Segmenting soil maps

D G Rossiter

D G Rossiter (ORCID 0000-0003-4558-1286)

2025-01-06

Table of contents

1. Abstract

Digital soil maps (DSM) of soil properties or classes are made at set grid resolutions, whereas soil patterns on the landscape have characteristic scales depending on the soil geomorphology. By segmenting a gridded map at various resolutions we attempt to identify soil patterns at various scales. The main aim is to evaluate DSM products in terms of how well they represent the soil landscape. The GeoPAT suite of programs can segment into (usually grouped) grid cells, minimum of 10x the original grid cell resolution of the source maps.

This script sets up the methods; separate scripts import the functions from here (exported with `purl`) and use them in case studies.

2. Introduction

GeoPAT is an implementation of the concepts of Jasiewicz *et al.* (2018).

GeoPAT itself is described in Jasiewicz *et al.* (2015) and its full documentation is by Netzel *et al.* (2018). Chapter 3 has basic workflows, Chapter 2 the detailed command descriptions.

Jasiewicz, J., Stepinski, T., & Niesterowicz, J. (2018). Multi-scale segmentation algorithm for pattern-based partitioning of large categorical rasters. COMPUTERS & GEOSCIENCES, 118, 122-130. https://doi.org/10.1016/j.cageo.2018.06.003

Jasiewicz, J., Netzel, P., & Stepinski, T. (2015). GeoPAT: A toolbox for pattern-based information retrieval from large geospatial databases. Computers & Geosciences, 80, 62-73.https://doi.org/10.1016/j.cageo.2015.04.002

Netzel, P., Nowosad, J., Jasiewicz, J., Niesterowicz, J., & Stepinski, T. (2018). GeoPAT 2: User's manual. https://zenodo.org/records/1291123

Figure 1 shows the segmentation workflow.

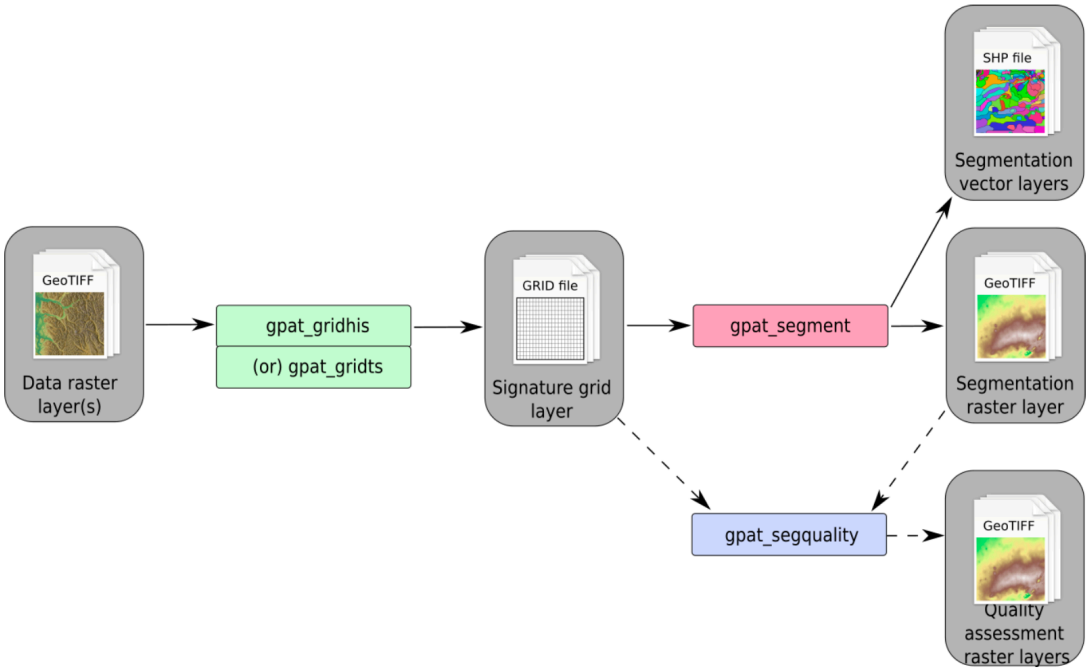

Figure 3.13: Workflow path for segmentation

Figure 1: Segmentation workflow (GeoPAT Manual)

The segmentation can be followed by *regionalization*, i.e., grouping related patterns, in this case soil "combinations", into map units. Regionalization is by clustering segments output by `gpat_segment` into distinct pattern types. Outputs of the `gpat_distmtx` module can be used as a distance matrix in a clustering program, of which there are many in R.

This document (so far) only produces and evaluates the segmentation.

3. Setup

3.1 Packages

The only non-base R packages are:

- `terra` to handle raster and vector structures
- `sf` "simple features" for some plots
- `ggplot` for GGPlot graphics
- `RColorBrewer` for palettes
- `aqp` for Shannon entropy

```
require(terra)
```

```
Loading required package: terra
```

```
terra 1.8.7
```

```
require(sf)
```

```
Loading required package: sf
```

```
Linking to GEOS 3.13.0, GDAL 3.10.0, PROJ 9.5.1; sf_use_s2() is TRUE
```

```
require(ggplot2)
```

```
Loading required package: ggplot2
```

```
require(RColorBrewer)
```

```
Loading required package: RColorBrewer
```

```
require(gstat) # variograms
```

```
Loading required package: gstat
```

```
require(aqp)   # entropy
```

```
Loading required package: aqp
```

```
This is aqp 2.1.0
```

3.2 File paths

GeoPAT is a stand-alone program made of several modules, they must be called via R's `system` command. Source code is at https://github.com/Nowosad/geopat2. These commands must be compiled for each operating system, and the path will be different.

```
# geopat_path <- "/Users/rossiter/dl/geoPAT_program/geopat2_MacOS"
geopat_path <- "/usr/local/bin"
```

```
gridhis_path <- file.path(geopat_path, "gpat_gridhis")
segment_path <- file.path(geopat_path, "gpat_segment")
segqual_path <- file.path(geopat_path, "gpat_segquality")
grid2txt_path <- file.path(geopat_path, "gpat_grid2txt")
distmtx_path <- file.path(geopat_path, "gpat_distmtx")
```

A base directory where we keep our GeoPAT input files (grids) and results (segmentation); this path will be expanded for each case study.

```
base_path <- "/Users/rossiter/tmp/GeoPAT"
```

3.3 Key parameters

Several choices influence the grid sizes that will be used to make segments. These should correspond to the map scale: soil associations, soil series, soil phases… related to the landscapes structure.

The defaults for grid creation in `geopat_gridhis` are

- `-z 150` *motifel size*, and
- `-f 100` *shift*, both in input raster cell units.

A *motifel* ("motif element"?) is the elementary unit of analysis: a square block of pixels representing a local pattern. It is quantified by a histogram of features (e.g., co-occurrence, decomposition – see below). Then distance between motifels is calculated as a distance between their histograms by various metrics, e.g., Jensen-Shannon Divergence).

The *motifel size* defines the size of individual scene for which the histogram is calculated.

The *shift* is the number of grid cells by which scenes shift along the grid in vertical and horizontal directions. The resolution of the output grid equals to the resolution of the input raster map multiplied by the shift. For example shift is 100 cells and the input raster has 25 m resolution, the output grid would have $25\ \text{m} \times 100 = 2500\ \text{m}$ resolution.

Thus the motifel size can be the shift size with some buffer; it could be the same, and it could even be smaller. If shift is set to the same value as motifel size, the input map will be simply divided into a grid of non-overlapping motifels. The idea of a larger motifel than size is to account for border effects.

A key part of the analysis is adjusting these until patterns emerge, at various scales.

3.4 Helper functions

A function to compile Minimum Legible Area (in ha) from the scale. Input is the denominator of the scale ratio, e.g., for 1:50 000 it is 50 000.

```
scale.to.mla <- function(scale.number) {
  return((((scale.number/1000)^2)/400))
}
```

The square root of this, when converted to m^2), is then the grid side in m. Example:

```
(mla <- scale.to.mla(50000))
```

```
[1] 6.25
```

```
sqrt(mla*(100^2))
```

```
[1] 250
```

So to correspond to a 1:50 000 scale map, we look for patterns in a 2.5 x 2.5 m^2 grid cell, which corresponds to a MLA of 6.25 m^2.

This function computes the shift, in map units, from the scale number. It is not used in the computation but can be called separately to figure out the desired shift.

The relative proportion of the motifel vs. shift can be adjusted elsewhere in the workflow.

Note that the segmentation combines the rectangular tiling into a "brick" topology, see GeoPAT manual Appendix A. These cells are 2 x 2 of the square tiles. So to get the pattern at a given grid size, we need to specify the original grid size at 2x resolution.

```
grid.dimensions <- function(scale.number) {
  shift <- sqrt(scale.to.mla(scale.number)*(100^2))/2
  # /2 to account for aggregation in the segmentation phase
  return(shift)
}
```

Nominal scale from shift in meters:

```
effective.scale <- function(shift.m) {
  return(shift.m*400)
}
```

The MLA used by the program will be 1/4 the size of the MLA for the given scale, this to allow the "brick" topology.

4. Workflow

Here are functions for the workflow, adjustable with parameters.

The input map(s) must have been prepared as classified GeoTIFF.

4.1 Grid signature

Build a pattern signature for maps using `gridhis`. A separate signature must be created for each layer, with that layer's statistics in the grid.

"This module extracts a grid-of-scenes (grid of pattern signatures, grid of motifels). The output is a grid of the same spatial extent as the input raster map, but with a different cell size. Each cell in a new grid has only one attribute - the signature of its pattern. Pattern is calculated over a window centered on the cell and having a user-defined size."

Arguments:

0.`case_path`, where the files to process are found and results stored, as subdirectories `ds` and `results`

1. layer name of the property; file names are built from this – `source_TIFF` was the output of the data preparation for this layer It must be a classified raster – `input_GRID` will be the input to the segmentation
2. `shift.side` = shift size, in map units (e.g., meters)
3. `motifel.mult` = motifel size, as a proportion of shift.side, default 1.5 – this matches the ratio recommended in the manual (shift 100, motifel 150). From the manual: "Shift defines the resolution of the new grid. If window size is bigger than shift (recommended), motifels will overlap". So we set the default motifel to be 50% larger than the shift. It is the motifel that determines the size of the pattern.
4. `s` = signature type, default `cooc` "Spatial co-ocurrence of categories". See GeoPAT Manual Appendix B for options.
5. `n` = normalization type, default `pdf` (probability distribution function) which is recommended for `cooc`. See GeoPAT Manual Appendix C for options.

Outputs:

1. the signature grid as a GeoTIFF, input to the segmentation step
2. the signature grid as a text file (this is not yet used for more processing)

Returns:

1. vector with shift and motifel size, in grid units. This is used to keep outputs from different runs separate

The function computes the shift and motifel size from the resolution and a multiplier for the motifel side. But these can not be < 10 grid cells.

Some details on the methods:

From the `-l` option to `gpat_gridhis`:

List of signatures:

`prod` Cartesian product of input category lists `cooc` Spatial coocurrence of categories `fdec` Full decomposition `lind` Landscape indices vector `linds` Selected landscape indices vector `ent` Shannon entropy

List of local normalization methods:

`01` Normalize to the interval [0, 1] `pdf` Normalize to pdf (sum(hi) = 1) `N01` Normalize to N(0, 1) (hi=(hi-avg)/std) `none` Signature without any normalization

Spatial co-ocurrence of categories: `s = cooc` uses a color co-occurrence histogram, a variant of the Gray-Level Co- occurrence Matrix (GLCM) used to characterize texture in grayscale images. In GeoPAT, color is replaced by cell class and a single cell separation of one pixel is used to calculate a co-occurrence histogram.

"The co-occurrence signature is designed to be effective in encapsulating spatial structures exhibiting high complexity patterns."

Reference: Haralick, R. M., Shanmugam, K., & Dinstein, I. (1973). Textural features for image classification. IEEE Transactions on Systems, Man, and Cybernetics, SMC-3(6), 610-621. https://doi.org/10.1109/TSMC.1973.4309314. A popular explanation is Hall-Beyer, M. (2017). GLCM Texture: A Tutorial v. 3.0 March 2017. http://hdl.handle.net/1880/51900

Alternatives are (1) to use a selected landscape index (FRAGSTATS-style); (2) a hierarchical decomposition.

Probability Distribution Function `n = pdf` normalization, i.e., fit the empirical distribution with the sample mean (to 0) and standard deviation (to 1), so all layers have the same weight.

Alternative is (1) stretch minimum and maximum to [0…1].

```
# system(gridhis_path)
make.signature <- function(case_path, layer.name,
                           shift.side, motifel.mult = 1.5,
                           s = "'cooc'", n = "'pdf'") {

  # paths for this case
  ds_path <- file.path(case_path, "ds")
  results_path <- file.path(case_path, "results")

  # source files
  ## classified -- for segmentation
  source_TIFF <- file.path(ds_path, paste0(layer.name, ".tif"))
  # minimun shift is 10
  r.c <- rast(file.path(ds_path, paste0(layer.name, ".tif")))
  res <- res(r.c)[1]
  f <- max(shift.side, 10)  # must have at least 10 cells per side
  z <- round(shift.side*motifel.mult)
  #
  # what resolution could be achieved
  print(paste("Shift size (grid units):", f, "; (map units):", f*res))
  print(paste0("Effective scale 1:", sprintf("%d", effective.scale(f*res))))
  print(paste("Motifel size (grid units):", z, "; (map units):", z*res))
  # print(paste("Grid cell area:", sprintf("%d", (f*res)^2)))
  # print(paste("Motifel size (grid units):", z, "; (map units):", z*res))
  #
  input_GRID <- file.path(results_path, paste0(layer.name, "_", f, "_", z,
".tif"))
  input_TXT <- file.path(results_path, paste0(layer.name, "_", f, "_", z,
```

```
".txt"))
#  if (!file.exists(input_GRID)) {
    system(paste0(gridhis_path,
                  " -i ", source_TIFF,
                  " -o ", input_GRID,
                  " -z ", z,  # motifel size
                  " -f ", f,  # shift size
                  " -s ", s,
                  " -n ", n))
#    system(paste0(grid2txt_path,
#                  " -i ", input_GRID,
#                  " -o ", input_TXT))
#  }
  return(c(f, z))
}
```

4.2 Segmentation

The grids with signatures are inputs to the segmentation algorithm.

"This module segments a grid-of-scenes into regions of uniform patterns in the same fashion as traditional segmentation algorithms segment image (or other rasters) into segments of uniform color and texture. The difference is that `gpat_segment` segments a *grid-of-scenes* rather than an ordinary grid, and uses *scene signature as attribute* rather than color or image texture to decide how to delineate the segments."

See GeoPAT manual $\S2.2.2.3$ for details, and Appendix D for options.

Accept the default "average" linkage, can specify "complete" linkage (`-c` argument).

Propose the default minimum distance thresholds to build areas from grids: `-lthreshold=0.1` (to control the sizes of segments) and `-uthreshold=0.3` (to prevent the growth of inhomogeneous segments). In multilayer segmentation "a motifel must meet the threshold conditions in all input layers to be joined to a segment. This is a rigorous solution and assure the full compatibility of all layers in the output segmentation."

These thresholds `{l,u}threshold` are the minimum and maximum distance thresholds to build areas. To get larger segments, reduce the minimum and increase the maximum.

So this is quite strict: each layer's motifel in a grid cell must be similar enough to a neighbour's in order for them to merge. The `-uthreshold` can be relaxed for multi-layers, if the defaults give too fine a segmentation.

An alternative is to weight the layers (`-w=` argument, with a list of weights parallel to the list of layers). "The weighted geometric mean of all layers in the motifel must be below the threshold to be joined to a segment. Weights allow to control the role of individual layers. Layer with a large weight value will have more impact on the resulting segmentation."

Another parameter is `-swap=0.001` by default. "There is a chance that some motifels located at segments' boundaries show more affinity to motifels across the boundary than to

motifels in its own segment. The swap parameter is used to alleviate this problem. It adjusts the boundaries between the segments."

"For each border motifel its linkage to the parent segment, D_{par}, as well as its linkage to the segment across the boundary, D_{neigh} is calculated. If the difference between D_{par} and D_{neigh} is positive (the linkage to the segment across the boundary is smaller than the linkage to parent segment) the motifel is marked for possible reassignment (resulting in an adjustment to the boundary). The boundary between segments is adjusted only when the swap value is larger than the difference between D_{par} and D_{neigh}."

Function to do the segmentation, using GeoPAT command `gpat_segment`. Arguments:

0.`case_path`, where the files to process are found and results stored, in subdirectory `results`

1. Vector of layers (as names) to include in the segmentation calculation
2. `f.z` 2-element vector with shift and motifel, to label outputs
3. Similarity measure, default `jsd` (Jensen-Shannon Distance)

4, 5. Segmentation thresholds `{l,u}threshold`, defaults 0.1, 0.3

6. Vector of per-layer weights, default `NA`.

The segmentation output as grid and polygons are named by the layer list.

Input file is a GRID (signature); outputs are a raster TIFF with segments and a vector file GPKG also with segments.

The function returns the constructed names of the segments files.

```
# system(segment_path)
segment.it <- function(case_path, layers, f.z,
                       m = "jsd",
                       lthreshold = 0.1, uthreshold = 0.3,
                       w = NA) {

  results_path <- file.path(case_path, "results")
  #
  segmented.name <- paste(layers, collapse="-")
  if (nchar(segmented.name) > 64) segmented.name <- paste0("Layers",
length(layers))
  segment_TIFF <- file.path(results_path,
                            paste0(segmented.name, "_", f.z[1], "_", f.z[2],
"_seg.tif")) # segmented grid
  segment_GPKG <- file.path(results_path,
                            paste0(segmented.name, "_", f.z[1], "_", f.z[2],
"_seg.gpkg")) # polygons
  input.names <- paste0(" -i ", file.path(results_path,
                                          paste0(layers, "_", f.z[1], "_",
```

```
f.z[2], ".tif"))) #  input grid
  # make one (long) string of the inputs
  input.names <- paste(input.names, collapse = "   ")
  #
  system(paste0(segment_path, input.names,
                " -o ", segment_TIFF,
                " -v ", segment_GPKG,
                " -m ", m,
                if(!is.na(w[1])) { paste0(" -w=",w) },
#                " -c",           # complete linkage
               paste0(" --lthreshold=", lthreshold),
               paste0(" --uthreshold=", uthreshold),
               " --swap=0.001") # low value to encourage swapping
#                 " --maxhist=0"  # turn off leveraging, increased computation
  )
  # report the number of segments
  # tmp <- vect(segment_GPKG)
  # print(paste("Number of segments:", nrow(tmp)))
  return(c(segment_TIFF, segment_GPKG))
}
```

4.3 Evaluation

The quality of the segmentation can be computed for each input layer, i.e., how well was that layer segmented, whether it was segmented as one layer or as part of a multi-layer segmentation.

Arguments:

0. `case_path`, where the files to evaluate are found, in subdirectory `results`
1. layers for which to evaluate the segmentation
2. `segment_m_TIFF` The segmentation results `.tif` from the previous step
3. `f.z` 2-element vector with shift and motifel, to label outputs
4. Similarity measure, default `jsd` (Jensen-Shannon Divergence)

It produces two output maps, one for the internal inhomogeneity of a grid, and one for its isolation from its neighbours. Their names are returned

```
# system(segqual_path)
evaluate.it <- function(case_path, layers, segment_m_TIFF, f.z, m = "jsd") {

  results_path <- file.path(case_path, "results")
  #
  segmented.name <- paste(paste0(layers, "_", f.z[1], "_", f.z[2]),
collapse="-")
  if (nchar(segmented.name) > 64) segmented.name <- paste0("Layers",
length(layers), "_", f.z[1], "_", f.z[2])
```

```
  inhomogeneity_m_TIFF <- file.path(results_path, paste0(segmented.name,
"_inhomo.tif"))
  isolation_m_TIFF <- file.path(results_path, paste0(segmented.name,
"_iso.tif"))
  input.names <- paste0(" -i ", file.path(results_path, paste0(layers, "_",
f.z[1], "_", f.z[2] , ".tif")))
  #
  system(paste0(segqual_path, input.names,
                " -s ", segment_m_TIFF,
                " -g ", inhomogeneity_m_TIFF,
                " -o ", isolation_m_TIFF,
                " -m", m))
  #
  return(c(inhomogeneity_m_TIFF, isolation_m_TIFF))
}
```

4.4 Computation

A function to compute and display all the steps.

Arguments:

1. `shift.side` number of cells on a side of the grid
2. `case_path` base directory where the source DSM raster layers are stored in subdirectory `ds` and where the results will be stored in subdirectory `results`.
3. `layers` list of layers in that directory
4. `motifel.mult` size of motifel grid side, as a proportion of the shift size; default 1.5
5. `s` signature type, default `cooc`, see `make.signature()`
6. `n` normalization type, default `pdf`, see `make.signature()`
7. `m` similarity metric, default `jsd` = Jensen-Shannon distane
8. `thresholds` two-element vector with the lower and upper segmentation thresholds, default `c(0.1, 0.3)`
9. Per-layer weights, default `NA`.
10. `plot.seg.numbers` whether or not to plot segment #s on maps, default `FALSE`; this can be useful for understanding why segments are built as they are.
11. `palette` to use when plotting maps with segment boundaries overlaid. Default `""` will give the terra default terra::map.pal("viridis")
12. `variogram.analysis` to compute fitted variogram parameters for each segment, summarize them, and plot. Default `FALSE` because this is *very* time-consuming. Probably should only be turned on when there are 50 or fewer segments.

Returns:

1. Number of segments

Side effects: summarizes and maps evaluation statistics

```
compute.it <- function(shift.side, case_path, layers,
                       motifel.mult = 1.5,
                       s = 'cooc',  # same default as make.signature
                       n = 'pdf',  # same default as make.signature
                       m = "jsd",
                       thresholds = c(0.1, 0.3),
                       w = NA,
                       plot.seg.numbers = FALSE,
                       palette = "",
                       variogram.analysis = FALSE) {

  print(paste("Shift side:", shift.side))

  # signatures
  for (l in layers) {
    # will convert these sizes in meters to size in grid cells
    f.z <- make.signature(case_path, l, shift.side, motifel.mult, s, n)
  }

  # segmentation
  segment.results <- segment.it(case_path,
                                layers, f.z, m, w,
                                lthreshold = thresholds[1],
                                uthreshold = thresholds[2])

  # plot segmentation results over all input layers
  ds_path <- file.path(case_path, "ds")
  v.seg <- vect(segment.results[2])
  print(paste("Number of segments:", dim(v.seg)[1]))
  for (l in layers) {
    r.c <- rast(file.path(ds_path, paste0(l, ".tif")))
    lvls <- (length(unique(values(r.c[[1]]))))
    if (palette == "") { col.ramp <- map.pal("viridis", n=lvls)
    } else { col.ramp <- colorRampPalette(brewer.pal(
      n = 9, name = palette))(lvls)
    }
    terra::plot(r.c, main = paste0(l, "; Shift: ", shift.side*res(r.c)[1], "
m"), col = col.ramp)
    terra::plot(v.seg, add = TRUE)
    if (plot.seg.numbers) terra::text(v.seg, labels = "segment_id",
                                      inside = TRUE, cex = 0.8)

    ### compute and display segmentation metrics
```

```
    ## 1- from continuous map, from which classes were made
    source_TIFF_p <- file.path(ds_path, paste0(l, "_p.tif"))
    r.p <- rast(source_TIFF_p)
    # use a generic name for the property, for `variogram()` etc.
    names(r.p) <- "z"

    # 1a - segment standard deviations
    seg.sd <- unlist(extract(r.p, v.seg, fun = sd, na.rm = TRUE)[2])
    print("Segment standard deviations:")
    print(quantile(seg.sd, na.rm = TRUE))
    #
    if (plot.seg.numbers)  {
      terra::plot(r.p, main = "Segment standard deviation",
                  legend = FALSE, col = col.ramp)
      terra::plot(v.seg, add = TRUE)
      terra::text(v.seg, labels = round(seg.sd, 2),
                  inside = TRUE, cex = 0.8)
    }

    if (variogram.analysis) {
      # 1b - segment variogram parameters
      # Convert segment vector file to sf, this splits into separate polygons
      (v.sf <- st_as_sf(v.seg))
      # structure to save variogram parameters
      vgm.params <- data.frame(id = 0, psill=0, range=0, nugget=0)
      system.time(
        # each segment, in the same order as in the SpatVector
        for (s in v.sf$segment_id) {
          # extract one polygon
          v <- v.sf[s,]
          # convert it back to SpatVect
          v.v <- vect(v)
          # get the values in that segment
          r.e <- crop(r.p, v.v) # reduce bounding box
          r.e <- mask(r.e, v.v)    # just the cells in the segment
          # plot(r.e)
          # convert back to sf (via sp) to compute the variogram
          r.e.spdf <- as.data.frame(r.e, xy = TRUE)
          sp::coordinates(r.e.spdf) <- ~x + y
          sp::proj4string(r.e.spdf) <- crs(r.e)
          r.e.sf <- st_as_sf(r.e.spdf)
          # variogram, using the general target variable name
          vg <- variogram(z ~ 1, r.e.sf)
          # plot(vg)
          #
          # initial estimates for the model -- use default NA:
          # "If any of the initial parameters of model are NA,
          #   they are given default values as follows.
          #   The range parameter is given one third of the maximum value of
object$dist.
```

```
          #   The nugget value is given the mean value of the first three
values of object$gamma.
          #  The partial sill is given the mean of the last five values of
object$gamma"
          vgm <- vgm("Exp")
          # fit it
          vgmf <- fit.variogram(vg, vgm)
          # save the parameters
          # if range is too long, the fit did not converge
          p1 <- st_point(st_bbox(r.e.sf)[1:2])
          p2 <- st_point(st_bbox(r.e.sf)[3:4])
          (diag.r.e <- st_distance(p1, p2))
          if (vgmf$range[2] >  diag.r.e) {
            df <- data.frame(id = s, psill = NA, range = NA, nugget = NA)
          } else {
            df <- data.frame(id = s, psill = vgmf$psill[2], range =
vgmf$range[2], nugget = vgmf$psill[1])
          }
          vgm.params <- rbind(vgm.params, df)
        }
      )
      # remove dummy first row from the parameter dataframe
      vgm.params <- vgm.params[-1, ]
      # these should be parallel
      # sum(vgm.params$id - v.seg$segment_id)
      print("Partial sills:")
      print(summary(vgm.params$psill))
      print("Nuggets:")
      print(summary(vgm.params$nugget))
      print("Proportional nuggets:")
      print(summary(vgm.params$nugget/(vgm.params$psill +
vgm.params$nugget)))
      print("Ranges:")
      # adjust to km, assuming the original is in metres
      vgm.params$range <- vgm.params$range/1000
      print(summary(vgm.params$range))
      # total sill is similar to variance, we have that directly

      # show variogram parameters on map
      terra::plot(r.p, main = "Variogram range (km)", legend = FALSE, col =
col.ramp)
      terra::plot(v.seg, add = TRUE)
      # terra::text(v.seg, labels = "segment_id",
      #             inside = TRUE, cex = 0.8, col="blue", halo = TRUE)
      terra::text(v.seg, labels = round(vgm.params$range,1),
                  inside = TRUE, cex = 0.8)
      #
      terra::plot(r.p, main = "Variogram total sill", legend = FALSE, col =
col.ramp)
      terra::plot(v.seg, add = TRUE)
```

```
      # terra::text(v.seg, labels = "segment_id",
      #             inside = TRUE, cex = 0.8, col="blue", halo = TRUE)
      terra::text(v.seg, labels = round(vgm.params$psill +
vgm.params$nugget,1),
                  inside = TRUE, cex = 0.8)
      #
      #
      terra::plot(r.p, main = "Variogram proportional nugget [0..1]", legend
= FALSE, col = col.ramp)
      terra::plot(v.seg, add = TRUE)
      # terra::text(v.seg, labels = "segment_id",
      #             inside = TRUE, cex = 0.8, col="blue", halo = TRUE)
      terra::text(v.seg, labels = round(vgm.params$nugget/(vgm.params$psill +
vgm.params$nugget),2),
                  inside = TRUE, cex = 0.8)
    }

    ## 2 - from classified map
    # extract the values of each segment, as a table
    seg.vals <- extract(r.c, v.seg, fun = table)
    # as proportion
    seg.vals.p <- apply(seg.vals[, -1], MAR = 1, FUN = function(x) {x/sum(x)}
)
    #2a - Normalized Shannon entropy
    seg.ent <- apply(seg.vals.p, MARGIN = 2,
                     function(x) { aqp::shannonEntropy(x, b =
dim(seg.vals.p)[1]) } )
    print("Segment normalized Shannon entropy:")
    print(quantile(seg.ent, na.rm = TRUE))
    #
    if (plot.seg.numbers)  {
      terra::plot(r.c, main = "Shannon entropy, [0..1]",
                  legend = FALSE, col = col.ramp)
      terra::plot(v.seg, add = TRUE)
      terra::text(v.seg, labels = round(seg.ent, 2),
                  inside = TRUE, cex = 0.8)
    }

  }

  # evaluation of segmentation quality
  evaluation.results <- evaluate.it(case_path,
                                    layers,
                                    segment.results[1],
                                    f.z,
                                    m)
  # Quality: (1) Inhomogeneity of the segment:
  r_m.inhomo <- rast(evaluation.results[1])
  # Quality: (2) Isolation from neighbours:
  r_m.iso <- rast(evaluation.results[2])
```

```
  # Overall segmentation quality:
  r_m.qual = (1 - r_m.inhomo/r_m.iso)

  # plot the three evaluations on one figure
  par(mfrow = c(2, 2))
  plot(r_m.inhomo,
       col = rev(heat.colors(64)),
       main = "Inhomogeneity")
  terra::plot(v.seg, add = TRUE)
  if (plot.seg.numbers) terra::text(v.seg, labels = "segment_id",
                                    inside = TRUE, cex = 0.8)
  #
  plot(r_m.iso,
       col = heat.colors(64),
       main = "Isolation")
  terra::plot(v.seg, add = TRUE)
  if (plot.seg.numbers) terra::text(v.seg, labels = "segment_id",
                                    inside = TRUE, cex = 0.8)#
  #
  plot(r_m.qual,
       col = topo.colors(64),
       main = "Overall discrimination")
  terra::plot(v.seg, add = TRUE)
  if (plot.seg.numbers) terra::text(v.seg, labels = "segment_id",
                                    inside = TRUE, cex = 0.8)
  par(mfrow = c(1, 1))

  # show summary of quality metrics
  print("Inhomogeneity values:")
  print(quantile(values(r_m.inhomo), seq(.1, .9, by = 0.1)))
  print("Isolation values:")
  print(quantile(values(r_m.iso), seq(.1, .9, by = 0.1)))
  print("Discrimination values:")
  print(quantile(values(r_m.qual), seq(.1, .9, by = 0.1)))

  # return the number of segments
  return(dim(v.seg)[1])
}
```

5. Export

Export the workspace, to be imported into case studies.

```
save.image(file = "SegmentingSoilMaps_Functions.RData")
```