Step 5.1 Generating habitat maps: spatial predictions

5.1.b Hierarchical modeling strategy

Diane ESPEL

2025-06-20

Contents

1 Objectives

This script automates the generation of habitat classification maps across multiple years by applying a pre-trained Random Forest model to multispectral raster stacks. The script loops through different typology levels (i.e., classification granularity) and temporal raster datasets, generating:

Its primary objective is to generate final habitat classification maps by leveraging optimized models for each typology level, ranging from level 1 to the maximum specified level, providing a single-date raster of raw predictions (`"Final_RF_....tif"`);

The script applies a smoothing filter to enhance map quality (`"Smoothed_final_RF_....tif"`) and provides simplified vector shapefiles (`"Smoothed_simplified_final_map_RF_.....shp"`) for subsequent analysis and visualization.

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(terra)  # For handling spatial raster data and geospatial analysis
library(ggplot2)  # For creating advanced and customizable data visualizations
library(randomForest)  # For building Random Forest models for classification/regression
library(viridis)  # Provides color palettes optimized for perceptual uniformity, useful in
↪  plots/maps
library(sf)  # For handling spatial vector data (simple features) in R
library(dplyr)  # For efficient data manipulation (filter, select, mutate, etc.)
library(purrr)  # For functional programming tools, simplifies working with lists and iteration
library(rmapshaper)  # For simplifying and processing spatial vector data (e.g. ms_simplify for
↪  reducing complexity)
library(stats)  # Base R stats package; provides functions like `focal` for neighborhood operations
↪  on rasters
```

2.3 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 `"Res1"`: the resolution of the multispectral imagery acquisition
- the `"maxTypoLevel"`: the maximum typology level
- the `"type_model"`: the adopted modeling strategy (i.e. flat or hierarchical)

```r
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
Res1 = "50cm"  # spatial resolution of multispectral imagery
maxTypoLevel = 4  # maximum typology level
type_model = "HIERARCHICAL"  # modeling strategy (FLAT or HIERARCHICAL)
```

2.4 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 model is located under `"results/Model/Final_model"`
- input raster stacks are located under `"data/raster/Cut_image/"`
- output predictions will be saved under `"results/Predictions"`

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

# Path to open Final model
open_final_model_path = paste0(localscratch, "results/Model/Final_model")
```

```r
# Path to open raster stack
open_cut_raster_path = paste0(localscratch, "data/raster/Cut_image/")

# Path to save results
save_predictions_path = paste0(localscratch, "results/Predictions")
```

##**Predict habitat classes**

The core processing loop iterates over each habitat classification level, loading the corresponding final Random Forest model trained with the selected hyperparameters.

- For each typology level (l), the script iterates over yearly raster stacks that contain spectral and topographic variables (e.g., Green, NIR, GRVI, DTM). For levels higher than 1, it also loads previously predicted habitat maps from lower levels (e.g., Hab__L1, Hab__L2, etc.) and combines them with spectral layers to enrich the feature space. This hierarchical input construction allows the model to leverage both raw environmental data and prior classification outputs. All layers are stacked and renamed accordingly to ensure consistency before being passed to the model for prediction.
- The combined raster stack is then used to generate predictions using the loaded model (`FinalModel`). Predicted maps are saved as GeoTIFF raster files and visualized in PNG format using perceptually uniform color palettes.
- To reduce noise and small isolated classification errors, a modal filter (a type of spatial smoothing) with a dynamically calculated window size is applied. The modal filter replaces each pixel's value by the most frequent (`modal`) value within a square neighborhood around it:
  - First, the spatial resolution of the raster is obtained using `res(raster_pred)`. This gives the size of one pixel in map units (e.g., meters).
  - The smoothing window is defined by a `radius` of 10 map units. To translate this radius into pixels, the code divides the radius by the pixel `resolution`.
  - The window size in pixels is rounded up (`ceiling()`) to ensure the neighborhood fully covers the intended radius.
  - Since focal filters in R require an odd-sized window to have a central pixel (for proper centering), the window size is adjusted by adding 1 if it is even.
  - The final neighborhood matrix used for filtering is of size (`2 * window_size + 1`) in both rows and columns, creating a square window centered on each pixel.
  - The `focal()` function applies the modal filter, which replaces each pixel's value by the most common class label within the window.
  - The smoothed predictions are also saved as raster files.
- Finally, the smoothed raster predictions are converted to vector polygons, reprojected to a specified coordinate reference system. and simplified to reduce geometric complexity without losing essential shape information. The `ms_simplify()` function is used to simplify the geometry of the spatial vector object. The simplification applies the Douglas-Peucker algorithm (`method = "dp"`), which reduces the number of vertices while preserving the overall shape of the polygons. The parameter `keep = 1` indicates that all vertices are retained, so no actual simplification occurs here, but this can be adjusted to reduce complexity if needed. Setting `explode = TRUE` separates multipart geometries into individual features, making subsequent processing easier. Additionally, `drop_null_geometries = TRUE` ensures that any empty or invalid geometries created during the simplification are removed, resulting in a cleaner and more manageable spatial dataset.
- Surface area and perimeter metrics are computed and appended as attributes. The resulting vector data are saved as shapefiles, completing the workflow for each habitat level and year.

Throughout, memory management is considered by removing large objects after use, facilitating efficient processing of potentially large spatial datasets.

```r
# Define a list of all raster files
raster_files <- list.files(open_cut_raster_path, pattern = paste0(District, "_",
    Island, "_Final_raster_stack_.*_cut\\.TIF$"), full.names = TRUE)


for (l in seq_len(maxTypoLevel)) {

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

    # Define specific folder for model path
    ModelFolder = paste0(open_final_model_path, "/", "Hab_L", l)
    set.seed(72143 * l)

    # Load final model
    FILE2 = paste0(ModelFolder, "/", "Final_RF_", type_model, "_model_", District,
        "_", Island, "_", Satellite1, "_level_", l, ".rds")
    FinalModel <- readRDS(FILE2)

    # Loop on each raster/year ------------------------------------------------

    for (raster_file in raster_files) {

        message("File treatment: ", basename(raster_file))

        # Extract Year from file name
        file_base <- basename(raster_file)
        parts <- strsplit(file_base, "_")[[1]]
        Year <- parts[5]

        # Load raster stack
        message("Loading raster stack : ", file_base)

        raster_sat <- terra::rast(raster_file)
        names(raster_sat) <- c("G", "NIR", "GRVI", "Greenness", "BSI", "NDWI", "DTM")

        if (l == 1) {
            raster_final = raster_sat
            names(raster_final) <- names(raster_sat)
        } else {
            # Load also Hab_L-i predicted raster(s)
            if (l == 2) {
                raster_L1 <- terra::rast(paste0(save_predictions_path, "/", "Hab_L",
                  l - 1, "/", "Final_map_RF_", type_model, "_model_", District, "_",
                  Island, "_", Satellite1, "_", Year, "_", Res1, "_level_", l - 1,
                  ".TIF"))
                raster_final = c(raster_L1, raster_sat)

            }
            if (l == 3) {
                raster_L1 <- terra::rast(paste0(save_predictions_path, "/", "Hab_L",
                  l - 2, "/", "Final_map_RF_", type_model, "_model_", District, "_",
                  Island, "_", Satellite1, "_", Year, "_", Res1, "_level_", l - 2,
                  ".TIF"))
                raster_L2 <- terra::rast(paste0(save_predictions_path, "/", "Hab_L",
                  l - 1, "/", "Final_map_RF_", type_model, "_model_", District, "_",
                  Island, "_", Satellite1, "_", Year, "_", Res1, "_level_", l - 1,
                  ".TIF"))
                raster_final = c(raster_L1, raster_L2, raster_sat)
            }
            if (l == 4) {
                raster_L1 <- terra::rast(paste0(save_predictions_path, "/", "Hab_L",
                  l - 3, "/", "Final_map_RF_", type_model, "_model_", District, "_",
                  Island, "_", Satellite1, "_", Year, "_", Res1, "_level_", l - 3,
                  ".TIF"))
                raster_L2 <- terra::rast(paste0(save_predictions_path, "/", "Hab_L",
                  l - 2, "/", "Final_map_RF_", type_model, "_model_", District, "_",
                  Island, "_", Satellite1, "_", Year, "_", Res1, "_level_", l - 2,
                  ".TIF"))
                raster_L3 <- terra::rast(paste0(save_predictions_path, "/", "Hab_L",
                  l - 1, "/", "Final_map_RF_", type_model, "_model_", District, "_",
                  Island, "_", Satellite1, "_", Year, "_", Res1, "_level_", l - 1,
                  ".TIF"))
                raster_final = c(raster_L1, raster_L2, raster_L3, raster_sat)
            }

            hab_names <- paste0("Hab_L", 1:(l - 1))
            sat_names <- names(raster_sat)
            names(raster_final) <- c(hab_names, sat_names)
        }

        # Prediction on raster stack
        raster_pred <- terra::predict(raster_final, FinalModel)

        # Define specific folder for spatial predictions
        MapFolder = paste0(save_predictions_path, "/", "Hab_L", l)
        dir.create(MapFolder, showWarnings = FALSE, recursive = TRUE)

        # Save spatial predictions (.tif)
        NOMtiff = paste0(MapFolder, "/", "Final_map_RF_", type_model, "_model_",
            District, "_", Island, "_", Satellite1, "_", Year, "_", Res1, "_level_",
            l, ".TIF")
        writeRaster(raster_pred, NOMtiff, overwrite = TRUE)

        # Save spatial predictions (.png)
        NOMpng = paste0(MapFolder, "/", "Final_map_RF_", type_model, "_model_", District,
            "_", Island, "_", Satellite1, "_", Year, "_", Res1, "_level_", l, ".png")
        png(file = NOMpng, width = 1000, height = 1000)
        plot(raster_pred, col = viridis(length(unique(values(raster_pred)))), main = paste0("Carte
         ↪  finale des habitats niveau ",
            l), axes = FALSE, cex.lab = 1.5)
        dev.off()

        # Apply modal filter with a square window of size (2*window_size + 1)
        resolution <- res(raster_pred)  # Compute resolution of the raster
        radius <- 10  # Define the radius of the smoothing window (in map units)
        window_size <- radius/resolution  # Calculate window size in pixels
        window_size <- ceiling(window_size)  # Round up to ensure window fully covers the radius
        window_size <- ifelse(window_size%%2 == 0, window_size + 1, window_size)  # Ensure window
↪   size is odd (required for a centered focal filter)
        filtered_raster_pred <- focal(raster_pred, w = matrix(1, nrow = window_size[1] *
            2 + 1, ncol = window_size[2] * 2 + 1), fun = modal)  # apply smooth filter
        rm(raster_pred)  # free memory

        # Save smooth map (.tif)
        NOMtiff2 = paste0(MapFolder, "/", "Smoothed_final_map_RF_", type_model, "_model_",
            District, "_", Island, "_", Satellite1, "_", Year, "_", Res, "_level_",
            l, ".TIF")
        writeRaster(filtered_raster_pred, NOMtiff2, overwrite = TRUE)

        # Convert raster to vector
        vector_map <- terra::as.polygons(filtered_raster_pred, fun = function(x) {
            x > 0
        })
        vector_layer <- terra::project(vector_map, "EPSG:32739")
        rm(filtered_raster_pred)

        # Simplify geometry
        vector_layer_sf <- st_as_sf(vector_layer)
        vector_layer_sf_simplified <- ms_simplify(vector_layer_sf, keep = 1, method = "dp",
            explode = TRUE, drop_null_geometries = TRUE)

        # Add surface and perimeter
        vector_layer_sf_simplified$Surface <- st_area(vector_layer_sf_simplified)
        vector_layer_sf_simplified$Perimeter <- st_length(st_boundary(vector_layer_sf_simplified))

        # Save shapefile
        FILE3 = paste0(MapFolder, "/", "Smoothed_simplified_final_map_RF_", type_model,
            "_model_", District, "_", Island, "_", Satellite1, "_", Year, "_", Res,
            "_level_", l, ".shp")
        st_write(vector_layer_sf_simplified, FILE3, driver = "ESRI Shapefile", append = FALSE)

        # Cleanup
        rm(vector_map, vector_layer, vector_layer_sf, vector_layer_sf_simplified)

    }  # End of raster files loop

}  # End of typology level loop
```