Package 'scapesClassification'

October 14, 2022

Title User-Defined Classification of Raster Surfaces

Version 1.0.0

Depends R (>= 3.5.0)

Description Series of algorithms to translate users' mental models of seascapes, landscapes and, more generally, of geographic features into computer representations (classifications). Spaces and geographic objects are classified with user-defined rules taking into account spatial data as well as spatial relationships among different classes and objects.

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.1.2

Suggests gifski, knitr, leafem, leaflet, leafpop, mapview, raster, spelling, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

Imports terra, methods

VignetteBuilder knitr

URL https://github.com/ghTaranto/scapesClassification, https://ghtaranto.github.io/scapesClassification/

BugReports https://github.com/ghTaranto/scapesClassification/issues

Language en-US

NeedsCompilation no

Author Gerald H. Taranto [aut, cre] (<https://orcid.org/0000-0002-7968-1982>)

Maintainer Gerald H. Taranto <gh.taranto@gmail.com>

Repository CRAN

Date/Publication 2022-03-16 13:30:02 UTC

R topics documented:

anchor.cell . . . 2
anchor.seed . . . 5
anchor.svo . . . 9
attTbl . . . 12
classify.all . . . 14
cond.4.all . . . 16
cond.4.nofn . . . 18
cond.parse . . . 24
cond.reclass . . . 25
conditions . . . 27
cv.2.rast . . . 32
ngb8 . . . 33
ngbList . . . 34
obj.border . . . 37
obj.nbs . . . 39
peak.cell . . . 41
pi.add . . . 43
pi.sgm . . . 46
reclass.nbs . . . 49
rel.pi . . . 52

Index **55**

`anchor.cell` *Cell numbers to class vector*

Description

Converts a vector of cell numbers into a class vector.

Usage

```
anchor.cell(
  attTbl,
  r,
  anchor,
  class,
  classVector = NULL,
  class2cell = TRUE,
  class2nbs = TRUE,
  overwrite_class = FALSE,
  plot = FALSE,
  writeRaster = NULL,
  overWrite = FALSE
)
```

Arguments

attTbl	data.frame, the attribute table returned by the function attTbl.
r	single or multi-layer raster of the class SpatRaster (see help("rast", terra)) used to compute the attTbl.
anchor	integer vector of raster cell numbers.
class	numeric, the classification number to assign to all cells that meet the function conditions.
classVector	numeric vector, if provided, it defines the cells in the attribute table that have already been classified and that have to be ignored by the function (unless the argument overwrite_class = TRUE).
class2cell	logic, attribute the classification number to the cells of the argument anchor. If there is a classVector input, the classification number is only assigned to classVector NA-cells.
class2nbs	logic, attribute the classification number to cells adjacent to the ones of the argument anchor. If there is a classVector input, the classification number is only assigned to classVector NA-cells.
overwrite_class	logic, if there is a classVector input, reclassify cells that were already classified and that meet the function conditions.
plot	logic, plot the class vector output.
writeRaster	filename, if a raster name is provided, save the class vector in a raster file.
overWrite	logic, if the raster names already exist, the existing file is overwritten.

Details

Converts a vector of cell numbers into a class vector. If there is a classVector input, then the class vector is updated assigning a classification number to all cells that meet the function conditions.

Value

Update classVector with the new cells that were classified by the function. If there is no classVector input, the function returns a new class vector. See conditions for more details about class vectors.

See Also

conditions(), anchor.svo(), attTbl()

Examples

```
# DUMMY DATA
##############################################################################
# LOAD LIBRARIES AND DATA
library(scapesClassification)
library(terra)

# CELL NUMBERS OF A DUMMY RASTER (7X7)
```

```
r_cn <- terra::rast(matrix(1:49, nrow = 7, byrow = TRUE), extent=c(0,1,0,1))

# COMPUTE ATTRIBUTE TABLE AND LIST OF NEIGHBORHOODS
at  <- attTbl(r_cn, "dummy_var")
nbs <- ngbList(r_cn)
################################################################################

################################################################################
# ANCHOR.CELL
################################################################################
cv1  <- anchor.cell(attTbl = at, r = r_cn, anchor = 1:7, class  = 10,
                    class2cell = TRUE, class2nbs  = FALSE)

cv2 <- anchor.cell(attTbl = at, r = r_cn, anchor = 1:7, class  = 10,
                   class2cell = FALSE, class2nbs  = TRUE)

cv3 <- anchor.cell(attTbl = at, r = r_cn, anchor = 1:7, class  = 10,
                   class2cell = TRUE, class2nbs  = TRUE)

# Convert class vectors to rasters
r_cv1 <- cv.2.rast(r = r_cn, index = at$Cell, classVector = cv1)
r_cv2 <- cv.2.rast(r = r_cn, index = at$Cell, classVector = cv2)
r_cv3 <- cv.2.rast(r = r_cn, index = at$Cell, classVector = cv3)
################################################################################

################################################################################
# PLOTS
################################################################################
oldpar <- par(mfrow = c(2,2))
m = c(1, 3.5, 2.5, 3.5)

# 1)
plot(r_cv1,type="classes",axes=FALSE,legend=FALSE,asp=NA,colNA="#818792",col="#78b2c4",mar=m)
text(r_cn)
mtext(side=3, line=1, adj=0, cex=1, font=2, "ANCHOR.CELL")
mtext(side=3, line=0, adj=0, cex=0.9, "anchor cells '1:7'")
mtext(side=1, line=0, cex=0.9, adj=0, "class2cell = TRUE; class2nbs = FALSE")
legend("bottomright", ncol = 1, bg = "white", fill = c("#78b2c4", "#818792"),
       legend = c("Classified cells","Unclassified cells"))

# 2)
plot(r_cv2,type="classes",axes=FALSE,legend=FALSE,asp=NA,colNA="#818792",col="#78b2c4",mar=m)
text(r_cn)
mtext(side=3, line=1, adj=0, cex=1, font=2, "ANCHOR.CELL")
mtext(side=3, line=0, adj=0, cex=0.9, "anchor cells '1:7'")
mtext(side=1, line=0, cex=0.9, adj=0, "class2cell = FALSE; class2nbs = TRUE")
legend("bottomright", ncol = 1, bg = "white", fill = c("#78b2c4", "#818792"),
       legend = c("Classified cells","Unclassified cells"))

# 3)
plot(r_cv3,type="classes",axes=FALSE,legend=FALSE,asp=NA,colNA="#818792",col="#78b2c4",mar=m)
text(r_cn)
mtext(side=3, line=1, adj=0, cex=1, font=2, "ANCHOR.CELL")
```

```
mtext(side=3, line=0, adj=0, cex=0.9, "anchor cells '1:7'")
mtext(side=1, line=0, cex=0.9, adj=0, "class2cell = TRUE; class2nbs = TRUE")
legend("bottomright", ncol = 1, bg = "white", fill = c("#78b2c4", "#818792"),
       legend = c("Classified cells","Unclassified cells"))
par(oldpar)
```

`anchor.seed` *Identify seed cells*

Description

Returns a vector of cell numbers at the locations of seed cells and growth buffers.

Usage

```
anchor.seed(
  attTbl,
  ngbList,
  rNumb = FALSE,
  class = NULL,
  cond.filter = NULL,
  cond.seed,
  cond.growth = NULL,
  lag.growth = Inf,
  cond.isol = NULL,
  lag.isol = 1,
  sort.col = NULL,
  sort.seed = "max",
  saveRDS = NULL,
  overWrite = FALSE,
  isol.buff = FALSE,
  silent = FALSE
)
```

Arguments

`attTbl`	data.frame, the attribute table returned by the function `attTbl`.
`ngbList`	list, the list of neighborhoods returned by the function `ngbList`.
`rNumb`	logic, the neighborhoods of the argument `ngbList` are identified by cell numbers (`rNumb=FALSE`) or by row numbers (`rNumb=TRUE`) (see `ngbList`). It is advised to use row numbers for large rasters.
`class`	numeric, the classification number to assign to all cells that meet the function conditions. If `NULL`, a new class number is assigned every time a new seed cell is identified. Growth buffers have the same classification number as the seed cell to which they refer.

cond.filter	character string, defines for what cells the arguments cond.seed, cond.growth and cond.isol have to be evaluated. It can be NULL. Absolute conditions can be used (see conditions).
cond.seed	character string, the conditions to identify seed cells. Absolute conditions can be used (see conditions). It cannot be NULL.
cond.growth	character string, the conditions to define a growth buffer around seed cells. It can be NULL. Absolute and focal cell conditions can be used (see conditions).
lag.growth	0 or Inf, defines the evaluation lag of *focal cell conditions* in cond.growth.
cond.isol	character string, the conditions to define an isolation buffer around seed cells and growth buffers. It can be NULL. Absolute and focal cell conditions can be used (see conditions).
lag.isol	0 or Inf, defines the evaluation lag of *focal cell conditions* in cond.isol.
sort.col	character, the column name in the attTbl on which the sort.seed is based on. It determines in what order seed buffers are computed.
sort.seed	character, the order seed buffers are computed is based on the value seed cells have in the column of attribute table column named sort.col. If sort.seed="max", buffers are computed from the seed cell having the maximum value to the seed cell having the minimum value. If sort.seed="min", buffers are computed in the opposite order.
saveRDS	filename, if a file name is provided save the class vector as an RDS file.
overWrite	logic, if the RDS names already exist, existing files are overwritten.
isol.buff	logic, return the isolation buffer (class = -999).
silent	logic, progress is not printed on the console.

Details

This function implements an algorithm to identify seed cells, growth buffers and isolation buffers.

Condition arguments

The function takes as inputs four sets of conditions with cond.growth and cond.isol taking into account class contiguity and continuity (see conditions):

1. cond.filter, the conditions to define what cells have to be evaluated by the function.
2. cond.seed, the conditions to identify, at each iteration, the seed cell. The seed cell is the cell around which growth and isolation conditions are applied.
3. cond.growth, the conditions to define a buffer around the seed cell.
4. cond.isol, the conditions to isolate one seed cell (and its growth buffer) from another.

Iterations

The argument cond.filter defines the set of cells to be considered by the function.

1. A seed cell is identified based on cond.seed and receives a classification number as specified by the argument class. If class=NULL, then a new class is assigned to every new seed cell.

2. Cells connected with the seed cell meeting the conditions of `cond.growth` are assigned to the same class of the seed cell (growth buffer). The rule evaluation take into account class continuity (see `conditions`).
3. Cells connected with the seed cell (or with its growth buffer) meeting the conditions of `cond.isol` are assigned to the isolation buffer (`class = -999`). The rule evaluation take into account class continuity (see `conditions`).
4. A new seed cell is identified based on `cond.seed` which is now only evaluated for cells that were not identified as seed, growth or isolation cells in previous iterations.
5. A new iteration starts. Seed, growth and isolation cells identified in previous iteration are ignored in successive iterations.
6. The function stops when it cannot identify any new seed cell.

Relative focal cell conditions and evaluation lag

- The arguments `lag.growth` and `lag.isol` control the evaluation lag of *relative focal cell conditions* (see `conditions`).
- When `lag.*` are set to `0`, *relative focal cell conditions* have a standard behavior and compare the values of the `test cells` against the value of the `focal cell`.
- When `lag.*` are set to `Inf`, *relative focal cell conditions* compare the values of the `test cells` against the value of the `seed cell` identified at the start of the iteration.

Value

Class vector. See `conditions` for more details about class vectors.

See Also

`conditions()`, `attTbl()`, `ngbList()`

Examples

```
# DUMMY DATA
################################################################################
# LOAD LIBRARIES
library(scapesClassification)
library(terra)

# LOAD THE DUMMY RASTER
r <- list.files(system.file("extdata", package = "scapesClassification"),
                pattern = "dummy_raster\\.tif", full.names = TRUE)
r <- terra::rast(r)

# COMPUTE THE ATTRIBUTE TABLE
at <- attTbl(r, "dummy_var")

# COMPUTE THE LIST OF NEIGBORHOODS
nbs <- ngbList(r)
```

```
##############################################################################
# EXAMPLE PLOTS
##############################################################################
oldpar <- par(mfrow = c(1,2))
m <- c(4.5, 0.5, 2, 3.2)

# 1a. Do not show isol.buff
as <- anchor.seed(attTbl = at, ngbList = nbs, rNumb = FALSE, class = NULL, silent = TRUE,
                cond.filter = "dummy_var > 1", cond.seed = "dummy_var == max(dummy_var)",
                  cond.growth = "dummy_var<dummy_var[] & dummy_var>2",
                  cond.isol = "dummy_var<dummy_var[]")

plot(cv.2.rast(r,classVector=as), type="classes", mar=m, col=c("#00A600", "#E6E600"),
     axes=FALSE, plg=list(x=1, y=1, cex=.80, title="Classes"))
text(r); lines(r)
mtext(side=3, line=0, cex=1, font=2, adj=0, "1a. Do not show 'isol.buff'")
mtext(side=1, line=0, cex=1, font=2, adj=1, "cond.filter:")
mtext(side=1, line=1, cex=1, font=2, adj=1, "cond.seed:")
mtext(side=1, line=2, cex=1, font=2, adj=1, "cond.growth:")
mtext(side=1, line=3, cex=1, font=2, adj=1, "cond.isol:")
text(xFromCell(r,c(20,43)),yFromCell(r,c(20,43))-0.05,"SEED",col="red",cex=0.80)

# 1b. Show isol.buff
as <- anchor.seed(attTbl = at, ngbList = nbs, rNumb = FALSE, class = NULL, silent = TRUE,
                cond.filter = "dummy_var > 1", cond.seed = "dummy_var == max(dummy_var)",
                  cond.growth = "dummy_var<dummy_var[] & dummy_var>2",
                  cond.isol = "dummy_var<dummy_var[]", isol.buff = TRUE)

plot(cv.2.rast(r,classVector=as), type="classes", col=c("#00000040", "#00A600", "#E6E600"),
     mar=m, axes=FALSE, plg=list(x=1, y=1, cex=.80, title="Classes"))
text(r); lines(r)
mtext(side=3, line=0, cex=1, font=2, adj=0, "1b. Show 'isol.buff' (class=-999)")
mtext(side=1, line=0, cex=1, adj=0, "dummy_var > 1")
mtext(side=1, line=1, cex=1, adj=0, "dummy_var == max(dummy_var)")
mtext(side=1, line=2, cex=1, adj=0, "dummy_var<dummy_var[] & dummy_var>2")
mtext(side=1, line=3, cex=1, adj=0, "dummy_var<dummy_var[]")
text(xFromCell(r,c(20,43)),yFromCell(r,c(20,43))-0.05,"SEED",col="red",cex=0.80)

# 2a. Lag.growth = Inf
as <- anchor.seed(attTbl = at, ngbList = nbs, rNumb = FALSE, class = NULL, silent = TRUE,
                cond.filter = "dummy_var > 1", cond.seed = "dummy_var == max(dummy_var)",
                  cond.growth = "dummy_var<dummy_var[]", lag.growth = Inf)

plot(cv.2.rast(r,classVector=as), type="classes", mar=m, col=c("#00A600"),
     axes=FALSE, plg=list(x=1, y=1, cex=.80, title="Classes"))
text(r); lines(r)
mtext(side=3, line=0, cex=1, font=2, adj=0, "2a. Lag.growth* = Inf")
mtext(side=1, line=0, cex=1, font=2, adj=1, "cond.filter:")
mtext(side=1, line=1, cex=1, font=2, adj=1, "cond.seed:")
mtext(side=1, line=2, cex=1, font=2, adj=1, "cond.growth*:")
mtext(side=1, line=3, cex=1, font=2, adj=1, "cond.isol:")
text(xFromCell(r,c(20)),yFromCell(r,c(20))-0.05,"SEED",col="red",cex=0.80)
```

```
# 2b. Lag.growth = 0
as <- anchor.seed(attTbl = at, ngbList = nbs, rNumb = FALSE, class = NULL, silent = TRUE,
                cond.filter = "dummy_var > 1", cond.seed = "dummy_var == max(dummy_var)",
                  cond.growth = "dummy_var<dummy_var[]", lag.growth = 0)

plot(cv.2.rast(r,classVector=as), type="classes", mar=m, col=c("#00A600", "#E6E600"),
     axes=FALSE, plg=list(x=1, y=1, cex=.80, title="Classes"))
text(r); lines(r)
mtext(side=3, line=0, cex=1, font=2, adj=0, "2b. Lag.growth* = 0")
mtext(side=1, line=0, cex=1, adj=0, "dummy_var > 1")
mtext(side=1, line=1, cex=1, adj=0, "dummy_var == max(dummy_var)")
mtext(side=1, line=2, cex=1, adj=0, "dummy_var < dummy_var[]")
mtext(side=1, line=3, cex=1, adj=0, "NULL")
text(xFromCell(r,c(20,43)),yFromCell(r,c(20,43))-0.05,"SEED",col="red",cex=0.80)

# 3a. Without sorting
as <- anchor.seed(attTbl = at, ngbList = nbs, rNumb = FALSE, class = NULL, silent = TRUE,
                  cond.filter = "dummy_var > 1", cond.seed = "dummy_var >= 5",
                  cond.isol = "dummy_var<dummy_var[]", isol.buff = TRUE)

seeds <- which(!is.na(as) & as !=-999)
cc    <- c("#00000040", terrain.colors(8)[8:1])
plot(cv.2.rast(r,classVector=as), type="classes", mar=m, col=cc,
     axes=FALSE, plg=list(x=1, y=1, cex=.80, title="Classes"))
text(r); lines(r)
mtext(side=3, line=0, cex=1, font=2, adj=0, "3a. Without sorting")
mtext(side=1, line=0, cex=1, font=2, adj=1, "cond.filter:")
mtext(side=1, line=1, cex=1, font=2, adj=1, "cond.seed:")
mtext(side=1, line=2, cex=1, font=2, adj=1, "cond.growth:")
mtext(side=1, line=3, cex=1, font=2, adj=1, "cond.isol:")
text(xFromCell(r,seeds),yFromCell(r,seeds)-0.05,"SEED",col="red",cex=0.80)

# 3b. Sort buffer evaluation based on 'dummy_var' values
as <- anchor.seed(attTbl = at, ngbList = nbs, rNumb = FALSE, class = NULL, silent = TRUE,
                  cond.filter = "dummy_var > 1", cond.seed = "dummy_var >= 5",
                  cond.isol = "dummy_var<dummy_var[]", isol.buff = TRUE,
                  sort.col = "dummy_var", sort.seed = "max")

seeds <- which(!is.na(as) & as !=-999)
plot(cv.2.rast(r,classVector=as), type="classes",col=c("#00000040", "#00A600", "#E6E600"),
     mar=m, axes=FALSE, plg=list(x=1, y=1, cex=.80, title="Classes"))
text(r); lines(r)
mtext(side=3, line=0, cex=1, font=2, adj=0, "3b. Sort.col='dummy_var'; Sort.seed='max'")
mtext(side=1, line=0, cex=1, adj=0, "dummy_var > 1")
mtext(side=1, line=1, cex=1, adj=0, "dummy_var >= 5")
mtext(side=1, line=2, cex=1, adj=0, "NULL")
mtext(side=1, line=3, cex=1, adj=0, "dummy_var < dummy_var[]; isol.buff = -999")
text(xFromCell(r,seeds),yFromCell(r,seeds)-0.05,"SEED",col="red",cex=0.80)
par(oldpar)
```

---

anchor.svo *Anchor cells from spatial vector objects*

Description

Returns a vector of raster cell numbers extracted at the locations of a spatial object.

Usage

```
anchor.svo(
  r,
  dsn,
  only_NAs = FALSE,
  fill_NAs = FALSE,
  plot = FALSE,
  saveRDS = NULL,
  writeRaster = NULL,
  overWrite = FALSE
)
```

Arguments

r	single or multi-layer raster of the class SpatRaster (see help("rast", terra)).
dsn	data source name (filename) or an sf, a Spatial or a SpatVector object.
only_NAs	logic, cell numbers extracted only for incomplete cases at the locations of a spatial object. Incomplete cases are cells having an NA-value in one or more layers of the raster object.
fill_NAs	logic, cell numbers extracted at the locations of a spatial object *and* at contiguous locations that are incomplete cases.
plot	logic, plot anchor cells.
saveRDS	filename, if a file name is provided save the anchor cell vector as an RDS file.
writeRaster	filename, if a raster name is provided save the anchor cell vector as a raster file.
overWrite	logic, if RDS and raster names already exist, existing files are overwritten.

Details

When the arguments only_NA and fill_NAs are FALSE the numeric output is equivalent to the output of the function terra::extract(r, dsn, cells = TRUE)[["cell"]].

Value

Numeric vector of raster cell numbers.

Examples

```
# DUMMY DATA
################################################################################
# LOAD LIBRARIES AND DATA
library(scapesClassification)
library(terra)
```

```
# CELL NUMBERS OF A DUMMY RASTER (7X7)
r_cn <- terra::rast(matrix(1:49, nrow = 7, byrow = TRUE), extent=c(0,1,0,1))

# SET SOME NA-VALUE
r_cn[c(9, 10, 11, 17, 18)] <- NA

# BULD A DUMMY POLYGON
pol <- rbind(c(0,0.95), c(0.28,1), c(0.24, 0.72), c(0.05,0.72), c(0,0.95))
pol <- terra::vect(pol, type="polygons")
################################################################################

################################################################################
# ANCHOR.SVO
################################################################################
ac1 <- anchor.svo(r_cn, pol, only_NAs = FALSE, fill_NAs = FALSE)
ac2 <- anchor.svo(r_cn, pol, only_NAs = TRUE,  fill_NAs = FALSE)
ac3 <- anchor.svo(r_cn, pol, only_NAs = FALSE, fill_NAs = TRUE)
ac4 <- anchor.svo(r_cn, pol, only_NAs = TRUE,  fill_NAs = TRUE)

# RASTER CELL NUMBERS 2 RASTER
r1 <- r_cn; r1[] <- NA; r1[ac1] <- 1
r2 <- r_cn; r2[] <- NA; r2[ac2] <- 1
r3 <- r_cn; r3[] <- NA; r3[ac3] <- 1
r4 <- r_cn; r4[] <- NA; r4[ac4] <- 1
################################################################################

################################################################################
# PLOTS
################################################################################
oldpar <- par(mfrow = c(2,2))
m = c(1, 3.5, 2.5, 3.5)

# 1)
plot(r1, type="classes", col="#78b2c4", colNA="grey", axes=FALSE, legend=FALSE, asp=NA, mar=m)
plot(pol, add = TRUE, lwd = 2.5, border = "red")
text(r_cn)
mtext(side=3, line=1, cex=0.9, adj=0, "only_NAs = FALSE")
mtext(side=3, line=0, cex=0.9, adj=0, "fill_NAs = FALSE")
ac1 <- paste("ac =", paste0(sort(ac1), collapse = ","))
mtext(side=1, line=0, cex=0.9, adj=0, ac1)
legend("bottomleft", ncol = 1, bg = "white",
       legend = c("Anchor cell (ac)", "Polygon"), fill = c("#78b2c4", "red"))

# 2)
plot(r2, type="classes", col="#78b2c4", colNA="grey", axes=FALSE,legend=FALSE, asp=NA, mar=m)
plot(pol, add = TRUE, lwd = 2.5, border = "red")
text(r_cn)
mtext(side=3, line=1, cex=0.9, adj=0, "only_NAs = TRUE")
mtext(side=3, line=0, cex=0.9, adj=0, "fill_NAs = FALSE")
ac2 <- paste("ac =", paste0(sort(ac2), collapse = ","))
mtext(side=1, line=0, cex=0.9, adj=0, ac2)
legend("bottomleft", ncol = 1, bg = "white",
       legend = c("Anchor cell (ac)", "Polygon"), fill = c("#78b2c4", "red"))
```

```
# 3)
plot(r3, type="classes", col="#78b2c4", colNA="grey", axes=FALSE, legend=FALSE, asp=NA, mar=m)
plot(pol, add = TRUE, lwd = 2.5, border = "red")
text(r_cn)
mtext(side=3, line=1, cex=0.9, adj=0, "only_NAs = FALSE")
mtext(side=3, line=0, cex=0.9, adj=0, "fill_NAs = TRUE")
ac3 <- paste("ac =", paste0(sort(ac3), collapse = ","))
mtext(side=1, line=0, cex=0.9, adj=0, ac3)
legend("bottomleft", ncol = 1, bg = "white",
       legend = c("Anchor cell (ac)", "Polygon"), fill = c("#78b2c4", "red"))

# 4)
plot(r4, type="classes", col="#78b2c4", colNA="grey", axes=FALSE, legend=FALSE, asp=NA, mar=m)
plot(pol, add = TRUE, lwd = 2.5, border = "red")
text(r_cn)
mtext(side=3, line=1, cex=0.9, adj=0, "only_NAs = TRUE")
mtext(side=3, line=0, cex=0.9, adj=0, "fill_NAs = TRUE")
ac4 <- paste("ac =", paste0(sort(ac4), collapse = ","))
mtext(side=1, line=0, cex=0.9, adj=0, ac4)
legend("bottomleft", ncol = 1, bg = "white",
       legend = c("Anchor cell (ac)", "Polygon"), fill = c("#78b2c4", "red"))
par(oldpar)
```

`attTbl` *Attribute table*

Description

Converts a single or a multi-layer raster into an attribute table (`data.frame`).

Usage

```
attTbl(r, var_names = NULL)
```

Arguments

`r`	single or multi-layer raster of the class `SpatRaster` (see `help("rast", terra)`).
`var_names`	character vector, raster layers' names in the attribute table. If `NULL`, then the original layers' names are used.

Details

Attribute tables come with a column named `"Cell"` which stores raster cell numbers and associate each row of the attribute table with a cell of the raster object. The remaining columns of the attribute table store the data contained in the raster layers. Note that only raster cells having no missing value in no layer (**complete cases**) are included in the attribute table.

Value

data.frame

Note

Attribute table contains only complete cases, i.e., raster cells having a value for every layer of the stack.

Examples

```
library(scapesClassification)
library(terra)

## CREATE A DUMMY RASTER ##
r <- terra::rast(matrix(c(NA,100,100,NA,100,100,0,0,0),
                        nrow = 3,
                        ncol = 3,
                        byrow = TRUE))

## RASTER CELL NUMBERS ##
rcn <- r; rcn[] <- 1:9

## PLOT DATA AND CELL NUMBERS ##
oldpar <- par(mfrow = c(1,2))
m <- c(4, 1, 4, 1)

plot(r, col="grey90", colNA="red3", mar=m, asp = NA, axes=FALSE, legend=FALSE)
text(r)
lines(r)
mtext(side=3, line=0.2, adj=0, cex=1.5, font=2, "Dummy_var")
legend("bottomright", ncol=1, bg="white", fill=c("red3"),
       legend = c("NA cells (1 and 4)"))

plot(rcn, col="grey90", mar=m, asp=NA, axes=FALSE, legend=FALSE)
text(rcn)
lines(rcn)
mtext(side=3, line=0.2, adj=0, cex=1.5, font=2, "Cell numbers")
par(oldpar)

## VISUALIZE ATTRIBUTE TABLE ##

at <- attTbl(r, var_names = c("dummy_var"))
at

# Note that cells 1 and 4 have missing values and therefore are not included in the table
any(at$Cell %in% c(1,4))
```

`classify.all` *Classify All Unclassified Cells*

Description

Classify all cells in `classVector` that have not yet been classified based on contiguity and continuity conditions.

Usage

```
classify.all(attTbl, ngbList, rNumb = FALSE, classVector)
```

Arguments

`attTbl`	data.frame, the attribute table returned by the function `attTbl`.
`ngbList`	list, the list of neighborhoods returned by the function `ngbList`.
`rNumb`	logic, the neighborhoods of the argument `ngbList` are identified by cell numbers (`rNumb=FALSE`) or by row numbers (`rNumb=TRUE`) (see `ngbList`). It is advised to use row numbers for large rasters.
`classVector`	numeric vector, defines the cells in the attribute table that have already been classified. See `conditions` for more information about class vectors.

Details

The neighborhood of unclassified cells is considered. Among neighbors, the class with the highest number of members is assigned to the unclassified cell. If two or more classes have the same number of members, then one of these classes is assigned randomly to the unclassified cell. The function considers *class continuity*, thus, even cells that at first were not contiguous to any class will be classified if continuous with at least one cell having a class (see `conditions`).

Value

Update `classVector` with the new cells that were classified by the function. See `conditions` for more details about class vectors.

See Also

`attTbl()`, `ngbList()`, `conditions()`

Examples

```
# DUMMY DATA
##############################################################################
# LOAD LIBRARIES
library(scapesClassification)
library(terra)
```

```
# LOAD THE DUMMY RASTER
r <- list.files(system.file("extdata", package = "scapesClassification"),
                pattern = "dummy_raster\\.tif", full.names = TRUE)
r <- terra::rast(r)

# COMPUTE THE ATTRIBUTE TABLE
at <- attTbl(r, "dummy_var")

# COMPUTE THE LIST OF NEIGBORHOODS
nbs <- ngbList(r)

################################################################################
# CLASSIFY.ALL
################################################################################
# compute example class vector
cv <- cond.4.all(attTbl = at, cond = "dummy_var <= 1", class = 1)
# update example calss vector
cv <- cond.4.all(attTbl = at, cond = "dummy_var <= 3", class = 2,
                 classVector = cv) # input previous class vector

# classify all unclassified cells
ca <- classify.all(attTbl = at, ngbList = nbs, rNumb = TRUE, classVector = cv)


# Convert class vectors into rasters
r_cv <- cv.2.rast(r, at$Cell, classVector = cv)
r_ca <- cv.2.rast(r, at$Cell,classVector = ca)
################################################################################
# PLOTS
################################################################################
oldpar <- par(mfrow = c(1,2))
m <- c(3, 1, 5, 1)

# 1)
plot(r_cv, type="classes", axes=FALSE, legend=FALSE, asp=NA, mar=m,
     colNA="#818792", col=c("#78b2c4", "#cfc1af"))
text(r)
mtext(side=3, line=2, adj=0, cex=1, font=2, "COND.4.ALL")
mtext(side=3, line=1, adj=0, cex=0.9, "Step1: 'dummy_var<=1', Class: 1")
mtext(side=3, line=0, adj=0, cex=0.9, "Step2: 'dummy_var<=3', Class: 2")
legend("bottomright", bg = "white", fill = c("#78b2c4", "#cfc1af", "#818792"),
       legend = c("Class 1", "Class 2", "Unclassified cells"))

# 2)
plot(r_ca, type="classes", axes=FALSE, legend=FALSE, asp=NA,  mar=m,
     colNA="#818792", col=c("#78b2c4", "#cfc1af"))
text(r)
mtext(side=3, line=2, adj=0, cex=1, font=2, "CLASSIFY.ALL")
mtext(side=3, line=1, adj=0, cex=0.9, "Classify all unclassified cells")
legend("bottomright", bg = "white", fill = c("#78b2c4", "#cfc1af", "#818792"),
       legend = c("Class 1", "Class 2"))
par(oldpar)
```

cond.4.all *Test conditions for all cells*

Description

Evaluate conditions for unclassified cells and classify them if conditions are true.

Usage

```
cond.4.all(attTbl, cond, classVector = NULL, class, ovw_class = FALSE)
```

Arguments

attTbl	data.frame, the attribute table returned by the function attTbl.
cond	character string, the conditions a cell have to meet to be classified as indicated by the argument class. If there is a classVector input, the classification number is only assigned to unclassified cells unless the argument ovw_class = TRUE. See conditions for more details.
classVector	numeric vector, if provided, it defines the cells in the attribute table that have already been classified. See conditions for more information about class vectors.
class	numeric, the classification number to assign to all cells that meet the function conditions.
ovw_class	logic, if there is a classVector input, reclassify cells that were already classified and that meet the function conditions.

Details

- The function evaluates the conditions of the argument conditions for all unclassified cells (i.e., classVector NA-cells).
- Cells that meet the function conditions are classified as indicted by the argument class.
- *Absolute test cell conditions* can be used (see conditions).

Value

Update classVector with the new cells that were classified by the function. If there is no classVector input, the function returns a new class vector. See conditions for more details about class vectors.

See Also

conditions(), attTbl(), cond.4.nofn(), cond.reclass()

Examples

```
# DUMMY DATA
################################################################################
library(scapesClassification)
library(terra)

# LOAD THE DUMMY RASTER
r <- list.files(system.file("extdata", package = "scapesClassification"),
                pattern = "dummy_raster\\.tif", full.names = TRUE)
r <- terra::rast(r)

# COMPUTE THE ATTRIBUTE TABLE
at <- attTbl(r, "dummy_var")

# COMPUTE THE LIST OF NEIGBORHOODS
nbs <- ngbList(r)

################################################################################
# COND.4.ALL
################################################################################
# compute new class vector
# conditions: "dummy_var == 1"
cv1 <- cond.4.all(attTbl = at, cond = "dummy_var <= 1", class = 1)

unique(cv1) # one class (class 1)

# update class vector `cv1`
# conditions: "dummy_var <= 3"
cv2   <- cond.4.all(attTbl = at, cond = "dummy_var <= 3", class = 2,
                    classVector = cv1) # input previous class vector

unique(cv2) # two classes (class 1 and class 2)

# convert class vector 2 raster
r_cv1 <- cv.2.rast(r, at$Cell, classVector = cv1)
r_cv2 <- cv.2.rast(r, at$Cell, classVector = cv2)

################################################################################
# PLOTS
################################################################################
oldpar <- par(mfrow = c(1,2))
m <- c(4.5, 0.5, 2, 3.2)

# 1.
r_cv1[which(is.na(values(r_cv1)))] <- 10
plot(r_cv1, type="classes", mar=m, col=c("#78b2c4","#818792"), axes=FALSE,
     plg=list(x=1, y=1, cex=.80, title="Classes",legend=c("1", "NA")))
text(r); lines(r)
mtext(side=3, line=1, adj=0, cex=1, font=2, "1. COND.4.ALL")
mtext(side=3, line=0, adj=0, cex=0.9, "New class vector")
mtext(side=1, line=0, cex=0.9, adj=0, "Rule: 'dummy_var <= 1'")
mtext(side=1, line=1, cex=0.9, adj=0, "Class: 1")
```

```
# 2.
r_cv2[which(is.na(values(r_cv2)))] <- 10
plot(r_cv2, type="classes", mar=m, col=c("#78b2c4","#cfad89","#818792"), axes=FALSE,
     plg=list(x=1, y=1, cex=.80, title="Classes",legend=c("1", "2", "NA")))
text(r); lines(r)
mtext(side=3, line=1, adj=0, cex=1, font=2, "2. COND.4.ALL")
mtext(side=3, line=0, adj=0, cex=0.9, "Update class vector (class 1 not overwritten)")
mtext(side=1, line=0, cex=0.9, adj=0, "Rule: 'dummy_var <= 3'")
mtext(side=1, line=1, cex=0.9, adj=0, "Class: 2")
par(oldpar)
```

`cond.4.nofn` *Test conditions for neighbors and neighbors of neighbors*

Description

Evaluate conditions for cells neighboring specific classes and classify them if conditions are true.

Usage

```
cond.4.nofn(
  attTbl,
  ngbList,
  rNumb = FALSE,
  classVector,
  class,
  nbs_of,
  cond,
  min.bord = NULL,
  max.iter = +Inf,
  peval = 1,
  directional = FALSE,
  ovw_class = FALSE,
  hgrowth = FALSE
)
```

Arguments

`attTbl`	data.frame, the attribute table returned by the function `attTbl`.
`ngbList`	list, the list of neighborhoods returned by the function `ngbList`.
`rNumb`	logic, the neighborhoods of the argument `ngbList` are identified by cell numbers (`rNumb=FALSE`) or by row numbers (`rNumb=TRUE`) (see `ngbList`). It is advised to use row numbers for large rasters.
`classVector`	numeric vector, defines the cells in the attribute table that have already been classified. See `conditions` for more information about class vectors.

class	numeric, the classification number to assign to all cells that meet the function conditions.
nbs_of	numeric or numeric vector, indicates the class(es) of focal and anchor cells. Conditions are only evaluated at positions adjacent to anchor and focal cells. If the classification number assigned with the argument class is also included in the argument nbs_of, the function takes into account *class continuity* (see conditions).
cond	character string, the conditions a cell have to meet to be classified as indicated by the argument class. The classification number is only assigned to unclassified cells unless the argument ovw_class = TRUE. See conditions for more details.
min.bord	numeric value between 0 and 1. A test cell is classified if conditions are true and if among its bordering cells a percentage equal or greater than min.bord belong to one of the classes of nbs_of. Percentages are computed counting only valid neighbors (i.e., neighbors with complete cases).
max.iter	integer, the maximum number of iterations.
peval	numeric value between 0 and 1. If *absolute or relative neighborhood conditions* are considered, test cells are classified if the number of positive evaluations is equal or greater than the percentage specified by the argument peval (see conditions).
directional	logic, absolute or relative neighborhood conditions are tested using the *directional neighborhood* (see conditions).
ovw_class	logic, reclassify cells that were already classified and that meet the function conditions.
hgrowth	logic, if true the classes in nbs_of are treated as discrete raster objects and the argument class is ignored.

Details

- The function evaluates the conditions of the argument cond for all unclassified cells in the neighborhood of focal and anchor cells (specified by the argument nbs_of). Unclassified cells are NA-cells in classVector.
- Cells that meet the function conditions are classified as indicted by the argument class.
- *Class continuity* is considered if the classification number assigned with the argument class is also included in the argument nbs_of. This means that, at each iteration, newly classified cells become focal cells and conditions are tested in their neighborhood.
- All types of conditions can be used. The condition string can only include one neighborhood condition ('{}') (see conditions).

Homogeneous growth (hgrowth)

If the argument hgrowth is true the classes in nbs_of are treated as discrete raster objects and the argument class is ignored. Iterations proceed as follow:

- cells contiguous to the first element of nbs_of are evaluated against the classification rules and, when evaluations are true, cells are assigned to that element;
- the same process is repeated for cells contiguous to the second element of nbs_of, then for cells contiguous to the third element and so on until the last element of nbs_of;

- once cells contiguous to the last element of `nbs_of` are evaluated the iteration is complete;
- cells classified in one iteration become focal cells in the next iteration;
- a new iteration starts as long as new cells were classified in the previous iteration and if the iteration number < `max.iter`.

Value

Update `classVector` with the new cells that were classified by the function. See `conditions` for more details about class vectors.

See Also

`conditions()`, `attTbl()`, `ngbList()`

Examples

```
# DUMMY DATA
################################################################################################
# LOAD LIBRARIES
library(scapesClassification)
library(terra)

# LOAD THE DUMMY RASTER
r <- list.files(system.file("extdata", package = "scapesClassification"),
                pattern = "dummy_raster\\.tif", full.names = TRUE)
r <- terra::rast(r)

# COMPUTE THE ATTRIBUTE TABLE
at <- attTbl(r, "dummy_var")

# COMPUTE THE LIST OF NEIGBORHOODS
nbs <- ngbList(r)

# SET A DUMMY FOCAL CELL (CELL #25)
at$cv[at$Cell == 25] <- 0

# SET FIGURE MARGINS
m <- c(2, 8, 2.5, 8)

################################################################################################
# ABSOLUTE TEST CELL CONDITION - NO CLASS CONTINUITY
################################################################################################

# conditions: "dummy_var >= 3"
cv1 <- cond.4.nofn(attTbl = at, ngbList = nbs,

                   # CLASS VECTOR - INPUT
                   classVector = at$cv,

                   # CLASSIFICATION NUMBER
                   class = 1,
```

```
                    # FOCAL CELL CLASS
                    nbs_of = 0,

                    # ABSOLUTE TEST CELL CONDITION
                    cond = "dummy_var >= 3")

# CONVERT THE CLASS VECTOR INTO A RASTER
r_cv1 <- cv.2.rast(r, at$Cell,classVector = cv1, plot = FALSE)

# PLOT
plot(r_cv1, type="classes", axes=FALSE, legend = FALSE, asp = NA, mar = m,
     colNA="#818792", col=c("#78b2c4", "#cfad89"))
text(r)
mtext(side=3, line=1, adj=0, cex=1, font=2, "CONDITION: ABSOLUTE TEST CELL")
mtext(side=3, line=0, adj=0, cex=1, "Class continuity: NO")
mtext(side=1, line=0, cex=0.9, adj=0, "Rule: 'dummy_var >= 3'")
legend("bottomright", bg = "white", fill = c("#78b2c4", "#cfad89", "#818792"),
       legend = c("Focal cell", "Classified cells", "Unclassified cells"))

##############################################################################################
# ABSOLUTE TEST CELL CONDITION - WITH CLASS CONTINUITY
##############################################################################################

# conditions: "dummy_var >= 3"
cv2 <- cond.4.nofn(attTbl = at, ngbList = nbs, classVector = at$cv,

                   # CLASSIFICATION NUMBER
                    class = 1,

                    nbs_of = c(0,  # FOCAL CELL CLASS
                               1), # CLASSIFICATION NUMBER

                    # ABSOLUTE CONDITION
                    cond = "dummy_var >= 3")

# CONVERT THE CLASS VECTOR INTO A RASTER
r_cv2 <- cv.2.rast(r, at$Cell,classVector = cv2, plot = FALSE)

# PLOT
plot(r_cv2, type="classes", axes=FALSE, legend = FALSE, asp = NA, mar = m,
     colNA="#818792", col=c("#78b2c4", "#cfad89"))
text(r)
mtext(side=3, line=1, adj=0, cex=1, font=2, "CONDITION: ABSOLUTE TEST CELL")
mtext(side=3, line=0, adj=0, cex=1, "Class continuity: YES")
mtext(side=1, line=0, cex=0.9, adj=0, "Rule: 'dummy_var >= 3'")
legend("bottomright", bg = "white", fill = c("#78b2c4", "#cfad89", "#818792"),
       legend = c("Focal cell", "Classified cells", "Unclassified cells"))

##############################################################################################
# ABSOLUTE NEIGHBORHOOD CONDITION
##############################################################################################

# conditions: "dummy_var{} >= 3"
```

```
cv3 <- cond.4.nofn(attTbl = at, ngbList = nbs, classVector = at$cv, nbs_of = c(0,1), class = 1,

                   # ABSOLUTE NEIGHBORHOOD CONDITION
                   cond = "dummy_var{} >= 3",

                   # RULE HAS TO BE TRUE FOR 100% OF THE EVALUATIONS
                   peval = 1)

# CONVERT THE CLASS VECTOR INTO A RASTER
r_cv3 <- cv.2.rast(r, at$Cell,classVector = cv3, plot = FALSE)

#PLOT
plot(r_cv3, type="classes", axes=FALSE, legend = FALSE, asp = NA, mar = m,
     colNA="#818792", col=c("#78b2c4", "#cfad89"))
text(r)
mtext(side=3, line=1, adj=0, cex=1, font=2, "CONDITION: ABSOLUTE NEIGHBORHOOD")
mtext(side=3, line=0, adj=0, cex=1, "Class continuity: YES")
mtext(side=1, line=0, cex=0.9, adj=0, "Rule: 'dummy_var{ } >= 3'")
mtext(side=1, line=0, cex=0.9, adj=1, "('{ }' cell neighborhood)")
mtext(side=1, line=1, cex=0.9, adj=0, "Fn_perc: 1 (100%)")
legend("bottomright", bg = "white", fill = c("#78b2c4", "#cfad89", "#818792"),
       legend = c("Focal cell", "Classified cells", "Unclassified cells"))

################################################################################
# RELATIVE NEIGHBORHOOD CONDITION
################################################################################

# conditions: "dummy_var > dummy_var{}"
cv4 <- cond.4.nofn(attTbl = at, ngbList = nbs, classVector = at$cv, nbs_of = c(0,1), class = 1,

                   # RELATIVE NEIGHBORHOOD CONDITION
                   cond = "dummy_var > dummy_var{}",

                   # RULE HAS TO BE TRUE FOR AT LEAST 60% OF THE EVALUATIONS
                   peval = 0.6)


# CONVERT THE CLASS VECTOR INTO A RASTER
r_cv4 <- cv.2.rast(r, at$Cell, classVector = cv4, plot = FALSE)

#PLOT
plot(r_cv4, type="classes", axes=FALSE, legend = FALSE, asp = NA, mar = m,
     colNA="#818792", col=c("#78b2c4", "#cfad89"))
text(r)
mtext(side=3, line=1, adj=0, cex=1, font=2, "CONDITION: RELATIVE NEIGHBORHOOD")
mtext(side=3, line=0, adj=0, cex=1, "Class continuity: YES")
mtext(side=1, line=0, cex=0.9, adj=0, "Rule: 'dummy_var > dummy_var{ }'")
mtext(side=1, line=0, cex=0.9, adj=1, "('{ }' cell neighborhood)")
mtext(side=1, line=1, cex=0.9, adj=0, "Fn_perc: 0.6 (60%)")
legend("bottomright", bg = "white", fill = c("#78b2c4", "#cfad89", "#818792"),
       legend = c("Focal cell", "Classified cells", "Unclassified cells"))

################################################################################
```

```
# RELATIVE FOCAL CELL CONDITION
##############################################################################################

# conditions: "dummy_var > dummy_var[]"
cv5 <- cond.4.nofn(attTbl = at, ngbList = nbs, classVector = at$cv, nbs_of = c(0,1), class = 1,

                   # RELATIVE FOCAL CELL CONDITION
                   cond = "dummy_var > dummy_var[]")


# CONVERT THE CLASS VECTOR INTO A RASTER
r_cv5 <- cv.2.rast(r, at$Cell,classVector = cv5, plot = FALSE)

#PLOT
plot(r_cv5, type="classes", axes=FALSE, legend = FALSE, asp = NA, mar = m,
     colNA="#818792", col=c("#78b2c4", "#cfad89"))
text(r)
mtext(side=3, line=1, adj=0, cex=1, font=2, "CONDITION: RELATIVE FOCAL CELL")
mtext(side=3, line=0, adj=0, cex=1, "Class continuity: YES")
mtext(side=1, line=0, cex=0.9, adj=0, "Rule: 'dummy_var > dummy_var[ ]'")
mtext(side=1, line=0, cex=0.9, adj=1, "('[ ]' focal cell)")
legend("bottomright", bg = "white", fill = c("#78b2c4", "#cfad89", "#818792"),
       legend = c("Focal cell", "Classified cells", "Unclassified cells"))

##############################################################################################
# HOMOGENEOUS GROWTH
##############################################################################################

# Dummy raster objects 1 and 2
ro <- as.numeric(rep(NA, NROW(at)))
ro[which(at$dummy_var == 10)] <- 1
ro[which(at$dummy_var == 8)] <- 2

# Not homogeneous growth
nhg <- cond.4.nofn(attTbl = at, ngbList = nbs, classVector = ro,
                   nbs_of = 1, class = 1, # GROWTH ROBJ 1
                   cond = "dummy_var <= dummy_var[] & dummy_var != 1")

nhg <- cond.4.nofn(attTbl = at, ngbList = nbs, classVector = nhg, # UPDATE nhg
                   nbs_of = 2, class = 2, # GROWTH ROBJ 2
                   cond = "dummy_var <= dummy_var[] & dummy_var != 1")


# Homogeneous growth
hg <- cond.4.nofn(attTbl = at, ngbList = nbs, classVector = ro,
                  nbs_of = c(1, 2), class = NULL,
                  cond = "dummy_var <= dummy_var[] & dummy_var != 1",
                  hgrowth = TRUE) # HOMOGENEOUS GROWTH

# Convert class vectors into rasters
r_nhg <- cv.2.rast(r, at$Cell,classVector = nhg, plot = FALSE)
r_hg  <- cv.2.rast(r, at$Cell,classVector = hg, plot = FALSE)
```

```
# Plots
oldpar <- par(mfrow = c(1,2))
m <- c(3, 1, 5, 1)

# Original raster objects (for plotting)
r_nhg[at$dummy_var == 10] <- 3
r_nhg[at$dummy_var == 8]  <- 4

r_hg[at$dummy_var == 10] <- 3
r_hg[at$dummy_var == 8]  <- 4
#t
# 1)
plot(r_nhg, type="classes", axes=FALSE, legend=FALSE, asp=NA, mar = m,
     colNA="#818792", col=c("#78b2c4", "#cfc1af", "#1088a0", "#cfad89"))
text(r)
mtext(side=3, line=1, adj=0, cex=1, font=2, "RASTER OBJECTS GROWTH")
mtext(side=3, line=0, adj=0, cex=0.9, "Not homogeneous (hgrowth = FALSE)")
mtext(side=1, line=0, cex=0.9, adj=0, "Growth rule:")
mtext(side=1, line=1, cex=0.9, adj=0, "'dummy_var<=dummy_var[ ] & dummy_var!=1''")
legend("topleft", bg = "white", y.intersp= 1.3,
       fill = c("#1088a0", "#cfc1af", "#78b2c4", "#cfc1af", "#818792"),
       legend = c("RO1", "RO2", "RO1 - growth", "RO2 - growth", "Unclassified cells"))
# 2)
plot(r_hg, type="classes", axes=FALSE, legend=FALSE, asp=NA, mar = m,
     colNA="#818792", col=c("#78b2c4", "#cfc1af", "#1088a0", "#cfad89"))
text(r)
mtext(side=3, line=1, adj=0, cex=1, font=2, "RASTER OBJECTS GROWTH")
mtext(side=3, line=0, adj=0, cex=0.9, "Homogeneous (hgrowth = TRUE)")
mtext(side=1, line=0, cex=0.9, adj=0, "Growth rule:")
mtext(side=1, line=1, cex=0.9, adj=0, "'dummy_var<=dummy_var[ ] & dummy_var!=1''")
legend("topleft", bg = "white", y.intersp= 1.3,
       fill = c("#1088a0", "#cfc1af", "#78b2c4", "#cfc1af", "#818792"),
       legend = c("RO1", "RO2", "RO1 - growth", "RO2 - growth", "Unclassified cells"))
par(oldpar)
```

`cond.parse` *Parse conditions*

Description

Parse the condition string so that it can be evaluated by the `cond.*` functions. Intended for internal use only.

Usage

```
cond.parse(names_attTbl, cond)
```

Arguments

names_attTbl	character vector, the column (i.e. variable) names of the attribute table returned by the function attTbl.
cond	character string, the condition string used by the cond.* functions to classify raster cells (see conditions).

Value

The function returns a two-element list. The first element contains the parsed conditions to be evaluated by the cond.* functions. The second element defines the condition type each variable refers to.

See Also

cond.4.all(), cond.4.nofn(), anchor.seed(), cond.reclass(), conditions()

---

cond.reclass *Test conditions and reclassify*

Description

Evaluate conditions for cells of a class and reclassify them if conditions are true.

Usage

```
cond.reclass(
  attTbl,
  ngbList = NULL,
  rNumb = FALSE,
  classVector,
  class,
  cond,
  reclass,
  peval = 1
)
```

Arguments

attTbl	data.frame, the attribute table returned by the function attTbl.
ngbList	list, the list of neighborhoods returned by the function ngbList. Only necessary if using an *absolute neighborhood condition* (see conditions).
rNumb	logic, the neighborhoods of the argument ngbList are identified by cell numbers (rNumb=FALSE) or by row numbers (rNumb=TRUE) (see ngbList). It is advised to use row numbers for large rasters.
classVector	numeric vector, defines the cells in the attribute table that have already been classified. See conditions for more information about class vectors.

class	numeric or numeric vector, indicates the class(es) for which conditions have to be evaluated.
cond	character string, the conditions a cell have to meet to be classified as indicated by the argument reclass. See conditions for more details.
reclass	numeric, the classification number to assign to all cells that meet the function conditions.
peval	numeric value between 0 and 1. If *absolute neighborhood conditions* are considered, test cells are classified if the number of positive evaluations is equal or greater than the percentage specified by the argument peval (see conditions).

Details

- The function evaluates the conditions of the argument cond for all cells in the classes of the argument class.
- Cells that meet the function conditions are re-classified as indicted by the argument reclass.
- Absolute test cell and neighborhood conditions can be used. The condition string can only include one neighborhood condition ('{}') (see conditions).

Value

Update classVector with the new cells that were classified by the function. See conditions for more information about class vectors.

See Also

conditions(), attTbl(), ngbList()

Examples

```
# DUMMY DATA
##############################################################################################
library(scapesClassification)
library(terra)

# LOAD THE DUMMY RASTER
r <- list.files(system.file("extdata", package = "scapesClassification"),
                pattern = "dummy_raster\\.tif", full.names = TRUE)
r <- terra::rast(r)

# COMPUTE THE ATTRIBUTE TABLE
at <- attTbl(r, "dummy_var")

# COMPUTE THE LIST OF NEIGBORHOODS
nbs <- ngbList(r)

##############################################################################
# RECLASS.NBS
##############################################################################

# Compute an example class vector
```

```
cv <- cond.4.all(attTbl = at, cond = "dummy_var > 1", class = 1)

# Reclassify cells
cr <- cond.reclass(attTbl = at, ngbList = nbs,

                   # CLASS VECTOR COMPUTED WITH THE RULE "dummy_var > 1"
                   classVector = cv,

                   # CELLS TO RECLASSIFY HAVE THIS CLASS
                   class = 1,

                   # ABSOLUTE NEIGHBORHOOD CONDITION
                   cond = "dummy_var{} >= 5", peval = 1,

                   # NEW CLASSIFICATION NUMBER
                   reclass = 2)

# Convert class vectors to rasters
r_cv <- cv.2.rast(r, cv)
r_cr <- cv.2.rast(r, cr)

#############################################################################
# PLOTS
#############################################################################
oldpar <- par(mfrow = c(1,2))
m <- c(3, 1, 5, 4)

# 1.
r_cv[which(is.na(values(r_cv)))] <- 10
plot(r_cv, type="classes", mar=m, col=c("#78b2c4","#818792"), axes=FALSE,
     plg=list(x=1, y=1, cex=.80, title="Classes",legend=c("1", "NA")))
text(r); lines(r)
mtext(side=3, line=1, adj=0, cex=1, font=2, "1. COND.4.ALL")
mtext(side=3, line=0, adj=0, cex=0.9, "New class vector")
mtext(side=1, line=0, cex=1, adj=0, "Class: 1")
mtext(side=1, line=1, cex=1, adj=0, "Rule: 'dummy_var > 1'")

# 2.
r_cr[which(is.na(values(r_cr)))] <- 10
plot(r_cr, type="classes", mar=m, col=c("#78b2c4","#cfad89","#818792"), axes=FALSE,
     plg=list(x=1, y=1, cex=.80, title="Classes",legend=c("1", "reclass", "NA")))
text(r); lines(r)
mtext(side=3, line=1, adj=0, cex=1, font=2, "2. COND.RECLASS")
mtext(side=3, line=0, adj=0, cex=0.9, "Reclassify cells meeting conditions")
mtext(side=1, line=0, cex=1, adj=0, "Class: 2")
mtext(side=1, line=1, cex=1, adj=0, "Rule: 'dummy_var{ } >= 5'; peval = 1")
par(oldpar)
```

---

conditions *scapesClassification conditions*

Description

Check for spelling and syntax errors in conditions (cond argument) and detect the type of conditions being used.

Usage

```
conditions(names_attTbl, cond, silent = FALSE)
```

Arguments

names_attTbl	character vector, the column (i.e. variable) names of the attribute table returned by the function attTbl.
cond	character string, the condition string used by the cond.* functions to classify raster cells (see conditions).
silent	logic, when true, the function returns only error messages.

Details

Conditions (alias classification rules)

- Classification rules evaluate either to true or false and determine what raster cells have to be classified.
- Conditions are passed to scapesClassification functions as a single character string. They can consist of combination of variables names (as named in the attribute table, see attTbl), arithmetic (+|-|*|/|^|%%|%/%), relational (>|<|>=|<=|==|!=|%/%) and logic operators (&||) and base R functions (e.g., abs(variable_name)).
- All variables included in an attribute table (see attTbl) can be included in a condition string by name (e.g., var name = "dummy_var"; condition = "dummy_var > 1").

Class vectors

- Class vectors map raster cells to numeric classes.
- The n^th^ element of a class vector stores the class of the raster cell stored in the n^th^ row of the corresponding attribute table (see attTbl).
- Class vectors can serve also as a function input. As inputs, they provide information about the groups of cells that have already been classified.
- Every time a class vector is provided as a function input, it is *updated* by assigning a numeric class to *unclassified cells* that meet function conditions.
- Unclassified cells are represented as NA values.

Rule evaluation: Global evaluation

- Classification rules are applied to all unclassified raster cells.
- Function using *global evaluation*: cond.4.all.

Rule evaluation: Focal evaluation

- Classification rules are applied only to raster cells at specific locations and are based on class (dis)contiguity and class continuity.
- **Class contiguity:**
  classification rules are applied only to raster cells contiguous to focal cells (identified in the `cond.*` functions by the argument `nbs_of`).
- **Class continuity:**
  join into the same class cells that respect the same rules and that are connected to the same focal cells. This means that, at each iteration, newly classified cells become focal cells and conditions are tested in their neighborhood.
- Function using *focal evaluation*: `anchor.seed`, `cond.4.nofn`, `cond.reclass`, `reclass.nbs` and `classify.all`.

Focal evaluation: Definitions

- **Cell neighborhood:** a cell with coordinates `(x, y)` has 8 neighbors with coordinates: `(x±1, y)`, `(x, y±1)` and `(x±1, y±1)`. Cells on the edge of a raster have less than 8 neighbors. See `ngbList`.
- **Focal cell:** cells whose neighbors are evaluated against the classification rule(s). In the classification functions *focal cells* are identified by the argument `nbs_of`.
- **Test cell:** the cell in the neighborhood of the focal cell that is being tested. At turns all cells in the neighborhood of a focal cell are tested against the classification rule(s).
- **Directional neighborhood:** it consists of the intersection between the focal and the test cell neighborhoods.

Condition type: Absolute conditions

1) Absolute test cell condition: compares cell values against a threshold value.

- This type of condition applies to all functions with a `conditions` argument.
- In global evaluations all cells meeting absolute conditions receive a classification number. In focal evaluations all `test cells` meeting absolute conditions receive a classification number.
- *Examples of valid conditions:* `"variable_A > 1 & variable_B != 0"`; `"(variable_A^2 < 50 & variable_B == 0) | abs(variable_C) > 50"`.
  Functions: `anchor.seed`, `cond.4.all`, `cond.4.nofn` and `cond.reclass`.

2) Absolute neighborhood condition: compares the values of the `test cell` and of its `neighborhood` against a threshold value.

- An absolute neighborhood condition is identified by a variable name followed by curly brackets (e.g., `"variable_name{}"`).
- A maximum of 9 evaluations are performed for each test cell (the test cell itself and the cells of its neighborhood are compared against a threshold value).
- Test cells receive a classification number if the rule is true for at least as many evaluations as the ones specified by the argument `peval`. The argument `peval` ranges from 0 to 1. When 9 evaluations are performed, `peval = 1` means that all `9` evaluations have to be true; `peval = 0.5` means that at least `4.5` (rounded to `5`) evaluations have to be true.

- Only one neighborhood rule is allowed for each condition string (e.g., it is not possible to have a condition string like "variable_A{} > 0 & variable_B{} > 1").
- The function cond.4.nofn can consider a directional neighborhood instead of the test cell neighborhood by setting the argument directional = TRUE.
- *Example of valid conditions:* "variable_A{} > 1 & abs(variable_B) != 0".
  Functions: cond.4.nofn and cond.reclass.

Condition type: Relative conditions

1) Relative focal cell condition: compares the test cell value against the focal cell value.

- A relative focal cell condition is identified by a variable name followed by square brackets (e.g., "variable_name[]").
- Rules are defined repeating twice the same variable name, once with square brackets and once without. Square brackets indicate the focal cell value. As an example, the rule "dummy_var < dummy_var[]" compares the value of the the test cell ("dummy_var") against the value of the focal cell ("dummy_var[]").
- Test cells are classified if the rule is true.
- *Examples of valid conditions:* "variable_A > variable_A[]"; "(variable_A > variable_A[] & variable_B{} < 10) | variable_C > 1". Note that the last example is a combination of absolute and focal cell conditions.
  Functions: anchor.seed and cond.4.nofn.

2) Relative neighborhood condition: compares the values of the test cell against the values of the test cell neighborhood.

- A relative neighborhood condition is identified by a variable name followed by curly brackets (e.g., "variable_name{}").
- Rules are defined repeating twice the same variable name, once with curly brackets and once without. Curly brackets indicate the test cell neighborhood. As an example, the rule 'dummy_var < dummy_var{}' compares the value of the the test cell (dummy_var) against the values of cells included in the test cell neighborhood (dummy_var{}).
- A maximum of 8 evaluations are performed for each test cell (the test cell is compared against each cell included in its neighborhood).
- Test cells receive a classification number if the rule is true for at least as many evaluations as the ones specified by the argument peval. The argument peval ranges from 0 to 1. When 8 evaluations are performed, peval = 1 means that all 8 evaluations have to be true; peval = 0.5 means that at least 4 evaluations have to be true.
- Only one neighborhood rule is allowed for each condition string (e.g., it is not possible to have a condition string like "variable_A{} > 0 & variable_B{} > variable_B").
- The function cond.4.nofn can consider a directional neighborhood instead of the test cell neighborhood by setting the argument directional = TRUE.
- *Example of valid conditions:* "variable_A > variable_A{}"; "(variable_A > variable_A{} & variable_B != variable_B[]) | variable_C > 1". Note that the last example is a combination of absolute and relative conditions.
  Functions: cond.4.nofn and cond.reclass.

Value

An error message if the function finds spelling or syntax errors or a string with the types of rules detected in the condition string.

See Also

cond.4.all(), cond.4.nofn(), anchor.seed(), cond.reclass(), conditions()

anchor.seed(), attTbl(), cond.4.all(), cond.4.nofn(), cond.reclass(), classify.all()

Examples

```
# LOAD LIBRARIES
library(scapesClassification)
library(terra)

################################################################################
# TYPES OF CONDITIONS
################################################################################

# As an example consider an attribute with the following columns
names_attTbl <- c("bathymetry", "slope")

# And the following conditions
cond <- "bathymetry>10"
conditions(names_attTbl, cond)

cond <- "bathymetry[]>bathymetry | abs(slope{}) < 5"
conditions(names_attTbl, cond)

cond <- "bathymetry[]>bathymetry | abs(slope{}) < slope"
conditions(names_attTbl, cond)

################################################################################
# FOCAL EVALUATION DEFINITIONS
################################################################################

# CELL NUMBERS OF A DUMMY RASTER (7X7)
r   <- terra::rast(matrix(1:49, nrow = 7, byrow = TRUE), extent=c(0,7,0,7))
nbs <- ngbList(r)

# CLASS VECTOR WITH ONE TEST AND ONE FOCAL CELL
cv <- as.numeric(rep(NA, 49))
cv[c(32, 25)] <- c(1, 2) # tc (class 1), fc (class 2)
r_cv <- cv.2.rast(r, classVector = cv)

# POLYGONS REPRESENTING NEIGHBORHOODS
fcn <- rbind(c(2,5), c(5,5), c(5,2), c(2,2), c(2,5))
fcn <- terra::vect(fcn, type="polygons")

tcn <- rbind(c(2,4), c(5,4), c(5,1), c(2,1), c(2,4))
tcn <- terra::vect(tcn, type="polygons")
```

```
# PLOT - FOCAL EVALUATION DEFINITIONS
m <- c(3.5, 8, 1.2, 8)
plot(r_cv, type = "class", asp = NA, legend = FALSE, axes = FALSE, mar = m,
     col = c("goldenrod3","#1088a0"), colNA= "grey90")
text(r)
mtext(side=3, line=0, adj=0, cex=1, font=2, "FOCAL EVALUATION")
mtext(side=1, line=0, adj=0, cex=0.9,
      "Focal cell neighborhood: 17, 18, 19, 24, 26, 31, 32, 33")
mtext(side=1, line=1, cex=0.9, adj=0,
      "Test cell neighborhood: 24, 25, 26, 31, 33, 38, 39, 40")
mtext(side=1, line=2, cex=0.9, adj=0,
     "Directional neighborhood: 24, 25, 26, 31, 32, 33")
lines(fcn, col="#1088a0", lwd=2)
lines(tcn, col="#cfad8999", lwd=2)
legend("bottomleft", ncol = 1, bg = "white", y.intersp= 1.3,
       legend = c("Focal cell", "Test cell"), fill = c("#1088a0", "goldenrod3"))
```

`cv.2.rast` *Class vector to raster*

Description

Transform a class vector or a generic vector into a raster.

Usage

```
cv.2.rast(
  r,
  classVector,
  index = NULL,
  plot = FALSE,
  type = "classes",
  writeRaster = NULL,
  overWrite = FALSE
)
```

Arguments

`r`	raster object.
`classVector`	numeric vector, the values to be assigned to the cell numbers indicated by `index`.
`index`	numeric vector, the cell numbers of the argument `r` to which assign the values of the argument `classVector`. If `NULL`, the column `Cell` of the attribute table `attTbl(r)` is used (see `attTbl`).
`plot`	logic, plot the raster.
`type`	character, type of map/legend. One of "continuous", "classes", or "interval".
`writeRaster`	filename, if a raster name is provided save the raster to a file.
`overWrite`	logic, if the raster names already exist, the existing file is overwritten.

Details

The arguments `index` and `vector` need to have the same length. The function assign the values of `vector` at the positions of `index` to an empty raster having the same spatial properties of the raster `r`.

Value

A class vector or a generic vector transformed into a raster.

Examples

```
library(scapesClassification)
library(terra)

# LOAD THE DUMMY RASTER
r <- list.files(system.file("extdata", package = "scapesClassification"),
                pattern = "dummy_raster\\.tif", full.names = TRUE)
r <- terra::rast(r)

# COMPUTE THE ATTRIBUTE TABLE
at <- attTbl(r, "dummy_var")

# COMPUTE THE LIST OF NEIGBORHOODS
nbs <- ngbList(r)

# Compute an example class vector
cv <- cond.4.all(attTbl = at, cond = "dummy_var > 1", class = 1)

# Class vector to raster
cv.2.rast(r, cv, plot = TRUE)
text(r) # add raster values
```

---

ngb8 *Eight neighbors*

Description

Return the 8 neighbors, as cell numbers, of each cell on a raster.

Usage

```
ngb8(n_row, n_col)
```

Arguments

`n_row`	Integer. The number of rows of a Raster or object.
`n_col`	Integer. The number of columns of a Raster object.

Details

A cell with coordinates (x, y) has 8 neighbors with coordinates: (x±1, y), (x, y±1) and (x±1, y±1). Cells on the edge of a raster have less than 8 neighbors. The function identifies the neighbors of a cell as cell numbers.

Value

Named list, the nth element of the list corresponds to the 8 adjacent cell numbers of the nth cell on the Raster* object.

See Also

ngbList()

Examples

```
## Matrix m mocking a raster of 3 rows and 4 columns
m <- matrix(1:12, nrow = 3, ncol = 4, byrow = TRUE)
m

nbs <- ngb8(3, 4)
nbs
```

---

ngbList *List of neighborhoods*

Description

Computes the neighborhoods of the cells of a raster. Neighborhoods are not computed for cells with missing values.

Usage

```
ngbList(r, rNumb = FALSE, attTbl = NULL)
```

Arguments

r	single or multi-layer raster of the class SpatRaster (see help("rast", terra)).
rNumb	logic, the neighbors of a raster cell are identified by **cell numbers** (rNumb=FALSE) or by **row numbers** (rNumb=TRUE). If true, the argument attTbl cannot be NULL.
attTbl	data.frame, the attribute table returned by the function attTbl (see attTbl). It is required only if the argument rNumb=TRUE.

Details

Definition of neighborhood

- A cell with coordinates `(x, y)` has 8 neighbors with coordinates: `(x±1, y)`, `(x, y±1)` and `(x±1, y±1)`. Cells on the edge of a raster have less than 8 neighbors.

Neighborhoods (`rNumb=FALSE`)

- Neighbors are identified by their cell numbers if the argument `rNumb=FALSE`.

Neighborhoods (`rNumb=TRUE`)

- Neighbors are identified by their positions in the attribute table (i.e. row numbers) if the argument `rNumb=TRUE`;
- When the argument `rNumb = TRUE`, neighbors with missing values are omitted;
- `(scapes)Classifications` are faster when the list of neighborhoods uses row numbers.

Neighborhood names

The list of neighborhoods is named.

- When `rNumb = FALSE`, the element name identifies the raster cell to which the neighborhood refers. For instance, the element with name `"n"` stores the neighborhood of the raster cell `n`.
- When `rNumb = TRUE`, the element name identifies the row number to which the neighborhood refers. For instance, the element with name `"n"` stores the neighborhood of the raster cell located in the `n`th row of the attribute table (`attTbl$Cell[n]`).

Value

Named list of integer vectors.

Note

- There is always a correspondence between the indices of the attribute table (`attTbl`) and the indices of the list of neighborhoods: the 1st element of the list corresponds to the neighbors of the cell stored in the 1st row of the attribute table; the 2nd element corresponds to the 2nd row; etc.
- There is a correspondence between the raster cell number and the indices of the list of neighborhoods only when no missing value is present in the raster.

See Also

`ngb8()`, `attTbl()`

Examples

```
library(scapesClassification)
library(terra)

## CREATE A DUMMY RASTER AND COMPUTE ATTRIBUTE TABLE ##
r <- terra::rast(matrix(c(NA,100,100,NA,100,100,0,0,0),
```

```
                        nrow = 3,
                        ncol = 3,
                        byrow = TRUE))

at <- attTbl(r, var_names = c("dummy_var"))

## RASTER CELL NUMBERS ##
rcn <- r; rcn[] <- 1:9

## PLOT DATA AND CELL NUMBERS ##
oldpar <- par(mfrow = c(1,2))
m <- c(4, 1, 4, 1)

plot(r, col="grey90", colNA="red3", mar=m, asp=NA, axes=FALSE, legend=FALSE)
text(r)
lines(r)
mtext(side=3, line=0.2, adj=0, cex=1.5, font=2, "Dummy_var")
legend("bottomright", ncol = 1, bg = "white", fill = c("red3"),
       legend = c("NA cells (1 and 4)"))

plot(rcn, col="grey90", mar=m, asp = NA, axes=FALSE, legend=FALSE)
text(rcn)
lines(rcn)
mtext(side=3, line=0.2, adj=0, cex=1.5, font=2, "Cell numbers")
par(oldpar)

## NEIGHBORHOODS - CELL NUMBERS ##

# Cells 1 and 4 are omitted because they are NAs
nbs_CELL <- ngbList(r, rNumb = FALSE)
nbs_CELL


## NEIGHBORHOODS - ROW NUMBERS ##

# Cells 1 and 4 are omitted because they are NAs
nbs_ROW <- ngbList(r, rNumb = TRUE, attTbl = at)
nbs_ROW

# Numbers in 'nbs_ROW' refer to row numbers
# (e.g. number 1 refers to the cell #2)
at$Cell[1]

# (e.g. number 2 refers to the cell #3)
at$Cell[2]

# (e.g. number 5 refers to the cell #7)
at$Cell[5]


## CONSIDER THE NEIGHBORHOOD OF CELL #2 ##

# Cell #2 corresponds to the 1st element of both 'nbs_CELL' and 'nbs_ROW'
```

```
# because raster cell 1 is an NA-cell
r[1]

# Neighborhood cell #2 corresponds to cells:
nbs_CELL[1]

# Neighborhood cell #2 corresponds to rows:
nbs_ROW[1]

# Rows can be converted to cell numbers
at$Cell[ nbs_ROW[[1]] ]

# Note that 'at$Cell[ nbs_ROW[[1]] ]' is not equal to 'nbs_CELL'
identical( at$Cell[ nbs_ROW[[1]] ] , nbs_CELL[[1]] )

# This is because raster cells 1 and 4 (NA-cells) are omitted in 'nbs_ROW'
setdiff(nbs_CELL[[1]], at$Cell[ nbs_ROW[[1]] ])
r[c(1,4)]
```

`obj.border` *Borders of raster objects*

Description

Identify the borders of raster objects.

Usage

```
obj.border(group, ngbList, silent = FALSE)
```

Arguments

`group`	named list, each element represents a raster object composed of several raster cells. If there are NA values on the raster surface, raster cells must be identified by attribute table row indices (each corresponding to a raster cell) (see `attTbl`).
`ngbList`	list, the list of neighborhoods returned by the function `ngbList`. The list of neighborhoods has to be computed setting the argument `rNumb=TRUE`.
`silent`	logic, progress bar is not printed on the console.

Value

The function returns a named list of object borders. List names identify the objects; list element values identify the raster cells comprising the borders.

Note

- Note that `group` has to be a **named** list whose names correspond to raster object IDs.
- If there are NA values on the raster surface, raster cells must be identified by attribute table row indices (each corresponding to a raster cell). Row indices can be converted into raster cells using the `Cell` column of the attribute table (e.g. `attTbl$Cell[indices]`) (see `attTbl`).

See Also

attTbl(), ngbList(), obj.nbs()

Examples

```
# DUMMY DATA
##############################################################################################
# LOAD LIBRARIES
library(scapesClassification)
library(terra)

# LOAD THE DUMMY RASTER
r <- list.files(system.file("extdata", package = "scapesClassification"),
                pattern = "dummy_raster\\.tif", full.names = TRUE)
r <- terra::rast(r)

# ADD NA-VALUE
r[11] <- NA

# COMPUTE THE ATTRIBUTE TABLE
at <- attTbl(r, "dummy_var")

# COMPUTE THE LIST OF NEIGBORHOODS
nbs <- ngbList(r, rNumb=TRUE, attTbl=at) # rnumb MUST be true to use obj.border

##############################################################################
# COMPUTE RASTER OBJECTS
##############################################################################
at$cv <- anchor.seed(at, nbs, silent=TRUE, class = NULL, rNumb=TRUE,
                     cond.filter = "dummy_var > 1",
                     cond.seed   = "dummy_var==max(dummy_var)",
                     cond.growth = "dummy_var<dummy_var[]",
                     lag.growth  = 0)

# Raster objects
RO <- split(1:NROW(at), at$cv)

print(RO) # values are attribute table row indices

##############################################################################
# COMPUTE BORDERS
##############################################################################
RO_bd <- obj.border(RO, nbs, silent = TRUE)

RO_bd1 <- at$Cell[RO_bd[["1"]]] # Convert row numbers to cell numbers
RO_bd2 <- at$Cell[RO_bd[["2"]]] # Convert row numbers to cell numbers

print(RO_bd)  # attribute table row indices
print(RO_bd1) # cell numbers
print(RO_bd2) # cell numbers

##############################################################################
```

```
# PLOT BORDERS
################################################################################
plot(cv.2.rast(r,at$cv), type="classes", col=c("#E6E600","#00A600"),
     main="Borders")
points(terra::xyFromCell(r, RO_bd1), pch=20, col="blue")
points(terra::xyFromCell(r, RO_bd2), pch=20, col="red")
text(xyFromCell(r, 11), "NA\nvalue")
```

`obj.nbs` *Shared borders of raster objects*

Description

Identify the shared borders of neighboring raster objects.

Usage

```
obj.nbs(grp.bord, ngbList, only_grp = NULL, silent = FALSE)
```

Arguments

`grp.bord`	named list, the list of borders returned by the function `obj.border`.
`ngbList`	list, the list of neighborhoods returned by the function `ngbList`. The list of neighborhoods has to be computed setting the argument `rNumb=TRUE`.
`only_grp`	character vector. If `NULL`, all IDs in `grp.bord` are considered. If IDs are provided, then they are the only ones considered.
`silent`	logic, progress bar is not printed on the console.

Value

The function returns a named list. Each element represents a raster object as identified by the list names and contains a nested named list. The names of the nested lists are the IDs of neighboring raster objects and their values identify the raster cells comprising the shared borders.

Note

If there are NA values on the raster surface, raster cells are identified by attribute table row indices (each corresponding to a raster cell). Row indices can be converted into raster cells using the `Cell` column of the attribute table (e.g. `attTbl$Cell[indices]`) (see `attTbl`).

See Also

`attTbl()`, `ngbList()`, `obj.border()`

Examples

```
# DUMMY DATA
################################################################################
# LOAD LIBRARIES
library(scapesClassification)
library(terra)

# LOAD THE DUMMY RASTER
r <- list.files(system.file("extdata", package = "scapesClassification"),
                pattern = "dummy_raster\\.tif", full.names = TRUE)
r <- terra::rast(r)

# ADD ONE NA VALUE
r[11] <- NA

# COMPUTE THE ATTRIBUTE TABLE
at <- attTbl(r, "dummy_var")

# COMPUTE THE LIST OF NEIGBORHOODS
nbs <- ngbList(r, rNumb=TRUE, attTbl=at) # rnumb MUST be true to use obj.nbs

################################################################################
# COMPUTE RASTER OBJECTS
################################################################################
at$cv <- anchor.seed(at, nbs, silent=TRUE, class = NULL, rNumb=TRUE,
                     cond.filter = "dummy_var > 1",
                     cond.seed   = "dummy_var==max(dummy_var)",
                     cond.growth = "dummy_var<dummy_var[]",
                     lag.growth  = 0)

RO <- split(1:NROW(at), at$cv)

print(RO)

################################################################################
# COMPUTE BORDERS
################################################################################
RO_bd <- obj.border(RO, nbs, silent = TRUE)

################################################################################
# COMPUTE SHARED BORDERS
################################################################################
RO_sbd <- obj.nbs(RO_bd, nbs, silent = TRUE)

# Convert row indices to cell numbers
RO_sbd1 <- RO_sbd[["1"]]
RO_sbd1 <- at$Cell[unlist(RO_sbd1)]

RO_sbd2 <- RO_sbd[["2"]]
RO_sbd2 <- at$Cell[unlist(RO_sbd2)]

# Borders in `RO_sbd` are identified by row indices
```

```
print(RO_sbd[["1"]]) # Row indices
print(RO_sbd1) # Cell numbers

print(RO_sbd[["2"]])  # Row indices
print(RO_sbd2) # Cell numbers

# Neighbor objects
names(RO_sbd[["1"]]) # RO1 has one neighbor, RO2
names(RO_sbd[["2"]]) # RO2 has one neighbor, RO1

################################################################################
# PLOT BORDERS
################################################################################
plot(cv.2.rast(r,at$cv), type="classes", col=c("#E6E600","#00A600"),
     main="Shared borders")
points(terra::xyFromCell(r, RO_sbd1), pch=20, col="blue")
points(terra::xyFromCell(r, RO_sbd2), pch=20, col="red")
text(xyFromCell(r, 11), "NA\nvalue")
```

---

peak.cell *Identify local maxima or minima*

Description

Identify local maxima or minima on a raster surface.

Usage

```
peak.cell(attTbl, ngbList, rNumb = FALSE, p_col, p_fun = "max", p_edge = FALSE)
```

Arguments

attTbl	data.frame, the attribute table returned by the function attTbl.
ngbList	list, the list of neighborhoods returned by the function ngbList.
rNumb	logic, the neighborhoods of the argument ngbList are identified by cell numbers (rNumb=FALSE) or by row numbers (rNumb=TRUE) (see ngbList). It is advised to use row numbers for large rasters.
p_col	character, the column of the attribute table over which maxima or minima are searched.
p_fun	character, if 'max' the function searches for local maxima; if 'min' the function searches for local minima.
p_edge	logic, if false local maxima or minima are not searched on edge cells. Edge cells are considered cells on the edge of the raster and cell neighboring NA-cells.

Details

- A cell constitutes a *local maximum* if its elevation value is larger than the values of all the cells in its neighborhood (see `ngbList`).
- A cell constitutes a *local minimum* if its elevation value is smaller than the values of all the cells in its neighborhood (see `ngbList`).

Value

A `classVector` with peak cells identified by the numeric class `1`. See `conditions` for more details about class vectors.

See Also

`conditions()`, `attTbl()`, `ngbList()`

Examples

```
# DUMMY DATA
################################################################################
# LOAD LIBRARIES
library(scapesClassification)
library(terra)

# LOAD THE DUMMY RASTER
r <- list.files(system.file("extdata", package = "scapesClassification"),
                pattern = "dummy_raster\\.tif", full.names = TRUE)
r <- terra::rast(r)

# COMPUTE THE ATTRIBUTE TABLE
at <- attTbl(r, "dummy_var")

# COMPUTE THE LIST OF NEIGBORHOODS
nbs <- ngbList(r)
################################################################################

# PEAK.CELL
################################################################################
# p_edge = FALSE
pc_a <- peak.cell(attTbl = at, ngbList = nbs, rNumb = FALSE,
                  p_col = "dummy_var", p_fun = "max", p_edge = FALSE)

# p_edge = TRUE
pc_b <- peak.cell(attTbl = at, ngbList = nbs, rNumb = FALSE,
                  p_col = "dummy_var", p_fun = "max", p_edge = TRUE)

# CONVERT THE CLASS VECTORS INTO RASTERS
r_pca <- cv.2.rast(r, at$Cell, classVector = pc_a, plot = FALSE)
r_pcb <- cv.2.rast(r, at$Cell, classVector = pc_b, plot = FALSE)
################################################################################

#PLOTS
```

```
##############################################################################
oldpar <- par(mfrow = c(1,2))
m <- c(4, 1, 4, 1)

# PLOT 1 - p_edge = FALSE
plot(r_pca, axes=FALSE, legend=FALSE, asp=NA, mar=m,
     colNA="#818792", col=c("#78b2c4", "#cfad89"))
text(r)
mtext(side=3, line=1, adj=0, cex=1, font=2, "PEAK.CELL")
mtext(side=3, line=0, adj=0, cex=0.9, "p_edge = FALSE")
legend("bottomright", bg = "white",
       legend = c("Peak cell", "Unclassified cells"),
       fill = c("#cfad89", "#818792"))

# PLOT 2 - p_edge = TRUE
plot(r_pcb, axes=FALSE, legend=FALSE, asp=NA, mar=m,
    colNA="#818792", col=c("#78b2c4", "#cfad89"))
text(r)
mtext(side=3, line=1, adj=0, cex=1, font=2, "PEAK.CELL")
mtext(side=3, line=0, adj=0, cex=0.9, "p_edge = TRUE")
legend("bottomright", bg = "white",
       legend = c("Peak cell", "Unclassified cells"),
       fill = c("#cfad89", "#818792"))
par(oldpar)
```

---

pi.add *Position index addition*

Description

Add new raster objects based on position index values.

Usage

```
pi.add(
  attTbl,
  ngbList,
  rNumb = FALSE,
  RO,
  mainPI = NULL,
  secPI = NULL,
  add.mPI = NULL,
  add.sPI = NULL,
  cond.filter = NULL,
  min.N = NULL,
  plot = FALSE,
  r = NULL
)
```

Arguments

attTbl	data.frame, the attribute table returned by the function attTbl.
ngbList	list, the list of neighborhoods returned by the function ngbList.
rNumb	logic, the neighborhoods of the argument ngbList are identified by cell numbers (rNumb=FALSE) or by row numbers (rNumb=TRUE) (see ngbList). It is advised to use row numbers for large rasters.
RO	column name, the name of the column with the raster object IDs.
mainPI	column name, the name of the column with main position index values.
secPI	column name, the name of the column with secondary position index values.
add.mPI	numeric, threshold of main position index values. Cells with values above the threshold are flagged as cells potentially being part of new raster objects.
add.sPI	numeric, threshold of secondary position index values. Cells with values above the threshold flagged as cells potentially being part of new raster objects.
cond.filter	character string, defines what cells have to be considered by the function the arguments. Test cell absolute conditions can be used (see conditions).
min.N	numeric, the minimum number of cells a raster object has to have to be included in the function output.
plot	logic, plot the results.
r	single or multi-layer raster of the class SpatRaster (see help("rast", terra)) used to compute the attribute table. Required only if plot = TRUE.

Details

New raster objects are added based on position index values. Two different position indices can be passed to the function (mainPI and secPI).

- Input raster objects are assigned to the same class to flag cells that are part of raster objects;
- Cells with values above mainPI **OR** above mainPI are flagged as cells potentially being part of new raster objects;
- If not connected to any of the existing raster objects, the groups of cells above position index values are assigned to new raster objects.
- Only raster objects with at least as many cells as specified by the argument min.N are included in the function output.
- If both mainPI and secPI are equal to NULL, the function will exclusively filter raster objects based on their size (min.N).

Value

The function returns a class vector with raster objects IDs. The vector has length equal to the number of rows of the attribute table. NA values are assigned to cells that do not belong to any raster object.

Note

Raster objects are added only if they do not share any border with input raster objects.

See Also

attTbl(), ngbList(), rel.pi(), pi.sgm(), conditions()

Examples

```
# DUMMY DATA
##############################################################################################
# LOAD LIBRARIES
library(scapesClassification)
library(terra)

# LOAD THE DUMMY RASTER
r <- list.files(system.file("extdata", package = "scapesClassification"),
                pattern = "dummy_raster\\.tif", full.names = TRUE)
r <- terra::rast(r)

# COMPUTE THE ATTRIBUTE TABLE
at <- attTbl(r, "dummy_var")

# COMPUTE THE LIST OF NEIGBORHOODS
nbs <- ngbList(r, attTbl=at)

##############################################################################
# COMPUTE RASTER OBJECTS
##############################################################################
at$RO[at$dummy_var==8] <- 1
at$RO <- cond.4.nofn(at, nbs, classVector = at$RO, class=1, nbs_of = 1,
                     cond = "dummy_var < dummy_var[] & dummy_var > 2")

# One raster object
unique(at$RO)

##############################################################################
# POSITION INDEX
##############################################################################
at$PI <- (at$dummy_var - mean(at$dummy_var))/stats::sd(at$dummy_var)

##############################################################################
# POSITION INDEX ADDITION
##############################################################################
RO1 <- pi.add(at, nbs,
              RO = "RO",     # Raster objects
              mainPI = "PI", # PI addition layer
              add.mPI = 1,   # add disjoint objects with PI values > 1
              plot = FALSE, r = r)

##############################################################################
# PLOT
##############################################################################
# Convert class vectors to raster
r_RO  <- cv.2.rast(r = r, classVector = at$RO)
r_RO1 <- cv.2.rast(r = r, classVector = RO1)
```

```
# Plot
oldpar <- par(mfrow = c(1,2))
m <- c(4.5, 0.5, 2, 3.2)

terra::plot(r_RO, type="classes", main="Raster objects - Input", mar=m,
            plg=list(x=1, y=1, cex=0.9))

terra::plot(r_RO1, type="classes", main="Raster objects - Output", mar=m,
            plg=list(x=1, y=1, cex=0.9))
text(xyFromCell(r,at$Cell), as.character(round(at$PI,2)),
cex = 0.8) # visualize relPI
text(0.01, 1, "Add on PI >= 1", adj=c(0,0), cex = 0.8)
par(oldpar)

# Two raster object
unique(RO1)
```

`pi.sgm` *Position index segmentation*

Description

Segment raster objects based on position index values.

Usage

```
pi.sgm(
  attTbl,
  ngbList,
  rNumb = FALSE,
  RO,
  mainPI = NULL,
  secPI = NULL,
  cut.mPI = NULL,
  cut.sPI = NULL,
  min.N = NULL,
  plot = FALSE,
  r = NULL
)
```

Arguments

`attTbl`	data.frame, the attribute table returned by the function `attTbl`.
`ngbList`	list, the list of neighborhoods returned by the function `ngbList`.
`rNumb`	logic, the neighborhoods of the argument `ngbList` are identified by cell numbers (`rNumb=FALSE`) or by row numbers (`rNumb=TRUE`) (see `ngbList`). It is advised to use row numbers for large rasters.

RO	column name, the name of the column with the raster object IDs.
mainPI	column name, the name of the column with main position index values.
secPI	column name, the name of the column with secondary position index values.
cut.mPI	numeric, threshold of main position index values. Cells with values below the threshold are excluded from raster objects.
cut.sPI	numeric, threshold of secondary position index values. Cells with values below the threshold are excluded from raster objects.
min.N	numeric, the minimum number of cells a raster object has to have to be included in the function output.
plot	logic, plot the results.
r	single or multi-layer raster of the class SpatRaster (see help("rast", terra)) used to compute the attribute table. Required only if plot = TRUE.

Details

Raster objects are segmented based on position index values. Two different position indices can be passed to the function (mainPI and secPI).

- Input raster objects are assigned to the same class to flag cells that are part of raster objects;
- Cells with values below mainPI **OR** below mainPI are flagged as not being part of any raster object;
- Each non-continuous group of raster object cells will identify an output raster object.
- Only raster objects with at least as many cells as specified by the argument min.N are included in the function output.
- If both mainPI and secPI are equal to NULL, the function will exclusively filter raster objects based on their size (min.N).

Value

The function returns a class vector with raster objects IDs. The vector has length equal to the number of rows of the attribute table. NA values are assigned to cells that do not belong to any raster object.

See Also

attTbl(), ngbList(), rel.pi(), pi.add()

Examples

```
# DUMMY DATA
##############################################################################
# LOAD LIBRARIES
library(scapesClassification)
library(terra)

# LOAD THE DUMMY RASTER
r <- list.files(system.file("extdata", package = "scapesClassification"),
```

```
                pattern = "dummy_raster\\.tif", full.names = TRUE)
r <- terra::rast(r)

# COMPUTE THE ATTRIBUTE TABLE
at <- attTbl(r, "dummy_var")

# COMPUTE THE LIST OF NEIGBORHOODS
nbs <- ngbList(r, attTbl=at)

################################################################################
# COMPUTE RASTER OBJECTS
################################################################################
at$RO <- anchor.seed(at, nbs, silent=TRUE, class = NULL, rNumb=TRUE,
                     cond.filter = "dummy_var > 1",
                     cond.seed   = "dummy_var==max(dummy_var)",
                     cond.growth = "dummy_var<dummy_var[]",
                     lag.growth  = Inf)

# One input raster object
unique(at$RO)

################################################################################
# NORMALIZED RELATIVE POSITION INDEX
################################################################################
at$relPI <- rel.pi(attTbl = at, RO = "RO", el = "dummy_var", type = "n")

################################################################################
# POSITION INDEX SEGMENTATION
################################################################################
RO1 <- pi.sgm(at, nbs,
              RO = "RO",        # Raster objects
              mainPI = "relPI", # PI segmentation layer
              cut.mPI = 0,      # segment on relPI values <= 0
              plot = FALSE, r = r)

################################################################################
# PLOT
################################################################################
# Convert class vectors to raster
r_RO  <- cv.2.rast(r = r, classVector = at$RO)
r_RO1 <- cv.2.rast(r = r, classVector = RO1)

# Plot
oldpar <- par(mfrow = c(1,2))
m <- c(4.5, 0.5, 2, 3.2)

terra::plot(r_RO, type="classes", main="Raster objects - Input", mar=m,
plg=list(x=1, y=1, cex=0.9))

terra::plot(r_RO1, type="classes", main="Raster objects - Output", mar=m,
            plg=list(x=1, y=1, cex=0.9))
text(xyFromCell(r,at$Cell), as.character(round(at$relPI,2))) # visualize relPI
text(0.01, 1, "Cut on relPI <= 0", adj=c(0,1), cex = 0.8)
```

```
par(oldpar)

# Two output raster objects
unique(RO1)
```

`reclass.nbs` *Reclassify neighbors*

Description

Evaluate if members of two classes are contiguous and, if they are, one of them is reclassified.

Usage

```
reclass.nbs(
  attTbl,
  ngbList,
  rNumb = FALSE,
  classVector,
  nbs_of,
  class,
  reclass,
  reclass_all = TRUE
)
```

Arguments

`attTbl`	data.frame, the attribute table returned by the function `attTbl`.
`ngbList`	list, the list of neighborhoods returned by the function `ngbList`.
`rNumb`	logic, the neighborhoods of the argument `ngbList` are identified by cell numbers (`rNumb=FALSE`) or by row numbers (`rNumb=TRUE`) (see `ngbList`). It is advised to use row numbers for large rasters.
`classVector`	numeric vector, defines the cells in the attribute table that have already been classified. See `conditions` for more information about class vectors.
`nbs_of`	numeric or numeric vector, indicates the class(es) of focal and anchor cells.
`class`	numeric or numeric vector, cells of classes `class` adjacent to cells belonging to one of the classes of `nbs_of` are reclassified as indicated by the argument `reclass`.
`reclass`	numeric, the classification number to assign to all cells that meet the function conditions.
`reclass_all`	logic, all cells of class `class` are also reclassified if they are connected to a reclassified cell.

Details

- The function evaluates if a cell of class `class` is adjacent to a cell of class `nbs_of` and, if it is, it is reclassifies as indicated by the argument `reclass`.
- If the argument `reclass_all = TRUE`, all cells of class `class` are also reclassified if they are connected to a reclassified cell.

Value

Update `classVector` with the new cells that were classified by the function. See `conditions` for more information about class vectors.

See Also

`attTbl()`, `ngbList()`, `cond.reclass()`, `classify.all()`

Examples

```
library(scapesClassification)
library(terra)

# LOAD THE DUMMY RASTER
r <- list.files(system.file("extdata", package = "scapesClassification"),
                pattern = "dummy_raster\\.tif", full.names = TRUE)
r <- terra::rast(r)

# COMPUTE THE ATTRIBUTE TABLE
at <- attTbl(r, "dummy_var")

# COMPUTE THE LIST OF NEIGBORHOODS
nbs <- ngbList(r)

################################################################################
# RECLASS.NBS
################################################################################

# Compute an inital class vector with `cond.4.all`
cv <- cond.4.all(attTbl = at, cond = "dummy_var > 5", class = 1)

# Update the class vector with a second class
cv <- cond.4.all(attTbl = at, cond = "dummy_var >= 2", class = 2,
                 classVector = cv)


# Reclassify cells of class 2 adjacent to cells of class 1

# reclass_all = FALSE
rc1 <- reclass.nbs(attTbl = at, ngbList = nbs,

                   # CLASS VECTOR `cv`
                   classVector = cv,
```

```
                    # CELLS OF CLASS...
                    class = 2,

                    # ...ADJACENT TO CELLS OF ANOTHER CLASS...
                    nbs_of = 1,

                    # ...WILL BE RECLASSIFIED...
                    reclass = 3,

                    # NO MORE RECLASSIFICATIONS
                    reclass_all = FALSE)

# reclass_all = TRUE
rc2 <- reclass.nbs(attTbl = at, ngbList = nbs,

                    # CLASS VECTOR `cv`
                    classVector = cv,

                    # CELLS OF CLASS...
                    class = 2,

                    # ...ADJACENT TO CELLS OF ANOTHER CLASS...
                    nbs_of = 1,

                    # ...WILL BE RECLASSIFIED...
                    reclass = 3,

                    # ...AND SO ALL CELLS OF CLASS 1 CONNECTED TO A RECLASSIFIED CELL
                    reclass_all = TRUE)

# Convert class vectors to rasters
r_cv  <- cv.2.rast(r, at$Cell,classVector = cv, plot = FALSE)
r_rc1 <- cv.2.rast(r, at$Cell,classVector = rc1, plot = FALSE)
r_rc2 <- cv.2.rast(r, at$Cell,classVector = rc2, plot = FALSE)

##############################################################################
# PLOTS
##############################################################################
oldpar <- par(mfrow = c(2,2))
m = c(0.1, 3.5, 3.2, 3.5)


# 1)
plot(r_cv, type="classes", axes=FALSE, legend = FALSE, asp = NA, mar=m,
     colNA="#818792", col=c("#1088a0", "#78b2c4"))
text(r)
mtext(side=3, line=2, adj=0, cex=1, font=2, "COND.4.ALL")
mtext(side=3, line=1, adj=0, cex=0.9, "Step1: 'dummy_var > 5', class: 1")
mtext(side=3, line=0, adj=0, cex=0.9, "Step2: 'dummy_var > 3', class: 2")
legend("bottomright", ncol = 1, bg = "white", y.intersp= 1.2,
       legend = c("Class 1", "Class 2", "Unclassified cells"),
      fill = c("#1088a0", "#78b2c4", "#818792"))
```

```
# 2)
plot(r_rc1, type="classes", axes=FALSE, legend = FALSE, asp = NA, mar=m,
     colNA="#818792", col=c("#1088a0", "#78b2c4", "#cfad89"))
text(r)
mtext(side=3, line=2, adj=0, cex=1, font=2, "RECLASS.NBS")
mtext(side=3, line=1, adj=0, cex=0.9, "Reclass: class 2 adjacent to class 1")
mtext(side=3, line=0, adj=0, cex=0.9, "reclass_all = FALSE")
legend("bottomright", ncol = 1, bg = "white", y.intersp= 1.2,
       legend = c("Reclassified cells"), fill = c("#cfad89"))

# 3)
plot(r_rc2, type="classes", axes=FALSE, legend = FALSE, asp = NA, mar=m,
     colNA="#818792", col=c("#1088a0", "#78b2c4", "#cfad89"))
text(r)
mtext(side=3, line=2, adj=0, cex=1, font=2, "RECLASS.NBS")
mtext(side=3, line=1, adj=0, cex=0.9, "Reclass: class 2 adjacent to class 1")
mtext(side=3, line=0, adj=0, cex=0.9, "reclass_all = TRUE")
legend("bottomright", ncol = 1, bg = "white", y.intersp= 1.2,
       legend = c("Reclassified cells"), fill = c("#cfad89"))
par(oldpar)
```

`rel.pi` *Relative position index*

Description

Compute the relative position index of raster objects.

Usage

```
rel.pi(attTbl, RO, el, type = "s", plot = FALSE, r = NULL)
```

Arguments

`attTbl`	data.frame, the attribute table returned by the function `attTbl`.
`RO`	column name, the name of the column with the raster object IDs.
`el`	column name, the name of column with the elevation values on which the relative position index is computed.
`type`	character, defines if position index values are *standardized* (`"s"`) or *normalized* (`"n"`).
`plot`	logic, plot the results.
`r`	single or multi-layer raster of the class `SpatRaster` (see `help("rast", terra)`) used to compute the attribute table. Required only if `plot = TRUE`.

Details

Position index values are computed only for cells that belong to a raster object.

- *Standardized position index values* (`type="s"`) are computed with the formula `( x - mean(x) ) / sd(x)`;
- *Normalized position index values* (`type="n"`) are computed with the formula `( x - min(x) ) / ( max(x) - min(x) )`;
- Variable `x` represents the elevation values of individual raster object.

Value

The function returns a vector with relative position index values. The vector has length equal to the number of rows of the attribute table. NA values are assigned to cells that do not belong to any raster object.

See Also

`attTbl()`, `ngbList()`, `pi.add()`, `pi.sgm()`

Examples

```
# DUMMY DATA
#############################################################################################
# LOAD LIBRARIES
library(scapesClassification)
library(terra)

# LOAD THE DUMMY RASTER
r <- list.files(system.file("extdata", package = "scapesClassification"),
                pattern = "dummy_raster\\.tif", full.names = TRUE)
r <- terra::rast(r)

# COMPUTE THE ATTRIBUTE TABLE
at <- attTbl(r, "dummy_var")

# COMPUTE THE LIST OF NEIGBORHOODS
nbs <- ngbList(r, rNumb=TRUE, attTbl=at) # rnumb MUST be true to use obj.nbs

#############################################################################
# COMPUTE RASTER OBJECTS
#############################################################################
at$RO <- anchor.seed(at, nbs, silent=TRUE, class = NULL, rNumb=TRUE,
                     cond.filter = "dummy_var > 1",
                     cond.seed   = "dummy_var==max(dummy_var)",
                     cond.growth = "dummy_var<dummy_var[]",
                     lag.growth  = 0)

# Convert class vector at$RO to raster and plot
r_RO  <- cv.2.rast(r = r, classVector = at$RO)
terra::plot(r_RO, type="classes", main="Raster objects",
            plg=list(x=1, y=1, cex=0.9))
```

```
##############################################################################
# STANDARDIZED RELATIVE POSITION INDEX
##############################################################################
relPI <- rel.pi(attTbl = at, RO = "RO", el = "dummy_var",
                type = "s",
                plot = TRUE, r = r)

# Annotate relPI
points(terra::xFromCell(r, at$Cell[which(at$RO==1)]),
       terra::yFromCell(r, at$Cell[which(at$RO==1)]) - 0.04,
       pch=20, col="yellow")
points(terra::xFromCell(r, at$Cell[which(at$RO==2)]),
       terra::yFromCell(r, at$Cell[which(at$RO==2)]) - 0.04,
       pch=20, col="darkgreen")
text(xyFromCell(r,at$Cell), as.character(round(relPI,2)))
legend(1.02, 0.4, legend=c("1", "2"), bty = "n", title="RO:", xpd=TRUE,
col=c("#E6E600", "#00A600"), pch=20, cex=0.9, pt.cex = 1.5)

##############################################################################
# NORMALIZED RELATIVE POSITION INDEX
##############################################################################
# Compute normalized relative position index
relPI <- rel.pi(attTbl = at, RO = "RO", el = "dummy_var",
                type = "n",
                plot = TRUE, r = r)

# Annotate relPI
points(terra::xFromCell(r, at$Cell[which(at$RO==1)]),
       terra::yFromCell(r, at$Cell[which(at$RO==1)]) - 0.04,
       pch=20, col="yellow")
points(terra::xFromCell(r, at$Cell[which(at$RO==2)]),
       terra::yFromCell(r, at$Cell[which(at$RO==2)]) - 0.04,
       pch=20, col="darkgreen")
text(xyFromCell(r,at$Cell), as.character(round(relPI,2)))
legend(1.02, 0.4, legend=c("1", "2"), bty = "n", title="RO:", xpd=TRUE,
col=c("#E6E600", "#00A600"), pch=20, cex=0.9, pt.cex = 1.5)
```

Index

anchor.cell, 2
anchor.seed, 5, *29, 30*
anchor.seed(), *25*, *31*
anchor.svo, 9
anchor.svo(), *3*
attTbl, *3*, *5*, 12, *14*, *16*, *18*, *25*, *28*, *32*, *34, 35*, *37*, *39*, *41*, *44*, *46*, *49*, *52*
attTbl(), *3*, *7*, *14*, *16*, *20*, *26*, *31*, *35*, *38, 39*, *42*, *45*, *47*, *50*, *53*

classify.all, 14, *29*
classify.all(), *31*, *50*
cond.4.all, 16, *28, 29*
cond.4.all(), *25*, *31*
cond.4.nofn, 18, *29, 30*
cond.4.nofn(), *16*, *25*, *31*
cond.parse, 24
cond.reclass, 25, *29, 30*
cond.reclass(), *16*, *25*, *31*, *50*
conditions, *3*, *6, 7*, *14*, *16*, *18–20*, *25, 26*, 27, *28*, *42*, *44*, *49, 50*
conditions(), *3*, *7*, *14*, *16*, *20*, *25, 26*, *31*, *42*, *45*
cv.2.rast, 32

ngb8, 33
ngb8(), *35*
ngbList, *5*, *14*, *18*, *25*, *29*, 34, *37*, *39*, *41, 42*, *44*, *46*, *49*
ngbList(), *7*, *14*, *20*, *26*, *34*, *38, 39*, *42*, *45*, *47*, *50*, *53*

obj.border, 37, *39*
obj.border(), *39*
obj.nbs, 39
obj.nbs(), *38*

peak.cell, 41
pi.add, 43
pi.add(), *47*, *53*
pi.sgm, 46
pi.sgm(), *45*, *53*

reclass.nbs, *29*, 49
rel.pi, 52
rel.pi(), *45*, *47*