Load the NLCD data into `R`. The NLCD data in the file `nlcd-L1.tif` is a `RasterStack`, with two bands located in R-Studio cloud data [see the Files tab in the lower right window].

\* Band 1 is the land cover classification for 2001.

\* Band-2 is the land cover classification for 2011.

</div>

<div class="alert alert-info">

\*\*Your task:\*\*

Plot the Land Cover classification raster `nlcd`.

Here, define nine (9) values contrasting colour scheme using the `hcl.colors()` function.

</div>

```{r nlcdPlot, exercise=TRUE, exercise.lines=10}

plot(x = \_\_\_\_\_\_\_, # raster to plot.

col = sample(hcl.colors(\_\_\_\_\_\_, palette = "Zissou")) # Define the colour ramp

)

```

```{r nlcdPlot-solution, exercise.reveal\_solution=F}

plot(x = nlcd, # raster to plot.

col = sample(hcl.colors(9, palette = "Zissou")) # Define the colour ramp

)

```

```{r nlcdPlot-check}

grade\_code()

```

However, it is impossible to determine the land-cover class for each grid cell based on this figure. For this, you will need to define the class names (the text name for each value in the raster). The object `classdf` is where the class names and colours are put together.

<div class="alert alert-info">

\*\*Your task:\*\*

Print the `classdf` object to see the classification.

</div>

```{r LookUpTble, exercise=TRUE}

#Note from the table above that there is no class with value 6

\_\_\_\_\_\_\_

```

```{r LookUpTble-solution, exercise.reveal\_solution=F}

#Note from the table above that there is no class with value 6

classdf

```

```{r LookUpTble-check}

grade\_code()

```

## Generate sample sites

Training and/or validation data can come from a variety of sources. In this example, you will generate the training and validation sample sites using the NLCD reference `RasterLayer.` Alternatively, you can use predefined sites that you may have collected from other sources. You will generate the sample sites following a stratified random sampling to ensure each Land-Use/Land-Cover class is sampled evenly.

<div class="alert alert-info">

\*\*Your task:\*\*

Use the function `sampleStratified` from the `raster` package to generate a stratified random sample of the cell values of `nlcd2011`. You will select 200 observations per class.

</div>

```{r StratSmpl, exercise=TRUE, exercise.lines=15}

# Define the training sites locations

# Set the random number generator to make the results reproducible.

set.seed(99)

# Sampling

samp2011 <- sampleStratified(x = \_\_\_\_\_, # Raster to be "sampled"

size = \_\_\_\_\_, # Number of samples per class

na.rm = TRUE, # NA values are removed from random sample?

sp = TRUE # Output as a SpatialPointsDataFrame?

)

# Explore the data in samp2011 using the head() function.

\_\_\_\_\_(\_\_\_\_\_\_)

```

```{r StratSmpl-solution, exercise.reveal\_solution=F}

# Define the training sites locations

# Set the random number generator to make the results reproducible.

set.seed(99)

# Sampling

samp2011 <- sampleStratified(x = nlcd2011, # Raster to be "sampled"

size = 200, # Number of samples per class

na.rm = TRUE, # NA values are removed from random sample?

sp = TRUE # Output as a SpatialPointsDataFrame?

)

# Explore the data in samp2011 using the head() function.

head(samp2011)

```

```{r StratSmpl-check}

grade\_code()

```

The stratified random sampling output is a `SpatialPointsDataFrame` object (because the `sp` argument is set to `TRUE`), which is `R`-speak for a point-shapefile. Also, when printing the `samp2011` object, you can see there are two variables. First, the `cell` column, that contains the sampled cell numbers from `nlcd2011`. Second, the `nlcd2011` column contains the class values (1-9) corresponding to each Land Cover Class.

You have sampled many places of the `nlcd2011` Landcover raster. \*But how can you be sure that each class has 200 random observations?\* For this, you can tabulate the results from your stratified random sampling using the function `table` (look into `?table` to know how the function works)

<div class="alert alert-info">

\*\*Your task:\*\*

Tabulate the results from your stratified random sampling `SpatialPointsDataFrame` using the function `table` (look into `?table` to know how the function works)

</div>

```{r SmplTbl, exercise=TRUE, exercise.lines=10}

# Extract the land cover classes identities (class values) for each plot sampled in the stratified random sample SpatialPointsDataFrame.

SampLCC <- \_\_\_\_\_\_\_\_\_\_$\_\_\_\_\_\_\_

# Use the table() function to tabulate the number of land cover classes sampled by the `sampleStratified` function

\_\_\_\_\_\_(\_\_\_\_\_\_)

```

```{r SmplTbl-solution, exercise.reveal\_solution=F}

# Extract the land cover classes identities for each plot sampled in the stratified random sample.

SampLCC <- samp2011$nlcd2011

# Use the table() function to tabulate the number of land cover classes sampled by the `sampleStratified` function

table(SampLCC)

```

```{r SmplTbl-check}

grade\_code()

```

The table above clearly shows that each land cover class has been sampled precisely 200 times.

## Extract values for sites

Here you will use Landsat data (`landsat5`) already loaded in the memory. As you now have a series of sites with an associated Land cover class, you can extract the cell values from the `landsat5` `RasterStack.` These band values will be the predictor variables and “class values” from `nlcd2011` will be the response variable. To extract these, you will use the function `extract`.

<div class="alert alert-info">

\*\*Your task:\*\*

Use the function `extract` to extract for each randomly selected site per Landcover class the (those in `samp2011`) hyperspectral bands in `landsat5`.

</div>

```{r HiprSpcDta, exercise=TRUE, exercise.lines=20}

# Extract the values from `landsat5`

sampvals <- extract(x = \_\_\_\_\_\_\_\_, # Define the raster to be sampled.

y = \_\_\_\_\_\_\_\_, # Define the spatial vector data with the locations to sample.

df = TRUE # Return the results as a data.frame

)

# Explore sampvals

head(sampvals)

# combine the class information with extracted values

sampdata <- data.frame(classvalue = samp2011@data$\_\_\_\_\_\_, # Get the Landcover land cover classes identities from the samp2011 object.

\_\_\_\_\_\_\_\_[, -1] # The extracted data.frame with the hyperspectral bands in `landsat5` minus the ID column

)

# Explore the resulting data.frame using the head() function.

\_\_\_\_\_\_\_\_(\_\_\_\_\_\_\_\_)

```

```{r HiprSpcDta-solution, exercise.reveal\_solution=F}

# Extract the values from `landsat5`

sampvals <- extract(x = landsat5, # Define the raster to be samples.

y = samp2011, # Define the spatial object data with the locations to sample.

df = TRUE # Return the results as a data.frame?

)

# Explore sampvals

head(sampvals)

# combine the class information with extracted values

sampdata <- data.frame(classvalue = samp2011@data$nlcd2011, # Get the Landcover land cover classes identities from the samp2011 object.

sampvals[, -1] # The extracted data.frame with the hyperspectral bands in `landsat5` minus the ID column

)

# Explore the resulting data.frame using the head() function.

head(sampdata)

```

```{r HiprSpcDta-check}

grade\_code()

```

## Train the CART classifier

Now you will train the CART classification algorithm using `sampdata` dataset. In this model, land cover classes are the response variable (set up as a factor as these are coded as numbers in the raster) and the 'blue', 'green', 'red', 'NIR', 'SWIR1', 'SWIR2' bands are the predictors.

For this, you will use the `rpart` function from the `rpart` package. It is preferred to use the function `rpart` instead of `tree` due to the way `rpart` handles surrogate variables. Also, it follows Breiman et al. (1984) [the reference that set the stage for this type of regressions] quite closely. The name of the function stands for ‘Recursive Partitioning’. For this, you will use the functions in the `rpart` package.

<div class="alert alert-info">

\*\*Your task:\*\*

Load the `rpart` package and build a CART model ( using the `rpart` function) that predicts land cover classes based on the hyperspectral signal of sampled sites.

Remember that the response variable needs to be a factor!! So, you can either Define the landcover class as a factor directly in the model (that is use the `classvalue` variable in `sampdata`) or recode `classnames1` as a factor in `sampdata` before building the model. Here you will use the first option.

One more point, This is a \*\*CLASIFICATION\*\* exercise. This is a clue for you to know which type of model (one of "`anova`", "`poisson`", "`class`" or "`exp`". ) you are building.

</div>

```{r CART1, exercise=TRUE, exercise.lines=20}

# Train the model

cartMod <- rpart(formula = as.factor(\_\_\_\_\_\_\_\_)~., # Define a formula linking the response (land cover classes) to the predictors (bands in Landsat5) - remember to Define the Land cover class as a factor

data = \_\_\_\_\_\_\_\_, # Define the object with the response/predictors data!

method = '\_\_\_\_\_\_\_\_', # Define the Type of model you are building.

cp = 0.01, # Define the minimum level of increase in the complexity parameter

minsplit = 5 #Define the minimum number of observations that must exist in a node in order for a split to be attempted

)

# Show the contents of the classification model you just built using the

cartMod

```

```{r CART1-solution, exercise.reveal\_solution=F}

# Train the model

cartMod <- rpart(formula = as.factor(classvalue)~., # Define a formula linking the response (land cover classes) to the predictors (bands in Landsat5) - remember to Define the Land cover class as a factor

data = sampdata, # Define the object with the response/predictors data!

method = 'class', # Define the Type of model you are building.

cp = 0.01, # Define the minimum level of increase in the complexity parameter

minsplit = 5 #Define the minimum number of observations that must exist in a node in order for a split to be attempted

)

# Show the contents of the classification model you just built using the

cartMod

```

```{r CART1-check}

grade\_code()

```

## Plot the CART classifier

As informative as the `print` of the `cartMod` object is, the best way to evaluate a tree-based calcification is to plot the classification tree.

<div class="alert alert-info">

\*\*Your task:\*\*

Plot the classification tree `cartMod` you build in the previous step.

</div>

```{r CARTplt, exercise=TRUE, exercise.lines=15}

# Plot the trained classification tree

plot(x = \_\_\_\_\_\_\_, # Define the classification/regression tree to plot.

uniform = TRUE, # Use a uniform vertical spacing of the nodes?

main = "Classification Tree" # Figure main title

)

# Place the leaves Names on the dendrogram Plot

text(x = \_\_\_\_\_\_\_, # Define the classification/regression tree to plot,

cex = 0.8 # Size of the text

)

```

```{r CARTplt-solution, exercise.reveal\_solution=F}

# Plot the trained classification tree

plot(x = cartMod, # Define the classification/regression tree to plot.

uniform = TRUE, # Use a uniform vertical spacing of the nodes?

main = "Classification Tree" # Figure main title

)

# Place the leaf’s Names on the dendrogram Plot

text(x = cartMod, # Define the classification/regression tree to plot,

cex = 0.8 # Size of the text

)

```

```{r CARTplt-check}

grade\_code()

```

In the classification tree plot made above, class values are printed at the leaf nodes. You can find the corresponding land cover names in `classdf`.

## Pruning your classification tree - 1

You will now focus on the set of procedures in `R` that you can use to prune classification/regression trees (that is, cut them at specific points). This "pruning" is based on the \*cost–complexity measure\*, which is the base of model simplification in regression trees.

One way to look at the changes in complexity is to generate a table of \*\*Complexity Parameters\*\* and the associated errors (`error` or deviance and `error` or cross-validation error). For this, you will use both the `printcp` and `plotcp` functions of the `raprt` package.

<div class="alert alert-info">

\*\*Your task:\*\*

\*\*Print\*\* (using `printcp()`) and \*\*plot\*\* (using `plotcp()`) the complexity parameter Table for your CART model (`cartMod`).

</div>

```{r CompParPrint, exercise=TRUE}

# Print the Complexity Parameter Table for a rpart Fit

\_\_\_\_\_\_\_(\_\_\_\_\_\_\_)

# Plot a Complexity Parameter Table for a rpart Fit. Here, note the dashed line, which shows the 1-SE of the minimum cp.

\_\_\_\_\_\_\_(\_\_\_\_\_\_\_)

```

```{r CompParPrint-solution, exercise.reveal\_solution=F}

# print the Complexity Parameter Table for a rpart Fit

printcp(cartMod)

# Plot a Complexity Parameter Table for a rpart Fit. Here note the dashed line, which shows the 1-SE of the minimum cp.

plotcp(cartMod)

```

```{r CompParPrint-check}

grade\_code()

```

## Pruning your classification tree - 2a

Model simplification in regression trees is the same as "pruning" your tree at the point your complexity parameter full fills either of two criteria:

1. The first level (i.e., least split) with minimum error. This is the point where \*xerror\* starts increasing again. The first level only kicks in when multiple levels have the same minimum error. \*\*This is the method most commonly used\*\*.

<div class="alert alert-info">

\*\*Your task:\*\*

Prune the classification tree using the `prune` function using the minimum criteria - that is, where `xerror` is the lowest.

Remember that your tree model is named `cartMod`, and that you can get a table with the change in Complexity Parameters for a given number of branches using the `printcp` function.

Finish by plotting the pruned tree.

</div>

```{r CARTprune1, exercise=TRUE, exercise.lines=25}

# Save the Complexity Parameter Table for Fitted rpart Object

printcpSumm <- as.data.frame(printcp(\_\_\_\_\_\_\_))

# Pruning based on the minimum criteria

# Extract the complexity parameter where xerror is the lowest

CPVal.Min <- printcpSumm[which.min(printcpSumm$\_\_\_\_\_\_\_), "CP"]

# Pruning based on the minimum criteria

cartMod.pruned.1 <- prune(\_\_\_\_\_\_\_, # Define the CART model to prune.

cp = \_\_\_\_\_\_\_ # Define the CP value at which the CART model will be pruned

)

# Plot the Pruned tree

plot(x = \_\_\_\_\_\_\_, # Define the classification/regression tree to plot.

uniform = TRUE, # Use a uniform vertical spacing of the nodes?

main = "Classification Tree - min criteria" # Figure main title

)

# Place the leaf’s Names on the dendrogram Plot

text(x = \_\_\_\_\_\_\_, # Define the classification/regression tree to plot,

cex = 0.8, # Size of the text

xpd = NA)

```

```{r CARTprune1-solution, exercise.reveal\_solution=F}

# Save the Complexity Parameter Table for Fitted rpart Object

printcpSumm <- as.data.frame(printcp(cartMod))

# Pruning based on the minimum criteria

# Extract the complexity parameter where xerror is the lowest

CPVal.Min <- printcpSumm[which.min(printcpSumm$xerror), "CP"]

# Pruning based on the minimum criteria

cartMod.pruned.1 <- prune(cartMod, # Define the CART model to prune.

cp = CPVal.Min # Define the CP value at which the CART model will be pruned

)

# Plot the Pruned tree

plot(x = cartMod.pruned.1, # Define the classification/regression tree to plot.

uniform = TRUE, # Use a uniform vertical spacing of the nodes?

main = "Classification Tree - min criteria" # Figure main title

)

# Place the leaf’s Names on the dendrogram Plot

text(x = cartMod.pruned.1, # Define the classification/regression tree to plot,

cex = 0.8, # Size of the text

xpd = NA)

```

```{r CARTprune1-check}

grade\_code()

```

Based on the first criteria, the `cp` value with minimum `xerror` is 0.012, a tree with ten (10) splits.

## Pruning your classification tree - 2b

Model simplification in regression trees is the same as "pruning" your tree at the point your complexity parameter full fills either of two criteria:

2. The first level where \*error\* falls into the ±1 \*xstd\* range of min(error). That is, $error < min(error) + xstd$. Graphically, it is the first level whose error is at or below horizontal line dashed line when using the `plotcp` function of `rpart.`

<div class="alert alert-info">

\*\*Your task:\*\*

Prune the classification tree using the `prune` function utilising the xerror` ±1 xstd criteria - The first value was that the xerror±xstd is at or below the min(xerror) + xstd.

Remember that your tree model is named `cartMod`, and that you can get a table with the change in Complexity Parameters and their variability for a given number of branches using the `printcp` function.

Finish by plotting the pruned tree.

</div>

```{r CARTprune2, exercise=TRUE, exercise.lines=30}

# Pruning based on the ±1 xstd criteria

# Save the Complexity Parameter Table for Fitted rpart Object

printcpSumm <- as.data.frame(\_\_\_\_\_\_\_(\_\_\_\_\_\_\_))

# What is the upper boundary of the min-xerror

MinxerrorUp <- sum(printcpSumm[which.min(printcpSumm$\_\_\_\_\_\_\_),c("xerror","xstd")])

# What is the lower boundary of each xerror

Lowxerror <- \_\_\_\_\_\_\_$xerror - \_\_\_\_\_\_\_$xstd

# Which number of splits is the first were Lowxerror <= MinxerrorUp

Crit.Split <- min(which(\_\_\_\_\_\_\_ <= \_\_\_\_\_\_\_))

# Extract the complexity parameter using the ±1 xstd range of min(error) criteria

CPVal.xstd <- printcpSumm$\_\_\_\_\_\_\_[\_\_\_\_\_\_\_]

# Pruning based on the ±1 xstd criteria

cartMod.pruned.2 <- prune(\_\_\_\_\_\_\_, # Define the CART model to prune.

cp = \_\_\_\_\_\_\_ # Define the CP value at which the CART model will be pruned

)

# Plot the Pruned tree

plot(x = \_\_\_\_\_\_\_, # Define the classification/regression tree to plot.

uniform = TRUE, # Use a uniform vertical spacing of the nodes?

main = "Classification Tree - ±1 xstd criteria" # Figure main title

)

# Place the leaf’s Names on the dendrogram Plot

text(x = \_\_\_\_\_\_\_, # Define the classification/regression tree to plot,

cex = 0.8, # Size of the text

xpd = NA)

```

```{r CARTprune2-solution, exercise.reveal\_solution=F}

# Pruning based on the ±1 xstd criteria

# Save the Complexity Parameter Table for Fitted rpart Object

printcpSumm <- as.data.frame(printcp(cartMod))

# What is the upper boundary of the min-xerror

MinxerrorUp <- sum(printcpSumm[which.min(printcpSumm$xerror),c("xerror","xstd")])

# What is the lower boundary of each xerror

Lowxerror <- printcpSumm$xerror-printcpSumm$xstd

# Which number of splits is the first were Lowxerror <= MinxerrorUp

Crit.Split <- min(which(Lowxerror <= MinxerrorUp))

# Extract the complexity parameter using the ±1 xstd range of min(error) criteria

CPVal.xstd <- printcpSumm$CP[Crit.Split]

# Pruning based on the ±1 xstd criteria

cartMod.pruned.2 <- prune(cartMod, # Define the CART model to prune.

cp = CPVal.xstd # Define the CP value at which the CART model will be pruned

)

# Plot the Pruned tree

plot(x = cartMod.pruned.2, # Define the classification/regression tree to plot.

uniform = TRUE, # Use a uniform vertical spacing of the nodes?

main = "Classification Tree - ±1 xstd criteria" # Figure main title

)

# Place the leaf’s Names on the dendrogram Plot

text(x = cartMod.pruned.2, # Define the classification/regression tree to plot,

cex = 0.8, # Size of the text

xpd = NA)

```

```{r CARTprune2-check}

grade\_code()

```

Based on the second criteria, the `cp` value with `xerror` ±1 xstd range of `min(xerror)` is 0.016, a tree with eight (8) splits. This is the value where the xerror±xstd is above the dashed line printed by `plotcp(cartMod)`.

Now you have done your first non-parametric regression to generate a supervised classification of a satellite image!!. Now you will approach the process of developing multiple models to address two of the central problems with regression trees: these can be over-elaborated and respond to random data features.

## Model evaluation - K-fold validation

Now that you have a classification model, you can evaluate the accuracy of the prediction of it (e.g., its' capacity to discriminate between cultivated crops, pasture, grassland and shrubs).

To do this assessment, you can use a k-fold validation approach. For this, you will use the `fold` function of the package `dismo`). In this technique, the data used to fit the model is split into `k` groups (typically five groups). In turn, one of the groups will be used for model testing, while you will use the rest of the data for model training (fitting).

<div class="alert alert-info">

\*\*Your task:\*\*

Do a five (5) k-fold partitioning of the training dataset (`sampdata`), stratifying the folds by the Land Cover Class (ensuring that all Classes, the `classvalue` variable is `sampdata` are equally represented).

</div>

```{r KFold, exercise=TRUE, exercise.lines=15}

# set the random seed for reproducibility

set.seed(99)

# Do a k-fold partitioning of the training dataset

k.foldDta <- kfold(\_\_\_\_\_\_\_, # the source data to use

k = \_\_\_\_\_\_\_, # number of Folds (here you use 5)

by = sampdata$\_\_\_\_\_\_\_) # define how to stratify the folds

# Tabulate the k-fold partitioning dataset by the Land Cover Class to see the number of samples per land cover class to be used in each fold.

table(\_\_\_\_\_\_\_,

sampdata$\_\_\_\_\_\_\_)

```

```{r KFold-solution, exercise.reveal\_solution=F}

# set the random seed for reproducibility

set.seed(99)

# Do a k-fold partitioning of the training dataset

k.foldDta <- kfold(sampdata, # the source data to use

k = 5, # number of Folds (here you use 5)

by = sampdata$classvalue) # define how to stratify the folds

# Tabulate the k-fold partitioning dataset by the Land Cover Class to see the number of samples per land cover class to be used in each fold.

table(k.foldDta,

sampdata$classvalue)

```

```{r KFold-check}

grade\_code()

```

With the k-fold partitioning, you can now train and test your CART model five times - five because you have five-folds!

<div class="alert alert-info">

\*\*Your task:\*\*

Train (using those points \*\*NOT\*\* in the k-fold dataset `k.foldDta`) and test (using those points \*\*IN\*\* the k-fold dataset `k.foldDta`) your \*\*CART\*\* model.

For each of the five (5) folds, compute a confusion matrix and store it as a component in a `CART.Kfold.List` list.

Finish by estimating the average confusion matrix over the five folds.

</div>

```{r CARTKfold, exercise=TRUE, exercise.lines=30}

# Kfold Validation for the CART model

CART.Kfold.List <- lapply(1:5, function(k){

# Define the Training data.frame - Observations NOT selected in the k-fold.

train <- sampdata[\_\_\_\_\_\_\_\_\_!= k, ]

# Define the test data.frame - Observations selected in the k-fold.

test <- \_\_\_\_\_\_\_\_\_[k.foldDta == k, ]

# Build a new CART model using the Train dataset

cartMod.kfold.Mod <- rpart(formula = as.factor(classvalue)~., # Define a formula linking the response (land cover classes) to the predictors (bands in Landsat5) - remember to Define the Land cover class as a factor

data = \_\_\_\_\_\_\_\_\_, # Define the object with the response/predictors data - here is the training data.

method = 'class', # Define the Type of model you are building.

cp = 0.01, # Define the minimum level of increase in the complexity parameter

minsplit = 5 #Define the minimum number of observations that must exist in a node for a split to be attempted

)

# Predict the k-fold model on the test data.frame

pclass <- predict(object = \_\_\_\_\_\_\_\_\_, # Define the randomForest with the model.

newdata = \_\_\_\_\_\_\_\_\_, # Define data.frame used to make the prediction - here is the test data.

type = "class" # indicating the type of output

)

# create a table to compare the test-data and the prediction

cbind(Observed = \_\_\_\_\_\_\_\_\_$classvalue,

Predicted = as.numeric(as.vector(pclass)))

})

# Confusion matrix for CART model the kfold validation

# Merge the tables in the list row-wise

CART.Kfold <- do.call("rbind",

CART.Kfold.List)

# Make the matrix a data.frame

CART.Kfold <- data.frame(CART.Kfold)

# Set the column names for the data.frame.

colnames(CART.Kfold) <- c('observed', 'predicted')

# Tabulate the (miss)matches between the observed and predicted test datasets

CART.conmat <- table(\_\_\_\_\_\_\_\_\_)

# Change the name of the classes in the tabulation.

colnames(CART.conmat) <- classdf[match(colnames(CART.conmat),classdf$classvalue1),"nlcdclass"]

rownames(CART.conmat) <- classdf[match(rownames(CART.conmat),classdf$classvalue1),"nlcdclass"]

# Print the confusion matrix

CART.conmat/200

```

```{r CARTKfold-solution, exercise.reveal\_solution=F}

# Kfold Validation for the CART model

CART.Kfold.List <- lapply(1:5, function(k){

# Define the Training data.frame - Observations NOT selected in the k-fold.

train <- sampdata[k.foldDta!= k, ]

# Define the test data.frame - Observations selected in the k-fold.

test <- sampdata[k.foldDta == k, ]

# Build a new CART model using the Train dataset

cartMod.kfold.Mod <- rpart(formula = as.factor(classvalue)~., # Define a formula linking the response (land cover classes) to the predictors (bands in Landsat5) - remember to Define the Land cover class as a factor

data = train, , # Define the object with the response/predictors data - here is the training data.

method = 'class', # Define the Type of model you are building.

cp = 0.01, # Define the minimum level of increase in the complexity parameter

minsplit = 5 #Define the minimum number of observations that must exist in a node for a split to be attempted

)

# Predict the k-fold model on the test data.frame

pclass <- predict(object = cartMod.kfold.Mod, # Define the randomForest with the model.

newdata = test, # Define data.frame used to make the prediction - here is the test data.

type = "class" # indicating the type of output

)

# create a table to compare the test-data and the prediction

cbind(Observed = test$classvalue,

Predicted = as.numeric(as.vector(pclass)))

})

# Confusion matrix for CART model the kfold validation

# Merge the tables in the list row-wise

CART.Kfold <- do.call("rbind",

CART.Kfold.List)

# Make the matrix a data.frame

CART.Kfold <- data.frame(CART.Kfold)

# Set the column names for the data.frame.

colnames(CART.Kfold) <- c('observed', 'predicted')

# Tabulate the (miss)matches between the observed and predicted test datasets

CART.conmat <- table(CART.Kfold)

# Change the name of the classes in the tabulation.

colnames(CART.conmat) <- classdf[match(colnames(CART.conmat),classdf$classvalue1),"nlcdclass"]

rownames(CART.conmat) <- classdf[match(rownames(CART.conmat),classdf$classvalue1),"nlcdclass"]

# Print the confusion matrix

CART.conmat/200

```

```{r CARTKfold-check}

grade\_code()

```

<div class="alert alert-info">

\*\*Your task:\*\*

Using the results of your cross-validation assessment (`CART.conmat`), estimate the miss-classification rate (proportion of observations wrongly classified by the model) for the CART model.

</div>

```{r MissClasRte, exercise=TRUE, exercise.lines=15}

### Miss classification of the CART model

CART.class.error <- sapply(colnames(\_\_\_\_\_\_\_), # The cross-validation confusion matrix

function(i){

sum(\_\_\_\_\_\_\_[i, !colnames(\_\_\_\_\_\_\_)%in%i])

})/apply(\_\_\_\_\_\_\_, 1, sum)

# Print the miss-classification of the RF model

CART.class.error

# Make a barplot of the miss-classification of the RF model

barplot(\_\_\_\_\_\_\_,

main = " CART model k-fold miss-classification rate",

ylim =c(0,1))

```

```{r MissClasRte-solution, exercise.reveal\_solution=F}

### Miss classification of the CART model

CART.class.error <- sapply(colnames(CART.conmat),

function(i){

sum(CART.conmat[i, !colnames(CART.conmat)%in%i])

})/apply(CART.conmat, 1, sum)

# Print the miss-classification of the RF model

CART.class.error

# Make a barplot of the miss-classification of the RF model

barplot(CART.class.error,

main = " CART model k-fold miss-classification rate",

ylim =c(0,1))

```

```{r MissClasRte-check}

grade\_code()

```

## Global Model evaluation

Now that you have trained your classification model (CART), you can use them to make predictions - even as these might not be the best models!. For this, you will use the models to \*classify\* all cells in `landsat5`. It is important that the layer names in the RasterStack object to be classified exactly match those used as predictors in the model. This would be the case if the same Raster object were used (via extract) to obtain the values to fit the model.

<div class="alert alert-info">

\*\*Your task:\*\*

Use your CART model (`cartMod`) to predict land cover maps for the region of interest using the information in `landsat5`.

</div>

```{r GlobMod, exercise=TRUE, exercise.lines=20}

# Use the predict function (as implemented for raster objects and the CART model to generate predictors of land Cover classes for 2011.

CART.pr2011 <- predict(object = \_\_\_\_\_\_\_, # Define the raster used to make the predictions.

model = \_\_\_\_\_\_\_, # Define the model used to make the predictions..

type = 'class' # Define the type of predicted value returned

)

# Plot the CART model PREDICTED map using the image() function and the HEX colours in classdf - remember that there is no class 6 in classdf, and so it does not have a colour.

image(\_\_\_\_\_\_\_,

main = "CART Predicted Land cover classification 2011", # Give the Name

col = c("#5475A8", "#B50000", "#D2CDC0", "#38814E", "#AF963C",NA, "#D1D182", "#FBF65D", "#C8E6F8"))

# Add a legend linking land cover class names to the colours.

legend("topright", # Define the position the legend.

inset = c(-0.24,0), # Define inset distance(s) from the margins.

fill = classdf$\_\_\_\_\_\_\_, # Define the colours of each land cover class.

legend = classdf$\_\_\_\_\_\_\_, # Define each land cover class name.

cex = 0.7, # Define the text size

xpd=NA) # Plot outside the Figure region

```

```{r GlobMod-solution, exercise.reveal\_solution=F}

# Use the predict function (as implemented for raster objects and the CART model to generate predictors of land cover classes for 2011.

CART.pr2011 <- predict(object = landsat5, # Define the raster used to make the predictions.

model = cartMod, # Define the model used to make the predictions..

type = 'class' # Define the type of predicted value returned

)

# Plot the CART model PREDICTED map using the image() function and the HEX colours in classdf - remember that there is no class 6 in classdf, and so it does not have a colour.

image(CART.pr2011,

main = "CART Predicted Land cover classification 2011", # Give the Name

col = c("#5475A8", "#B50000", "#D2CDC0", "#38814E", "#AF963C",NA, "#D1D182", "#FBF65D", "#C8E6F8"))

# Add a legend linking land cover class names to the colours.

legend("topright", # Define the position the legend.

inset = c(-0.24,0), # Define inset distance(s) from the margins.

fill = classdf$classcolor, # Define the colours of each land cover class.

legend = classdf$nlcdclass, # Define each land cover class name.

cex = 0.7, # Define the text size

xpd=NA) # Plot outside the Figure region

```

```{r GlobMod-check}

grade\_code()

```

Focusing on the modes, plotting and counting the misclassified cells (observed != predicted), you can get an overview of the classification error.

<div class="alert alert-info">

\*\*Your task:\*\*

Using your classified image (`CART.pr2011`), show the misclassified cells (colour them \*RED\*) and those classified correctly (colour them \*BLUE\*) by the CART model. for this you will contrast `CART.pr2011` with the observed land cover classification `nlcd2011`.

Also, estimate the proportion of all the cells that are misclassified (wrong predictions) by the CART model.

</div>

```{r ClasErr, exercise=TRUE, exercise.lines=10}

# Use the image function to show the cells that are misclassified (Predicted != Observed) and classified correctly (Predicted == Observed) based on the CART model

image(\_\_\_\_\_\_\_ == \_\_\_\_\_\_\_, # Which cells are correctly classified (Predicted == Observed)

main = "CART misclassified cells",

col = c("red", "blue")) ## red is for areas misclassified

# Estimate the proportion of misclassified cells (Predicted != Observed)/Ncells for the CART model.

sum((\_\_\_\_\_\_\_!= \_\_\_\_\_\_\_)[])/ncell(nlcd2011) # sum of miss-classified cells (Predicted != Observed)

```

```{r ClasErr-solution, exercise.reveal\_solution=F}

# Use the image function to show the cells that are misclassified (Predicted != Observed) and classified correctly (Predicted == Observed) based on the CART model

image(CART.pr2011 == nlcd2011, # Which cells are correctly classified (Predicted == Observed)

main = "CART misclassified cells",

col = c("red", "blue")) ## red is for areas misclassified

# Estimate the proportion of misclassified cells (Predicted != Observed)/Ncells for the CART model.

sum((CART.pr2011!= nlcd2011)[])/ncell(nlcd2011) # sum of miss-classified cells (Predicted != Observed)

```

```{r ClasErr-check}

grade\_code()

```

Although visual inspections are OK and counts of miss-classified cells show you the quality of the classification. You also want to assess the per-class error rate (the number of observations in a class the algorithm does misclassify)

<div class="alert alert-info">

\*\*Your task:\*\*

Estimate the confusion matrix for the raster’s predicted using the CART model for this you will tabulate the cells correctly and incorrectly classified by your model. That means contrasting your predictions `CART.pr2011` and observations `nlcd2011`.

Here the table will be build using the \*"class names"\* instead of the numeric codes.

</div>

```{r FullRAstAcc, exercise=TRUE, exercise.lines=15}

# Use the table() function to estimate the confusion matrix for the CART model

# The table is based on names and not numeric codes

CART.ConfMtx <- table(classdf$nlcdclass[match(\_\_\_\_\_\_\_\_\_[], classdf$classvalue1)], # Names for each cell in the Observed raster

classdf$nlcdclass[match(\_\_\_\_\_\_\_\_\_[], classdf$classvalue1)] # Names for each cell in the Predicted raster

)

# Define the per-class error rate [proportion of the cells for a given land cover class correctly classified]

CARTclass.error <- sapply(colnames(\_\_\_\_\_\_\_\_\_), # Do this by column manes in `CART.ConfMtx`

function(i){

sum(\_\_\_\_\_\_\_\_\_[i, !colnames(\_\_\_\_\_\_\_\_\_)%in%i]) # total of correctly classified per class

})

CARTclass.error/apply(\_\_\_\_\_\_\_\_\_, 1, sum) #Proportion of all observation in a class correctly classified

```

```{r FullRAstAcc-solution, exercise.reveal\_solution=F}

# Use the table() function to estimate the confusion matrix for the CART model

# The table is based on names and not numeric codes

CART.ConfMtx <- table(classdf$nlcdclass[match(nlcd2011[], classdf$classvalue1)], # Names for each cell in the Observed raster

classdf$nlcdclass[match(CART.pr2011[], classdf$classvalue1)] # Names for each cell in the Predicted raster

)

# Define the per-class error rate [proportion of the cells for a given land cover class correctly classified]

CARTclass.error <- sapply(colnames(CART.ConfMtx), # Do this by column manes in `CART.ConfMtx`

function(i){

sum(CART.ConfMtx[i, !colnames(CART.ConfMtx)%in%i]) # total of correctly classified per class

})

CARTclass.error/apply(CART.ConfMtx, 1, sum) #Proportion of all observation in a class correctly classified

```

```{r FullRAstAcc-check}

grade\_code()

```

## Final points

Supervised classification is the technique most often used for the quantitative analysis of remote sensing image data. At its core is the concept of segmenting the spectral domain into regions that can be associated with the ground cover classes of interest to a particular application.

For example, you could use the PCA axes instead of the raw data in the Landsat image. Alternatively, you could use more points to develop your model (right now, you use 200 points that are not even 1% of the available data - more data gives you a better model but requires more computer power!!). I leave these tasks to the interested student! This exercise gave you a guide of the procedure to do so, and although your final model is not the best, it is the starting point for developing this type of analysis.