Step 4.2 Building final model

Flat and Hierarchical modeling strategies

Diane ESPEL

2025-06-19

Contents

1 Objectives **1**

2 Script explanation **1**

2.1 **Clean environment and graphics** 1

2.2 **Load required packages** 1

2.3 **Create functions** 2

2.4 **Define Global Variables** 3

2.5 **Set working directory** 4

2.6 **Modelling** 4

1 Objectives

This script aims to train and evaluate final Random Forest habitat classification models at each level of a hierarchical typology, using the most frequently selected hyperparameters from previous tuning iterations (see step 4.1) for both flat and hierarchical modeling.

2 Script explanation

2.1 Clean environment and graphics

```r
rm(list = ls())  # Clear all objects from the R environment to start fresh
graphics.off()  # Close all graphics devices (if any plots are open)
```

2.2 Load required packages

```r
library(caret)  # Functions for model training and evaluation
library(randomForest)  # Random Forest classification and regression
library(e1071)  # Provides tuning functions for model optimization
library(dplyr)  # Data manipulation (select, filter, mutate, etc.)
library(stringr)  # String operations (not heavily used here)
library(pROC)  # For ROC curve computation and AUC metrics
library(stats)  # Base statistical functions
```

2.3 Create functions

We need to define

- A `"get_mode()"` function that calculates the mode of a given vector—that is, it returns the most frequently occurring value in the vector.

  To achieve this:

  - It first extracts all the unique values (`uniqv`) from the input vector `v`.
  - Then, it counts how many times each unique value appears in the vector.
  - Finally, it returns the unique value with the highest count, which represents the mode.

- A `"compute_macro_roc()"` function that returns a macro-average Receiver Operating Characteristic (ROC) curve from a set of multiple binary ROC curves.

  This is particularly relevant when evaluating multi-class or multi-label classification models, where each label is treated independently, and an overall performance summary is desired. The function operates by **interpolating all individual ROC curves** onto a common grid of values for the false positive rate (**1 - specificity**), and then averaging the corresponding sensitivity values to create a single, smooth representative curve.

  To achieve this:

  - The process begins by creating a regular grid of `n_points` values ranging from 0 to 1 along the x-axis, which represents **1 - specificity**.
  - For each binary ROC curve (contained in `roc_scores$rocs`), the **sensitivity** (true positive rate) and **1 - specificity** values are extracted and cleaned to remove duplicates.
  - Linear interpolation (`approx()`) is applied to align the sensitivity values with the common grid. These interpolated curves are stored and later averaged pointwise to compute the macro-averaged sensitivity.
  - The corresponding mean specificity is derived as 1 - x_vals, assuming symmetry in how the specificity grid was defined.
  - The function returns a list containing the interpolated x-axis grid (`x_vals`), the averaged sensitivity curve (`mean_sensitivity`), and the averaged specificity curve (`mean_specificity`). This output can then be used for plotting the macro-ROC curve or for further performance analysis. This approach provides a robust and interpretable way to assess the overall discriminative power of classification models across multiple classes or species.

```r
# Function to compute the mode (most frequent value) of a vector
get_mode <- function(v) {
    uniqv <- unique(v)  # Get all unique values in the vector
    tab <- tabulate(match(v, uniqv))  # Count occurrences of each unique value
    uniqv[which.max(tab)]  # Return the value with the highest count
}
```

```r
# Function to compute a macro-averaged ROC curve over multiple binary ROC
# curves
compute_macro_roc <- function(roc_scores, n_points = 100) {

    # Create a grid for 1 - specificity (x-axis)
    x_vals <- seq(0, 1, length.out = n_points)

    # Lists to store interpolated curves
    sensitivity_list <- list()
    specificity_list <- list()

    # Loop through each binary ROC curve pair
    for (pair in seq_along(roc_scores$rocs)) {
        roc_pair <- roc_scores$rocs[[pair]]

        for (j in 1:2) {
            roc_obj <- roc_pair[[j]]

            # Get 1 - specificity (x) and sensitivity (y)
            x <- 1 - roc_obj$specificities
            y <- roc_obj$sensitivities

            # Remove duplicates to avoid issues in approx
            dedup <- !duplicated(x)
            x <- x[dedup]
            y <- y[dedup]

            # Interpolate sensitivities on common x grid
            sens_interp <- approx(x, y, xout = x_vals, ties = "mean", rule = 2)$y
            sensitivity_list[[length(sensitivity_list) + 1]] <- sens_interp

            # Corresponding specificity is just 1 - x_vals
            specificity_list[[length(specificity_list) + 1]] <- 1 - x_vals
        }
    }

    # Compute mean curves
    mean_sensitivity <- rowMeans(do.call(cbind, sensitivity_list), na.rm = TRUE)
    mean_specificity <- rowMeans(do.call(cbind, specificity_list), na.rm = TRUE)

    # Return as a list (to plot or reuse)
    return(list(mean_sensitivity = mean_sensitivity, mean_specificity = mean_specificity,
        x_vals = x_vals))
}
```

2.4 Define Global Variables

Note: A global variable is a variable defined outside of any function. This means the variable is accessible from any part of the code, including inside functions. A global variable retains its value throughout the execution of the R script unless it is explicitly modified in the code.

It is important to define at a minimum:

- the `"District"`: the archipelago of interest (e.g. "CRO" for Crozet archipelago)
- the `"Island"`: the island within the archipelago of interest (e.g. "POS" for Possession island)

- the "Satellite1": the satellite name for multispectral imagery
- the "Year1": the year of imagery acquisition
- the "maxTypoLevel": the maximum typology level

```
District = "CRO"  # 3-letter code for archipelago (e.g. Crozet)
Island = "POS"  # 3-letter code for island (e.g. Possession)
Satellite1 = "Pleiades"  # satellite name of multispectral imagery
Year1 = "2022"  # acquisition year of multispectral imagery
maxTypoLevel = 4  # Define maximum typology level
```

2.5 Set working directory

You must define a general root directory ("localscratch") that serves as the base path for your input and output data. This directory should point to the local environment where:

- input final learning data file is located under "data/Learning_data/NewTypo"
- input best hyperparameters informations are located under "results/Model/Tuned_model"
- outputs will be saved under "results/Model/Final_model"

```
# Base local path (customize to your local environment)
localscratch = paste0("/scratch/despel/CARTOVEGE/")
# localscratch = paste0('your_local_path/')

# Path to open learning data
open_learning_new_path = paste0(localscratch, "data/Learning_data/NewTypo")

# Path to open tuned models
open_tuned_model_path = paste0(localscratch, "results/Model/Tuned_model")

# Path to save results
save_final_model_path = paste0(localscratch, "results/Model/Final_model")
```

##**Load and prepare learning data** ::: {align="justify"} This section loads the final dataset, cleans it, and prepares it for modeling:
Primary typology labels are removed and new labels are renamed

:::

```
# Open learning data
FILE1 = paste0(open_learning_new_path, "/Final_learning_plots_", District, "_", Island,
    "_", Satellite1, "_", Year1, "_ALL_SOURCES_EPSG32739.csv")
learning_data <- read.csv(FILE1, sep = ";", dec = ".", stringsAsFactors = FALSE)

# Remove old habitat columns (to keep only newTyppology columns)
learning_data <- learning_data %>%
    select(-matches("^Hab_L[1-4]$"))

# Rename *_corr columns to original names
learning_data <- learning_data %>%
    rename_with(.fn = ~gsub("_corr$", "", .), .cols = matches("^Hab_L[1-4]_corr$"))
```

2.6 Modelling

This section defines and executes two modelling strategies: "**FLAT**" (non-hierarchical) and "**HIERARCHICAL**", looping through each classification level (l) from 1 to maxTypoLevel.

For each strategy and level:

1. Parameter Tuning Retrieval: It reads the top-performing hyperparameter configurations (e.g., `ntree, mtry, nodesize`) from a tuning summary (`ParamSummary`), selecting the most frequently occurring values (with `get_mode()`)

2.Final Model Training: It trains a final randomForest model using the selected hyperparameters on the full dataset, excluding ID, coordinates, and the response variable.

3.Model Saving: The trained model is saved in both .Rdata and .rds formats for reproducibility and future use.

4.Variable Importance Evaluation:

- Saves the variable importance table
- Generates and saves importance plots

5.Predictions and Evaluation:

- Predicts habitat classes on the entire learning dataset.
- Saves a table comparing observed vs. predicted classes.
- Computes the confusion matrix and class-wise performance metrics (e.g., Sensitivity, Specificity).
- Calculates global metrics: overall accuracy, kappa coefficient, and mean AUC (Area Under the Curve).
- Plots and saves a macro-averaged ROC curve to visually assess classification quality.

6.Summary of Metrics: Key evaluation metrics (OOB error, accuracy, kappa, AUC) are written to a CSV for final reporting.

This structured process ensures that for each modelling strategy and habitat classification level, the model is optimally tuned, trained, evaluated, and documented thoroughly.

```r
# Define Modeling strategy
type_model_list=c("FLAT","HIERARCHICAL")


# Loop through model types
for (type_model in type_model_list){

  #type="FLAT" #debug
  print(paste0("Modeling strategy: ",type_model))

  # Train and run a final model for each level of typology
  for (l in seq (1:maxTypoLevel)){

    print(paste0("Working with habitat classification level: ", l))

    #  Define specific folder for habitat level
    TuningLevelFolder=paste0(open_tuned_model_path,"/","Hab_L",l)
    dir.create(TuningLevelFolder, showWarnings = FALSE)

    # Identify key column indices
    iid = which(colnames(learning_data) == "ID")
    ihab = which(colnames(learning_data) == paste0("Hab_L", l))
    ix = which(colnames(learning_data) == "xcoord_m")
```

```r
iy = which(colnames(learning_data) == "ycoord_m")


# Training Final model ----------------------------------------------------

# Load parameter tuning summary
FILE2 = paste0(TuningLevelFolder,
↪ "/","AllParamMetrics_tuned_RF_",type_model,"_model_",District,"_",
               Island,"_",Satellite1,"_level_",l,".csv")
ParamSummary=read.csv(FILE2, sep = ";", dec = ".", stringsAsFactors = FALSE)

# Keep top N tuning runs with highest performance
niter <- 10
best_iter <- ParamSummary[order(ParamSummary$ValPerformance), ][1:niter, ]

# Extract most frequent values (mode) for each hyperparameter
best_ntree <- get_mode(best_iter$ntree)
best_mtry <- get_mode(best_iter$mtry)
best_nodesize <- get_mode(best_iter$nodes)

cat("Best hyperparameters (top ", niter, " runs):\n")
cat("ntree =", best_ntree, "\n")
cat("mtry =", best_mtry, "\n")
cat("nodesize =", best_nodesize, "\n")

# Train a final model with the best hyperparameters
FinalModel <- randomForest(
  x = learning_data[, -c(iid, ihab, ix, iy)],   # Exclude ID, target, coords
  y = as.factor(learning_data[, ihab]),      # Target habitat class
  ntree = best_ntree,
  mtry = best_mtry,
  nodesize = best_nodesize,
  importance = TRUE      # Save variable importance
)

print(FinalModel)  # Show model summary

# Create the folder for each level of classification save the final model
FinalLevelFolder=paste0(save_final_model_path,"/","Hab_L",l)
dir.create(FinalLevelFolder, showWarnings = FALSE) # ShowWarnings=F to remove warnings message
 ↪  if file already exists

# Save FinalModel
save(FinalModel, file = paste0(FinalLevelFolder, "/Final_RF_",type_model,"_model_", District,
 ↪  "_",
                               Island, "_", Satellite1, "_level_", l, ".Rdata"))
saveRDS(FinalModel, file = paste0(FinalLevelFolder, "/Final_RF_",type_model,"_model_",
 ↪  District, "_",
                                  Island, "_", Satellite1, "_level_", l, ".rds"))

# Evaluate global variable importance (dataframe and plot) ----------------

print("Variable importance evaluation")

# Dataframe of variable importance (mean indices and importance for each class)
NOMcsv=paste0(FinalLevelFolder,"/","VarImportance_Final_RF_",type_model,"_model_",District,"_",
              Island,"_",Satellite1,"_level_",l,".csv")
VarImportance=as.data.frame(importance(FinalModel,scale=T)) # calculating variable importance
↪ for FinalModel and scales values from 0 to 100
```

```r
write.table(VarImportance,NOMcsv,sep=";",dec=".",row.names = FALSE)


# Plot of variable importance (mean indices)
NOMpng=paste0(FinalLevelFolder,"/","VarImportance_Final_RF_",type_model,"_model_",District,"_",
              Island,"_",Satellite1,"_level_",l,".png")
png(file = NOMpng, width = 500, height = 500)
p=varImpPlot(FinalModel, pch = 19, col = "black",main="Variable importance",cex=1)
print(p)
dev.off()

NOMsvg=paste0(FinalLevelFolder,"/","VarImportance_Final_RF_",type_model,"_model_",District,"_",
              Island,"_",Satellite1,"_level_",l,".svg")
svg(file = NOMsvg)
p=varImpPlot(FinalModel, pch = 19, col = "black",main="Variable importance",cex=1)
print(p)
dev.off()


# Predictions on learning_data   ------------------------------------------

# Prediction on all learning data
print("Model prediction on the entire dataset")
yPred <- randomForest::predict(FinalModel, newdata=learning_data[, -c(iid, ihab, ix, iy)],type
↪ = "response")   # predict class probabilities

# Save observations vs predictions
ObsPred_df=learning_data[,c(iid,ix,iy,ihab)]
col_name <- paste("Ypred")
ObsPred_df[[col_name]] <- yPred

↪ FILE4=paste0(FinalLevelFolder,"/","Comparison_Obs_Pred_Final_RF_",type_model,"_model_",District,
             "_",Island,"_",Satellite1,"_level_",l,".csv")
write.table(ObsPred_df,file=FILE4, sep=";",dec=".",row.names = FALSE)

# Final model Evaluation  -------------------------------------------------

## Confusion matrix -------------------------------
print("Confusion matrix")
yObs=as.factor(learning_data[, ihab])
ConfMatrix<- confusionMatrix(yObs,yPred,positive = NULL, dnn = c("Reference","Prediction"))
print(ConfMatrix)

↪ FILE5=paste0(FinalLevelFolder,"/","ConfusionMatrix_Final_RF_",type_model,"_model_",District,"_",
             Island,"_",Satellite1,"_level_",l,".csv")
write.table(ConfMatrix[["table"]],file=FILE5,sep=";",dec=".",row.names = FALSE)


# Relative metrics for each class  -----------------
print("Sensitivity, Specificity, etc. ")
StatsByClass=ConfMatrix[["byClass"]]
FILE6=paste0(FinalLevelFolder,"/","StatsByClass_Final_RF_",type_model,"_model_",District,"_",
             Island,"_",Satellite1,"_level_",l,".csv")
write.table(StatsByClass,file=FILE6,sep=";",dec=".",row.names = FALSE)
print(StatsByClass)


## General (mean) metrics  --------------------------
```

```
  # OOB rate
  print("OOB rate")
  conf=FinalModel$confusion[,-ncol(FinalModel$confusion)]  #Just remove the "class.error" column
↪ from the confusion matrix
  oob=1-sum(diag(conf))/sum(conf) # (1 - (TP +TN)/ Total obs)*100
  print(oob)

  # Overall accuracy
  print("Overall accuracy")
  OA=ConfMatrix[["overall"]][["Accuracy"]] # = nb correct prediction/ nb total of predictions
  print(OA)

  # Kappa coefficient
  print("Kappa coefficent")
  kappa=ConfMatrix[["overall"]][["Kappa"]] #extract kappa coefficient
  print(kappa)

  # mean AUC
  print("mean AUC -area under the ROC curve")
  yPred_proba <- randomForest::predict(FinalModel, newdata=learning_data[, -c(iid, ihab, ix,
↪ iy)], type = "prob")
  roc_scores <- multiclass.roc(response = yObs, predictor = yPred_proba) # compute ROC scores per
↪ pair
  AUC=roc_scores[["auc"]][1]
  auc_list=append(auc_list,AUC) #fill the AUC list
  print(AUC) #the more AUC is close to 1 the more the model is better


  # mean ROC
  print("Macro-averaged ROC curve")
  roc_result <- compute_macro_roc(roc_scores)


↪ NOMpng=paste0(FinalLevelFolder,"/","Mean_Roc_curve_Final_RF_",type_model,"_model_",District,"_",
                Island,"_",Satellite1,"_level_",l,".png")
  par(bty = "n")
  png(file = NOMpng, width = 500, height = 500)
  p=plot(1 - roc_result$mean_specificity, roc_result$mean_sensitivity,
       type = "l", col = "blue", lwd=2,
       main = "Macro-Averaged ROC Curve",
       xlab = "1 - Specificity", ylab = "Sensitivity")
  abline(a = 0, b = 1, col = "gray", lty =2, lwd = 0.5) # Add a grey line  x=y
  text(x = 0.8, y = 0.2, labels = paste0("AUC : ",round(AUC,7)), col = "black", cex = 1.2) # add
   ↪  "AUC" label
  print(p)
  dev.off()

  # Stack all general statistics
  print("Final metrics summary")
  Stats=c(oob,OA,kappa,AUC)
  names(Stats)=c("OOB_rate","Overall_accuracy","Kappa_coefficient","AUC")
  print(Stats)
  FILE7=paste0(FinalLevelFolder,"/","AllMetrics_Final_RF_",type_model,"_model_",District,"_",
               Island,"_",Satellite1,"_level_",l,".csv")
  write.table(Stats,file=FILE7, sep=";",dec=".",row.names = FALSE)


} # End of typology level loop
```

```
}
```