Step 2.5 Creating raster total

Diane ESPEL

2025-07-31

Contents

1 Objectives

This script is designed to create a comprehensive **raster stack** by combining multiple remote sensing data layers into a single multiband raster object, called `raster_total`. It integrates multispectral satellite bands (red, green, blue, near-infrared), derived spectral indices (such as NDVI, GRVI, VARI, and others), and topographic variables (digital terrain model and slope). By stacking these diverse but complementary datasets, the output raster provides a unified spatial dataset that can be used for subsequent ecological modeling, classification, or environmental analysis.

The script also handles resampling of layers to ensure consistent spatial resolution across all data sources and includes quality control steps such as cleaning slope values.

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 raster data and spatial operations
```

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 name of satellite providing multispectral imagery
- the `"Satellite2"`: the name of satellite providing terrain imagery
- the `"Year1"`: the acquisition year of the multispectral imagery
- the `"Month1"`: the acquisition month of the multispectral imagery
- the `"Year2"`: the acquisition year of terrain imagery
- the `"Res1"`: the resolution of the multispectral imagery
- the `"Res2"`: the resolution of terrain imagery

```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
Year1="2022"    # acquisition year of multispectral imagery
Month1="02" # acquisition month of multispectral imagery
Res1 = "50cm"  # spatial resolution of multispectral imagery
Satellite2="SRTM" # satellite name of DEM
Year2="2012"  # acquisition year of DEM
Res2="30m" # spatial resolution of DEM
```

2.4 Set working directory

To optimize memory, you must define two general root directories (`"localHOME"` and `"localscratch"`) that serve as the base path for your input and output data, respectively. These directories should point to the local environment where:

- input cloud mask can be located under `"data/vector/mask"`
- input rasters are located under `"data/raster/Cut_image"`
- output raster stack will be saved under `"data/raster/Cut_image"`

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

# Path to open potential cloud mask
open_mask_path <- paste0(localHOME, "/data/vector/mask")

# Path to open input raster
```

```r
open_cut_raster_path=paste0(localHOME,"data/raster/Cut_image")

# Path to save resampled topographic rasters
save_cut_topo_raster_path=paste0(localHOME,"data/raster/Cut_image")

# Path to save raster stack
save_cut_raster_path=paste0(localscratch,"data/raster/Cut_image")
```

2.5 Load and prepare rasters

The raster stacking process involves several prerequisites.

- This code first loads multispectral bands, spectral indices, and topographic rasters (DTM and slope) from file.
- It then resamples (`resample()`) the topographic rasters to match the multispectral resolution if needed
- This step also cleans and normalizes the slope raster (`slope_res`) to ensure that all values fall within a realistic and valid range.
  - The `classify()` function is first used to:
    - assign a value of 0 to all slope values less than or equal to 0,
    - assign 90 to values greater than 90. (All values from 0 to <90 are left unchanged) This reclassification helps eliminate negative or extreme slope values that may result from processing errors or noise.
  - Following this, the `clamp()` function is applied to strictly limit all slope values to the 0–90° range, replacing any out-of-bound values with NA. This ensures that the final corrected slope raster (slope_res_corr) is reliable and ready for further analysis.

If your classify() is already correct, clamp() might not be strictly necessary. But it's a good safety check to: ensure no unexpected values remain <0 or >90 and remove them (set to NA) if they exist.

```r
# Open MS features
print("Loading multispectral bands")
R <- rast(paste0(open_cut_raster_path,"/",District,"_",Island,"_",Satellite1, "_",Year1,"_",Month1,
↪  "_",Res1,"_band3_cut.TIF"))
G <- rast(paste0(open_cut_raster_path,"/",District,"_",Island,"_",Satellite1,
↪  "_",Year1,"_",Month1,"_",Res1, "_band2_cut.TIF"))
B <- rast(paste0(open_cut_raster_path,"/",District,"_",Island,"_",Satellite1,
↪  "_",Year1,"_",Month1,"_",Res1, "_band1_cut.TIF"))
NIR <- rast(paste0(open_cut_raster_path,"/",District,"_",Island,"_",Satellite1,
↪  "_",Year1,"_",Month1,"_",Res1, "_band4_cut.TIF"))

# Open Spectral indices
print("Loading spectral indices")
NDVI <- rast(paste0(open_cut_raster_path,"/",District,"_",Island,"_",Satellite1,
↪  "_",Year1,"_",Month1, "_",Res1,"_ndvi_cut.TIF"))
GRVI <- rast(paste0(open_cut_raster_path,"/",District,"_",Island,"_",Satellite1,
↪  "_",Year1,"_",Month1,"_",Res1, "_grvi_cut.TIF"))
VARI <- rast(paste0(open_cut_raster_path,"/",District,"_",Island,"_",Satellite1,
↪  "_",Year1,"_",Month1,"_",Res1, "_vari_cut.TIF"))
GCCI <- rast(paste0(open_cut_raster_path,"/",District,"_",Island,"_",Satellite1,
↪  "_",Year1,"_",Month1,"_",Res1, "_gcci_cut.TIF"))
Brightness <- rast(paste0(open_cut_raster_path,"/",District,"_",Island,"_",Satellite1,
↪  "_",Year1,"_",Month1,"_",Res1, "_brightness_cut.TIF"))
```

```r
BSI <- rast(paste0(open_cut_raster_path,"/",District,"_",Island,"_",Satellite1,
 ↪  "_",Year1,"_",Month1,"_",Res1, "_bsi_cut.TIF"))
NDWI <- rast(paste0(open_cut_raster_path,"/",District,"_",Island,"_",Satellite1,
 ↪  "_",Year1,"_",Month1,"_",Res1, "_ndwi_cut.TIF"))

# Open Topographic rasters : DTM and slope
print("Loading dtm and slope")
Dtm <-
 ↪  rast(paste0(open_cut_raster_path,"/",District,"_",Island,"_",Satellite2,"_",Year2,"_",Res2,"_dtm_cut.TIF"))
 ↪  # Digital Surface model
Slope <-
 ↪  rast(paste0(open_cut_raster_path,"/",District,"_",Island,"_",Satellite2,"_",Year2,"_",Res2,"_slope_cut.TIF"))
 ↪  # slope


# Resample topographic rasters to match multispectral resolution
print("Resampling DTM and slope to match the finer multispectral resolution")
if (res(Dtm)[1]>res(R)[1]){
  dtm_res=terra::resample(Dtm,R,method="bilinear")  #Resample topographic features to match R
 ↪  raster resolution
  slope_res=terra::resample(Slope,R,method="bilinear") # Resample topographic features to match R
 ↪  raster  resolution
}else{
  dtm_res=Dtm
  slope_res=Slope
}


rm(Dtm,Slope)# Remove original unresampled rasters

# Correct slope values
print("Correcting invalid slope values")
slope_res_corr <- classify(slope_res, matrix(c(-Inf, 0, 0, 90, Inf, 90), ncol=3, byrow=TRUE))#
 ↪  Reclassify values of slope_res
slope_res_corr <- clamp(slope_res_corr, lower=0, upper=90,values=FALSE) # limit values to the range
 ↪  [0, 90], removing any outside values
rm(slope_res) #remove useless layers

# save new layers
print("Saving cleaned and resampled DTM and slope rasters")
writeRaster(dtm_res,paste0(save_cut_topo_raster_path,"/",District,"_",Island,"_",Satellite2,"_",Year2,"_",Res1,"_dt
writeRaster(slope_res_corr,paste0(save_cut_topo_raster_path,"/",District,"_",Island,"_",Satellite2,"_",Year2,"_",Re
```

2.6 Stack rasters

All layers are combined(/stacked) into a single multiband-raster stack named `raster_total`.

This part also checks for the presence of a shapefile containing cloud and building mask. If the mask (**cloud__mask__file**) exists, the script :

- Rasterizes the vector mask so that pixels inside the polygons are assigned a value of **1** and those outside get **0**.
- Creates an inverted mask where pixels outside the polygons are **TRUE** and inside polygons are **FALSE**
- Applies this inverted mask to **raster__total** to remove (mask out) pixels inside the cloud/building polygons, effectively filtering out those unwanted areas.

If the mask shapefile does not exist, it copies **raster__total** into **raster__total2** without masking.

Finally, the cleaned raster stack (**raster__total2**) is saved as a GeoTIFF file.

```
#  Stack MS bands into raster_MSbands
print("Stacking multispectral bands")
raster_MS=c(R,G,B,NIR) # stack bands
names(raster_MS) = c("R", "G", "B", "NIR") # rename layers in the raster stack
rm(R,G,B,NIR) # remove useless bands
writeRaster(raster_MS,paste0(save_cut_raster_path,"/",District,"_",Island,"_",Satellite1,"_",Year1,"_",Month1,"_",R
print(raster_MS)

#  Stack all indices into raster_indices
print("Stacking spectral indices")
raster_indices=c(NDVI,GRVI,VARI,GCCI,Brightness,BSI,NDWI) # stack indices
names(raster_indices) = c("NDVI","GRVI","VARI","GCCI","Brightness","BSI","NDWI") # rename layers in
↪  the raster stack
rm(NDVI,GRVI,VARI,GCCI,Brightness,BSI,NDWI)  # remove useless layers
writeRaster(raster_indices,paste0(save_cut_raster_path,"/",District,"_",Island,"_",Satellite1,"_",Year1,"_",Month1,
print(raster_indices)

#  Stack all rasters into raster_total
print("Creating raster stack: all MS bands + indices + topographic layers")
raster_total<- c(raster_MS,raster_indices,dtm_res,slope_res_corr) #stack all layers
names(raster_total)= c("R", "G", "B",
↪  "NIR","NDVI","GRVI","VARI","GCCI","Brightness","BSI","NDWI","Dtm","Slope") # rename layers in
↪  the raster stack
rm(raster_MS,raster_indices,dtm_res,slope_res_corr) # remove useless layers

#  Mask clouds and buildings if present
cloud_mask_file <- paste0(open_mask_path,
↪  "/",District,"_",Island,"_Clouds_Buildings_",Year1,"_EPSG32739.shp")

if (file.exists(cloud_mask_file)) {

  cat("Mask shapefile found, masking out pixels inside the polygon...\n")

  # Load the mask shapefile as a vector object
  cloud_mask <- vect(cloud_mask_file)

 # Check CRS of raster and vector, reproject vector if different
  raster_crs <- crs(raster_total)
  vector_crs <- crs(cloud_mask)

  if (!identical(raster_crs, vector_crs)) {
    cloud_mask <- project(cloud_mask, raster_crs)
  }

  # Rasterize mask polygon: pixels inside polygon = 1, outside = 0
  mask_raster <- rasterize(cloud_mask, raster_total, field=1, background=0)

  # Invert mask: TRUE outside polygon (0), FALSE inside polygon (1)
  inverted_mask <- mask_raster == 0

  # Mask raster_total to remove pixels inside polygon
  raster_total2 <- mask(raster_total, inverted_mask, maskvalues=FALSE)

} else {
  cat("Mask shapefile NOT found, skipping masking.\n")
  raster_total2=raster_total
}
```

```r
# Save raster stack
writeRaster(raster_total2,paste0(save_cut_raster_path,"/",District,"_",Island,"_",Satellite1,"_Total_raster_stack_"
print(raster_total2)
```

This stacked raster consolidates all the available spectral and environmental information, facilitating integrated spatial analyses or machine learning workflows.