Package: pixelclasser (via r-universe)

October 29, 2026

Type Package

Title Classifies Image Pixels by Colour

Version 1.0.0

Description Contains functions to classify the pixels of an image file (jpeg or tiff) by its colour. It implements a simple form of the techniques known as Support Vector Machine adapted to this particular problem.

Encoding UTF-8

License GPL-3

LazyData true

RoxygenNote 7.1.1

Imports graphics, grDevices, jpeg, tiff,

Suggests knitr, rmarkdown, testthat, covr

VignetteBuilder knitr

URL https://github.com/ropensci/pixelclasser

BugReports https://github.com/ropensci/pixelclasser

Language en-GB

Config/pak/sysreqs libjpeg-dev libtiff-dev

Repository https://ropensci.r-universe.dev

Date/Publication 2021-02-22 21:16:31 UTC

RemoteUrl https://github.com/ropensci/pixelclasser

RemoteRef HEAD

RemoteSha e2539ca0ddec965e34003c591cd6d77b040aa260

Contents

classify_pixels 2
define_cat 3
define_rule 5

define_subcat 6
label_rule 7
pixelclasser 9
place_rule 9
plot_pixels 11
plot_rgb_plane 12
plot_rule 13
read_image 14
save_classif_image 15

Index **17**

`classify_pixels` *Classifies the pixels of an image*

Description

Classifies the pixels represented in an object of class `transformed_image` using the rules contained in a list of objects of class `pixel_cat`.

Usage

```
classify_pixels(image_prop, ..., unclassed_colour = "black", verbose = TRUE)
```

Arguments

`image_prop`	an array containing the image. It must be an object produced with function `read_image()`.
`...`	a list of objects of class `pixel_cat` containing the classification rules.
`unclassed_colour`	a character string setting the colour to be assigned to unclassified pixels. Defaults to "black".
`verbose`	a logical value. When TRUE (default) the function prints some statistics about the classification.

Details

This function generates a set of incidence matrices indicating whether a pixel belongs to a pixel category or not. An additional matrix identifies the pixels that do not belong to the defined categories, i e unclassed pixels. Depending on how the rules were defined, it can be void or contain pixels, but it is always present and named `unclassified`.

To create the incidence matrices for each category, a matrix for each rule is created and then combined with the matrices of the other using the `and` operator.

When a set of subcategories is used, the procedure is the same for each subcategory and then the matrices of the subcategories are combined again, this time using the `or` operator. See the help for `define_subcat` for more details.

`unclassed_colour` can be specified in any form understood by `grDevices::col2grb`.

Value

Returns an object of class `classified_image`, which is a list containing nested lists. Each first-level element corresponds to one of the pixel categories and its name is the category name. They contains the second-level list, which have the following elements:

- `colour`: a matrix defining a colour to paint the pixels in the classified image. Inherited from the `pixel_class` object defining the class.
- `incid_mat`: a logical matrix where `TRUE` values indicate that the pixel belongs to this pixel category.

See Also

`define_cat`, `col2rgb`.

Examples

```
# The series of steps to classify a image supplied in the package

yellow <- "#ffcd0eff"
blue <- "#5536ffff"

ivy_oak_rgb <- read_image(system.file("extdata", "IvyOak400x300.JPG",
                          package = "pixelclasser"))

rule_01 <- define_rule("rule_01", "g", "b",
                       list(c(0.345, 1/3), c(0.40, 0.10)), comp_op = "<")
rule_02 <- define_rule("rule_02", "g", "b",
                       list(c(0.345, 1/3), c(0.40, 0.10)), comp_op = ">=")

cat_dead_leaves <- define_cat("dead_leaves", blue, rule_01)
cat_living_leaves <- define_cat("living_leaves", yellow, rule_02)

ivy_oak_classified <- classify_pixels(ivy_oak_rgb, cat_dead_leaves,
                        cat_living_leaves)
```

`define_cat` *Creates a category object*

Description

Creates an object of class `pixel_cat`, which contains a list of objects of class `pixel_subcat`.

Usage

```
define_cat(cat_name, cat_colour, ...)
```

Arguments

cat_name	a character string containing the name of the category.
cat_colour	a character string defining the colour to paint the pixels with when creating a classified picture.
...	a list of pixel_subcat objects, or pixel_rule objects in case that subcategories are not needed. A mixed list of pixel_rule and pixel_subcat objects is not allowed.

Details

The function receives a list of objects of class pixel_subcat and creates a list of class pixel_cat with them. However, for cases that does not need subcategories, i e that only need a set of rules,need a single set of rules, these can be passed to the function, which creates an internal subcategory object to contain them. See the examples below.

Note that it is an error to pass a mixture of pixel_rule and pixel_subcat objects.

colour can be specified in any form understood by grDevices::col2grb.

Value

A list of class pixel_cat with the following elements:

- name: a character string containing the name of the pixel category.
- colour: a character string describing the colour of the pixels of the category in the classified images.
- subcats: a list containing the subcategories. Their names are the names of the elements of the list.

See Also

define_rule, define_subcat, col2rgb

Examples

```
# The rules are not consistent, they are only useful as examples
rule01 <- define_rule("R01", "g", "b",
                      list(c(0.35, 0.30), c(0.45, 0.10)), ">=")
rule02 <- define_rule("R02", "g", "b",
                      list(c(0.35, 0.253), c(0.45, 0.253)), ">=")
rule03 <- define_rule("R03", "g", "b",
                      list(c(0.35, 0.29), c(0.49, 0.178)), ">=")
rule04 <- define_rule("R04", "g", "b",
                      list(c(0.35, 0.253), c(0.45, 0.253)), "<")

subcat01 <- define_subcat("Subcat01", rule01, rule02)
subcat02 <- define_subcat("Subcat02", rule03, rule04)

cat01 <- define_cat("Cat01", "#ffae2a", subcat01, subcat02)

# A single category defined by a set of rules, not subcategories
```

```
cat02 <- define_cat("Cat02", "#00ae2a", rule01, rule02, rule03)
```

`define_rule` *Creates a rule object*

Description

Creates an object of class `pixel_rule` from a line in `rgb` space, defined by the user, and a relational operator.

Usage

```
define_rule(rule_name, x_axis, y_axis, rule_points, comp_op)
```

Arguments

`rule_name`	a character string containing the name of the rule.
`x_axis`	a character string selecting the colour variable used as x axis, one of `"r"`, `"g"` or `"b"`.
`y_axis`	a character string selecting the colour variable used as y axis, one of `"r"`, `"g"` or `"b"`.
`rule_points`	either an object of of class `"rule_points"` created with function `place_rule()`, or a list containing the coordinates of two points defining the line.
`comp_op`	a character string containing one of the comparison operators `">"`, `">="`, `"<"`, `"<="`.

Details

This function estimates the slope (`a`) and intercept (`c`) of the line `y = ax + c` using the coordinates of two points on the line. `x` and `y` are two colour variables selected by the user (`r`, `g`, or `b`). The line divides the plane in two subsets and the comparison operator selects the subset that contains the points (pixels) of interest.

When a list of two points is passed in `rule_points`, it is internally converted into an an object of class `rule_points`.

The pair of points used to define the line are not constrained to belong to the area occupied by the pixels, but they are used by `plot_rule()` as the start and end of the plotted line. Therefore, the extremes of the line can be selected in the most convenient way, provided that the line divides correctly the categories. Convenience means that the line should seem nice in the plot, if this matters.

Because the variables were transformed into proportions, the pixel are always inside the triangle defined by the points `(0, 0)`, `(1, 0)`, `(0, 1)`. So, the sides of this triangle can be considered as implicit rules which do not need to be created. In this way, a single line creates two polygons by cutting the triangle in two. The implicit rules can reduce the number of rules to create in most cases.

Value

A list of class `pixel_rule` containing the following elements:

- `rule_name`: a character string containing the rule name.
- `rule_text`: a character string containing the mathematical expression of the rule.
- `comp_op`: a character string containing the comparison operator used in the rule.
- `a`: a numerical vector containing the parameter `a` (slope) of the line.
- `c`: a numerical vector containing the parameter `c` (intercept) of the line.
- `x_axis`: a character string containing the colour variable selected as `x` axis.
- `y_axis`: a character string containing the colour variable selected as `y` axis.
- `first_point`: a numerical vector containing the coordinates of the first point used to estimate the line equation.
- `second_point`: a numerical vector containing the coordinates of the second point.

See Also

`define_subcat`, `define_cat`, `plot_rule`, `plot_rgb_plane`

Examples

```
# Creating the line by passing the coordinates of two points on the line:
rule01 <- define_rule("rule01", "g", "b",
                      list(c(0.35, 0.30), c(0.45, 0.10)),">")

# A vertical line as a rule; note that the equation is simplified
rule02 <- define_rule("rule02", "g", "b",
                      list(c(0.35, 0.30), c(0.35, 0.00)), ">")
## Not run:
# Creating the rule by passing an object of type rule_point:
rule_points01 <- place_rule("g", "b")
rule03 <- define_rule("rule03", "g", "b", rule_points01,">")

# Note that the creation of the intermediate object can be avoided:
rule04 <- define_rule("rule04", "g", "b", place_rule("g", "b"),">")

## End(Not run)
```

---

define_subcat *Creates a subcategory object*

Description

Creates an object of class `pixel_subcat` from a list of objects of class `pixel_rule`.

Usage

```
define_subcat(subcat_name, ...)
```

Arguments

subcat_name	a character string containing the name of the subcategory.
...	a list of objects of class pixel_rule.

Details

When the shape of the cluster of pixels belonging to one category is not convex, the rules become inconsistent (the lines cross in awkward ways) and the classification produced is erroneous. To solve this problem, the complete set of rules is divided into several subsets (subcategories) that break the original non-convex shape into a set of convex polygons. Note that any polygon can be divided in a number of triangles, so this problem always has solution. However, in many cases (such as the one presented in the pixelclasser vignette) a complete triangulation is not needed.

Internally, classify_pixels() classifies the points belonging to each subcategory and then joins the incidence matrices using the or operator, to create the matrix for the whole category.

Value

An object of class pixel_subcat, which is a list with the following elements:

- name a character string containing the name of the subcategory.
- rules_list a list of pixel_rule objects.

See Also

define_rule, define_cat

Examples

```
rule01 <- define_rule("R01", "g", "b",
                      list(c(0.35, 0.30), c(0.45, 0.10)), ">=")
rule02 <- define_rule("R02", "g", "b",
                      list(c(0.35, 0.253), c(0.45, 0.253)), ">=")

subcat01 <- define_subcat("Subcat_01", rule01, rule02)
```

---

label_rule *Adds a label to the rule*

Description

This function adds a label to the line that represents a rule on a plot created by plot_rgb_plane().

Usage

```
label_rule(rule, label = "", shift = c(0, 0), ...)
```

Arguments

`rule`	an object of class `pixel_rule`.
`label`	a string to label the line. It is attached at the coordinates of the start (first point) of the line.
`shift`	a numeric vector to set the displacement of the label from the start of the line. Expressed in graph units, defaults to c(0, 0).
`...`	additional graphical parameters passed to the underlying `graphics::text()` function.

Details

The function uses the information stored in the pixel_rule object to plot the label at the start of the line. The `shift` values, expressed in plot coordinates, are added to the coordinates of that point to place the label elsewhere. Note that ... can be used to pass values for the `adj` parameter to the underlying `graphics::text()` function, which also modifies the position of the label.

Use a character string understood by `grDevices::col2rgb()` to set the colour of the label.

Value

The function does not return any value.

See Also

plot_rgb_plane, define_rule, col2rgb, text

Examples

```
rule_01 <- define_rule("rule_01", "g", "b",
                       list(c(0.1, 0.8), c(0.40, 0.10)), "<")
plot_rgb_plane("g", "b")

# The rule is represented as a green line
plot_rule(rule_01, col = "green")

# And the label is added in three different positions by passing col and adj
# to the underlying function
label_rule(rule_01, label = expression('R'[1]*''), shift = c(0,0),
           col = 'black', adj = 1.5)
label_rule(rule_01, label = expression('R'[1]*''), shift = c(0.2, -0.4),
           col = 'blue', adj = 0)
label_rule(rule_01, label = expression('R'[1]*''), shift = c(0.3, -0.7),
           col = 'black', adj = -0.5)
```

pixelclasser *pixelclasser: Functions to classify pixels by colour*

Description

`pixelclasser` contains functions to classify the pixels of an image file (in format jpeg or tiff) by its colour. It uses a simple form of the technique known as Support Vector Machine, adapted to this particular problem. The original colour variables (`R, G, B`) are transformed into colour proportions (`r, g, b`), and the resulting two dimensional plane, defined by any convenient pair of the transformed variables is divided in several subsets (categories) by one or more straight lines (rules) selected by the user. Finally, the pixels belonging to each category are identified using the rules, and a classified image can be created and saved.

Details

To classify the pixels of an image, a series of steps must be done in the following order, using the functions shown in parenthesis:

- import the image into an R array of transformed (`rgb`) data (`read_image()`).
- plot the pixels of the image on the plane of two transformed variables that shows the categories of pixels most clearly (`plot_rgb_plane()`, `plot_pixels`).
- trace lines between the pixel clusters and use them to create classification rules (`place_rule()`, `define_rule`, `plot_rule()`).
- combine the rules to define categories. Sometimes the rules are combined into subcategories and these into categories (`define_cat()`, `define_subcat()`).
- use the categories to classify the pixels (`classify_pixels()`).
- save the results of the classification as an image, if needed (`save_clasif_image()`).

These steps are explained in depth in the vignette included in the package.

Author(s)

Carlos Real (carlos.real@usc.es)

place_rule *Places a line on the rgb plot*

Description

A wrapper function for `graphics::locator` that makes the creation of rules easier.

Usage

```
place_rule(x_axis, y_axis, line_type = "f")
```

Arguments

x_axis	a character string indicating the colour variable that corresponds to the x axis, one of "r", "g" or "b".
y_axis	a character string indicating the colour variable that corresponds to the y axis, one of "r", "g" or "b".
line_type	a character string indicating that the line is vertical "v", horizontal "h" or free ("f", the default).

Details

This function calls graphics::locator allowing to select two points, plots the line joining these points and returns a list containing their coordinates. The coordinates are rearranged to pass them to define_rule().

True horizontal and vertical lines are difficult to create by hand. In these cases, specifying "vertical" or "horizontal" (partial match allowed, i e "h") will copy the appropriate coordinate value from the first point to the second. Note that this is done after locator() returns, so the plot will show the line joining the original points, not the corrected ones. Use plot_rule() to see corrected line.

Value

A list of class rule_points containing the following elements:

- x_axis: a character string containing the colour variable selected as x axis.
- y_axis: a character string containing the colour variable selected as y axis.
- first_point: coordinates of the start point of the line.
- second_point: coordinates of the end point of the line.

See Also

locator, define_rule, plot_rule, plot_rgb_plane

Examples

```
## Not run:
plot_rgb_plane("r", "g")
line01 <- place_rule("r", "g")          # A "free" line
line02 <- place_rule("r", "g", "h")     # A horizontal line

## End(Not run)
```

plot_pixels *Plot the pixels of a transformed image*

Description

This function is a wrapper for function `points()` in package `graphics` for plotting the pixels of a transformed rgb image on the triangular diagram previously created by `plot_rgb_plane()`.

Usage

```
plot_pixels(image_rgb, x_axis, y_axis, ...)
```

Arguments

`image_rgb`	an object produced by `read_image()`.
`x_axis`	a character string indicating which colour variable use as x.
`y_axis`	a character string indicating which colour variable use as y.
`...`	additional graphical parameters, mainly to set the colour (`col`) of the points.

Details

It is advantageous to specify a colour such as `"#00000005"` which is black but almost transparent. In this way a kind of density plot is created because the clustering of points creates areas of darker colour. Note that a colour without specific transparency information defaults to an opaque colour, so `"#000000"` is the same as `"#000000ff"`. The colours can be specified in any form understandable by `grDevices::col2rgb`, but the hexadecimal string allows setting the colour transparency and it is the preferred style. Note also that the points are plotted using pch = ".", as any other symbol would clutter the graph.

Warning: plotting several million points in an R graph is a slow process. Be patient or reduce the size of the images as much as possible. Having a nice smartphone with a petapixel camera sensor is good for artistic purposes, but not always for efficient scientific work.

Value

The function does not return any value.

See Also

`plot_rgb_plane`, `col2rgb`

Examples

```
# Plotting the pixels of the example image included in this package
ivy_oak_rgb <- read_image(system.file("extdata", "IvyOak400x300.JPG",
                                      package = "pixelclasser"))
plot_rgb_plane("g", "b")
plot_pixels(ivy_oak_rgb, "g", "b", col = "#00000005")
```

`plot_rgb_plane` *Plots a triangular plot to be filled with pixels and rules*

Description

Plots a plane of the two variables selected by the user (`r`, `g` or `b`) and, to serve as visual references, the lines limiting the triangular area that can contain pixels (in blue) and the areas where one of the colour variables has the larger proportion values (in red). Points representing the pixels of a transformed image and lines representing the rules can be later added to the plot using functions `plot_pixels()` and `plot_rule()`.

Usage

```
plot_rgb_plane(
  x_axis,
  y_axis,
  plot_limits = TRUE,
  plot_guides = TRUE,
  plot_grid = TRUE,
  ...
)
```

Arguments

`x_axis`	a character string indicating which colour variable use as x.
`y_axis`	a character string indicating which colour variable use as y.
`plot_limits`	a logical value. When TRUE (default) the limits of the area where the pixels can be found are plotted.
`plot_guides`	a logical value. When TRUE (default) the limits of the area where one variable dominates are plotted.
`plot_grid`	a logical value; if TRUE (default) a grid is added.
`...`	allows passing graphical parameters to the plotting functions.

Details

Graphical parameters can be passed to the function to modify the appearance of the plot. Intended for passing `xlim` and `ylim` values to plot the part of the graph where the points are concentrated.

Because the variables were transformed into proportions, the pixel are always inside the triangle defined by the points `(0, 0)`, `(1, 0)`, `(0, 1)`. This triangle is plotted in blue. The point where all three variables have the same value is `(1/3, 1/3)`. The lines joining this point with the centers of the triangle sides divide the areas where one of the three variables has higher proportions than the other two. These lines are plotted as visual aids, so they can be deleted at will.

Value

The function does not return any value.

See Also

plot_pixels, plot_rule, define_rule

Examples

```
# Simplest call
plot_rgb_plane("g", "b")

# Plane without the red lines
plot_rgb_plane("g", "b", plot_guides = FALSE)

# Restricting the plane area to show
plot_rgb_plane("g", "b", xlim = c(0.2, 0.5), ylim = c(0.0, 0.33))
```

---

plot_rule *Plots the line that defines a rule*

Description

This function draws the line that defines a rule on the plot created by `plot_rgb_plane()`.

Usage

```
plot_rule(rule, label = "", ...)
```

Arguments

`rule`	an object of class `pixel_rule` produced by `define_rule()`.
`label`	a string to label the line. It is attached at the coordinates of the second point used to define the line.
`...`	additional graphical parameters passed to the underlying `lines()` function, for example to define the line colour or dashing style. They are also used for the line label.

Details

The function uses the information stored in the `pixel_rule` `object` to plot the line.

Use the ... to set the colour and other characteristics of the line. Use any character string understood by `col2rgb()`.

Labels can be added to the rule using `label_rule()`.

Value

The function does not return any value.

See Also

plot_rgb_plane, define_rule, label_rule col2rgb

Examples

```
rule_01 <- define_rule("rule_01", "g", "b",
                       list(c(0.345, 1/3), c(0.40, 0.10)), "<")

plot_rgb_plane("g", "b")
plot_rule(rule_01, col = "green")
```

---

read_image *Imports a jpg or tiff file.*

Description

Imports an image file (in JPEG or TIFF format) into an array, and converts the original R, G and B values in the file into proportions (r, g and b variables).

Usage

```
read_image(file_name)
```

Arguments

file_name	A character string containing the name of the image file.

Details

This function calls the functions readJPEG() and readTIFF() in packages jpeg and tiff to import the data into an R array. Then it transforms the data into proportions

Value

Returns an array of dimensions r x c x 3 and class transformed_image, being r and c the number of rows and columns in the image. The last dimension corresponds to the R, G and B variables (= bands) that define the colours of the pixels. The values in the array are the proportions of each colour (r, g, b), i.e. r = R / (R + G + B), and so on.

See Also

For more information about jpeg and tiff file formats, see the help pages of readJPEG and readTIFF functions in packages jpeg and tiff, respectively.

Examples

```
# An example that loads the example file included in the package
ivy_oak_rgb <- read_image(system.file("extdata", "IvyOak400x300.JPG",
                                      package = "pixelclasser"))
```

`save_classif_image` *Saves a classified image in TIFF or JPEG format*

Description

Creates an image file in TIFF or JPEG format from an array of class `classified_image`.

Usage

```
save_classif_image(classified_image, file_name, ...)
```

Arguments

`classified_image`	an object of class `classified_image`.
`file_name`	a character string with the name of the output file, including the extension.
`...`	further parameters to pass to functions `writeJPG` and `writeTIFF`. If void, the default values of these functions are used.

Details

The type of the output file (jpeg or tiff) is selected from the extension included in the file name. It must be one of (`"jpg"`, `"JPG"`, `"jpeg"`, `"JPEG"`, `"tif"`, `"TIF"`, `"tiff"`, `"TIFF"`).

Note that the default value for jpg quality is 0.7. For maximal quality set `quality = 1` using the ... argument. Such adjustments are not needed with `tiff` files, as this is a lossless format.

Value

It does not return anything, only creates the file.

See Also

`classify_pixels`

For more information about the options for file formatting see see the help pages of `readJPEG` and `readTIFF` functions in packages `jpeg` and `tiff`, respectively.

Examples

```
## Not run:

# Saving an hypothetical image. Note the use of quality to set the
# maximum quality level in the JPEG file
save_classif_image(image01_class, "./myimages/image01_classified.jpg",
                   quality = 1)

## End(Not run)
```

Index

classify_pixels, 2, *15*
col2rgb, *3, 4*, *8*, *11*, *14*

define_cat, 3, *3*, *6, 7*
define_rule, *4*, 5, *7, 8*, *10*, *13, 14*
define_subcat, *4*, 6, *6*

label_rule, 7, *14*
locator, *10*

pixelclasser, 9
place_rule, 9
plot_pixels, 11, *13*
plot_rgb_plane, *6*, *8*, *10, 11*, 12, *14*
plot_rule, *6*, *10*, 13, *13*

read_image, 14
readJPEG, *14, 15*
readTIFF, *14, 15*

save_classif_image, 15

text, *8*