Step 1.2 Creating photo-interpreted plots

1.2.3 Creating and sorting photo-interpreted plots

Diane ESPEL

2025-10-31

Contents

1 Objectives

This script aims to create standardized squared-plots (quadrats) around a set of georeferenced points derived from photo-interpretation.

It begins by loading and harmonizing three spatial layers representing inaccessible areas, surface points, and randomly distributed points. After merging them, it generates square polygons (4x4 meters) centered on each point, assigning relevant attributes such as source, date, and habitat classification. To ensure spatial independence of plots, overlapping quadrats are removed by retaining only the first in any intersecting group.

The final output consists of two shapefiles: one containing the non-overlapping quadrat polygons, and another with their centroids, both saved in the UTM Zone 39S coordinate reference system.

---

2 Script explanation

2.1 Clean environment and graphics

```
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

```
library(sp)       # For spatial data structures (legacy)
library(sf)       # For modern simple features spatial data handling
library(terra)    # For raster and vector data manipulation (newer alternative to raster)
library(dplyr)    # For bind_rows() function
```

2.3 Create functions

We define a `create_quadrat()` function: This function creates a square polygon based on given bounding box coordinates. It constructs a closed polygon by defining the four corners (top-left, top-right, bottom-right, bottom-left) and returns a geometry object using `st_polygon()`.

```
# Function to build a square polygon from bounding box coordinates
create_quadrat <- function(xmin, xmax, ymin, ymax) {
  st_polygon(list(matrix(c(
    xmin, ymax,  # Top-left (NW)
    xmax, ymax,  # Top-right (NE)
    xmax, ymin,  # Bottom-right (SE)
    xmin, ymin,  # Bottom-left (SW)
    xmin, ymax   # Close polygon (back to NW)
  ), ncol = 2, byrow = TRUE)))
}
```

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 `"Year1"`: the year of imagery acquisition (i.e. the photo-interpretation year)

```
District='CRO' # 3-letter code for the archipelago
Island='POS'   # 3-letter code for the island
Year1="2022"   # Year of imagery acquisition
```

2.5 Set working directory

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

- input polygon map of observed habitats is located under `"data/vector/Observed_map/PrimaryTypo"`
- outputs of merged photointerpreted points will be saved under `"data/vector/Observed_map/PrimaryTypo"`

```
# Base local path (customize to your local environment)
#localHOME = paste0("your_local_path/")
localHOME=paste0("/home/genouest/cnrs_umr6553/despel/CARTOVEGE/")

# Path to open input  photo-interpreted points
open_ObsMap_path=paste0(localHOME,"data/vector/Observed_map/PrimaryTypo")

# Path to save photo-interpreted plots
save_plots_path=paste0(localHOME,"data/vector/Plots/PrimaryTypo")
```

2.6 Read shapefiles

```
# Define full paths to the shapefiles to be loaded
FILE1 <- paste0(open_ObsMap_path, paste0("/","Corrected_observed_map_", District, "_", Island, "_",
 ↪  Year1, "_Poles_of_inaccessibility_EPSG32739.shp"))
FILE2 <-  paste0(open_ObsMap_path, paste0("/","Corrected_observed_map_", District, "_", Island,
 ↪  "_", Year1, "_Pts_on_surface_EPSG32739.shp"))
FILE3 <-  paste0(open_ObsMap_path, paste0("/","Corrected_observed_map_", District, "_", Island,
 ↪  "_", Year1, "_Random_points_EPSG32739.shp"))


# Load the shapefiles
PolesInaccess <- st_read(FILE1)
PtOnSurface <- st_read(FILE2)
RandomPts <- st_read(FILE3)
```

2.7 Merge points and harmonize columns across layers

The three data tables (`"PtOnSurface"`, `"PolesInaccess"`, `"RandomPts"`) are merged into a single table called `"all_points"`, retaining only the columns they have in common.

```
# Identify common columns shared between all three layers
common_columns <- Reduce(intersect, list(names(PolesInaccess ), names(PtOnSurface),
 ↪  names(RandomPts)))
cat("Common columns:\n")
print(common_columns)

# Subset each layer to retain only the common columns
PolesInaccess  <- PolesInaccess [, common_columns]
PtOnSurface <- PtOnSurface[, common_columns]
RandomPts <- RandomPts[, common_columns]

# Merge all point layers into one
all_points <- bind_rows(PolesInaccess , PtOnSurface, RandomPts)
```

```r
# Check for duplicate IDs and correct if needed
if (any(duplicated(all_points$id))==T) {
  warning("Duplicate IDs detected - correcting them.")
  print(unique(all_points$id[duplicated(all_points$id)]))
  all_points$id <- as.character(seq_len(nrow(all_points))) # Replace with unique IDs
} else {
  print("No duplicate IDs.")
}
```

2.8 Create quadrats

This section generates square-shaped training polygons (quadrats) of a defined length, centered on each point in the dataset. To achieve this :

- The coordinates of each point are used to calculate the bounding box of the square.
- A custom function builds a polygon from these bounds. Each quadrat is associated with metadata such as the source, acquisition date, coordinates, and habitat classifications.
- The result is an sf object containing the spatial polygons and their attributes, ready for further processing.

```r
length_quadrat <- 4        # Length of square side in meters
radius <- length_quadrat / 2  # Half-length for buffer calculation

# Coordinates must be in meters (EPSG:32739 UTM Zone 39S)
all_points <- all_points %>%
  mutate(
    xcoord_m = st_coordinates(.)[,1],   # Extract X (easting)
    ycoord_m = st_coordinates(.)[,2],   # Extract Y (northing)
    xMinus = xcoord_m - radius,         # Lower X bound
    xPlus  = xcoord_m + radius,         # Upper X bound
    yMinus = ycoord_m - radius,         # Lower Y bound
    yPlus  = ycoord_m + radius,         # Upper Y bound
    Surface = (length_quadrat^2)        # Quadrat area
  )

# Add metadata columns
all_points$Source <- "PHOTO-INTERPRETATION"
all_points$Date <- "2022"


# Generate list of quadrats as polygons
quadrat_list <- mapply(
  create_quadrat,
  all_points$xMinus, all_points$xPlus,
  all_points$yMinus, all_points$yPlus,
  SIMPLIFY = FALSE)

# Create an sf object for the quadrats with selected attribute columns
Spdf_field_polys <- st_sf(
  all_points %>%
    select(id, Longitude, Latitude, xcoord_m, ycoord_m, Date, Source, Surface, Hab_L1, Hab_L2,
     ↪  Hab_L3, Hab_L4),
  geometry = st_sfc(quadrat_list, crs = 32739))
```

2.9 Remove overlapping quadrats

This section filters out overlapping quadrats, keeping only the first in each overlapping set.

- The function `st_overlaps()` is used to create a logical matrix (overlap_matrix) that identifies which polygons overlap with each other. Unlike simple intersection checks, this specifically detects true geometric overlap
- We then identify polygons that overlap with at least one other polygon (`overlapping_entities`) and extract these for inspection.
- A logical vector `keep` is initialized to keep track of polygons to retain. The algorithm iterates over all polygons, and for each polygon still marked to keep, it finds any overlapping polygons and marks them as FALSE in keep (i.e., removes the duplicates).
- Using the `keep`vector, the code filters out overlapping polygons and retains only one polygon per overlapping group (the first one encountered).
- The resulting filtered polygons are stored in `Allpolys`, and columns are reordered for easier interpretation and export.

```r
cat("Filtering overlapping quadrats...\n")
print("If overlap occurs, keep the first quadrat and remove the intersecting ones.")

# Logical matrix indicating which features overlap (not just intersect)
overlap_matrix <- st_overlaps(Spdf_field_polys, sparse = FALSE)

# Find indices of features that overlap with at least one other
overlapping_entities <- which(rowSums(overlap_matrix) > 0)

# Extract these overlapping polygons
overlapping_polygons <- Spdf_field_polys[overlapping_entities, ]

# Number of features
n <- nrow(Spdf_field_polys)

# Vector to mark polygons to keep (TRUE = keep)
keep <- rep(TRUE, n)

# Loop to detect overlaps and flag the later duplicates
for (i in 1:n) {
  if (!keep[i]) next  # Skip if already excluded

  # Find polygons that overlap with polygon i (excluding itself)
  overlapping <- which(overlap_matrix[i, ] & (1:n != i))

  # Mark these overlapping polygons for removal (keep = FALSE)
  keep[overlapping] <- FALSE
}

# Filter to retain only non-overlapping quadrats
Allpolys <- Spdf_field_polys[keep, ]

# Reorder columns for final export
Allpolys <- Allpolys[, c("id", "xcoord_m", "ycoord_m", "Longitude", "Latitude", "Date", "Source",
 ↪  "Surface", "Hab_L1", "Hab_L2", "Hab_L3", "Hab_L4", "geometry")]
```

2.10 Save the final shapefiles

This section prepares the final outputs `Allpolys` for export :

- It first renames the columns to match naming conventions, then saves the quadrat polygons as a shapefile.
- It also generates the centroids of these quadrats, ensures they have the correct geometry format, and exports them as a separate shapefile.

```
# Rename columns to match export conventions
names(Allpolys) <- c("N_obs", "xcoord_m", "ycoord_m", "Longitude", "Latitude", "Date", "Source",
 ↪  "Surface", "Hab_L1", "Hab_L2", "Hab_L3", "Hab_L4", "geometry")

# Write final quadrat polygons to a shapefile
st_write(Allpolys, paste0(save_plots_path, "/Quadrats_", District, "_", Island,
 ↪  "_PHOTO-INTERPRETED_Polygons_EPSG32739.shp"), driver = "ESRI Shapefile", append = FALSE)

# Generate centroids from the quadrats
centroids <- st_centroid(Allpolys)

# Ensure correct geometry structure for output
centroids <- st_as_sf(centroids, coords = c("xcoord_m", "ycoord_m"), crs = 32739)

# Write centroid shapefile
st_write(centroids, paste0(save_plots_path, "/Quadrats_", District, "_", Island,
 ↪  "_PHOTO-INTERPRETED_Centroids_EPSG32739.shp"), driver = "ESRI Shapefile", append = FALSE)
```

This script generates non-overlapping square quadrats around photo-interpreted points, based on geographic coordinates, and exports both the quadrat polygons and their centroids as shapefiles for further spatial analysis.