Package 'pixelclasser'

May 9, 2026

Type Package

Title Classifies Image Pixels by Colour

Version 1.1.1

Date 2023-10-18

Description
Contains functions to classify the pixels of an image file 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 | file LICENSE

LazyData false

RoxygenNote 7.2.3

Imports graphics, grDevices, jpeg, tiff,

Suggests knitr, rmarkdown, testthat, covr

VignetteBuilder knitr, rmarkdown

Contact carlos.real@usc.es

Language en-GB

NeedsCompilation no

Author Carlos Real [aut, cre] (ORCID: <https://orcid.org/0000-0002-5433-6728>), Quentin Read [rev]

Maintainer Carlos Real <carlos.real@usc.es>

Repository CRAN

Date/Publication 2023-10-18 14:50:05 UTC

Contents

`classify_pixels` *Classifies the pixels of an image*

Description

Classifies the pixels represented in an object of class `"pixel_transformed_image"` using the rules contained in a list of objects of class `"pixel_category"`.

Usage

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

is.classified_image(x)

## S3 method for class 'pixel_classified_image'
summary(object, ...)
```

Arguments

`image_prop`	an object of class `"pixel_transformed_image"` created by function `read_image()`.
`...`	in `classify_pixels()` a list of objects of class `"pixel_category"`; in `summary()`, additional parameters (none needed by now).
`unclassed_colour`	a character string defining the colour of the unclassified pixels. Defaults to `"black"`.
`verbose`	a logical value. When `TRUE` (default) the function prints some statistics about the classification.
`x`	the R object being tested.
`object`	an object of class `"pixel_classified_image"`.

Details

This function uses the rules contained in "pixel_category" objects to create a set of incidence matrices indicating whether a pixel belongs to a pixel category or not.

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. An additional matrix identifies the pixels that do not belong to the defined categories, i.e. unclassed pixels. It is always present and named unclassified even in the case that all pixels were assigned to some class.

When a category is divided in a set of subcategories the pixels are classfied in subcategories as explained in the previous paragraph, and then the matrices of the subcategories are combined again, this time using the or operator. See "pixel_subcategory" for more details.

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

Value

Returns an object of class "pixel_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

pixel_category, col2rgb.

Examples

```
# These are the steps to classify an example image supplied in the package

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

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

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

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

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

deprecated *Deprecated functions*

Description

Deprecated functions

Usage

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

define_subcat(subcat_name, ...)

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

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

These functions were constructors for pixelclasser objects, but now they are substituted by constructors with the same name as the class, as is customary in R.

See Also

pixel_category, pixel_subcategory and pixel_rule.

label_rule *Adds a label to the rule*

Description

This function adds a label to a line representing a rule on the plots 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 `"pixel_rule"` objects to label the line at its start point. 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 additional parameters (`pos` and `adj`) to the underlying `graphics::text()` function, to further adjust the position of the label. Pass a character string understood by `grDevices::col2rgb()` to set the colour of the label (`col`).

Value

The function does not return any value.

See Also

plot_rgb_plane, pixel_rule, col2rgb, text

Examples

```
rule_01 <- pixel_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 a digital 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) manually 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 (using `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()`, `pixel_rule`, `plot_rule()`). The user places the lines manually.
- combine the rules to define categories. Sometimes the rules are combined into subcategories and these into categories (`pixel_category()`, `pixel_subcategory()`).
- 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)

pixel_category *Creates a category object*

Description

Creates an object of class "pixel_category", which contains a list of objects of class "pixel_subcategory".

Usage

```
pixel_category(cat_name, cat_colour, ...)

is.category(x)

## S3 method for class 'pixel_category'
summary(object, ...)
```

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.
...	in pixel_category() a list of objects of class "pixel_subcategory" or "pixel_rule"; in summary(), additional parameters (none needed by now).
x	the R object being tested
object	an object of class "pixel_category".

Details

The function receives a list of objects of class "pixel_subcategory" and creates an object of class "pixel_category" with them. However, subcategories are not always needed (see pixel_subcategory). In these cases "pixel_rule" objects can be passed to this function, which creates an internal subcategory object (named "S0") to contain them. See the examples below.

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

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

Value

An object of class "pixel_category" which is a list with the following elements:

- name: the character string with 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 of "pixel_subcategory" objects.

See Also

pixel_rule, pixel_subcategory, col2rgb

Examples

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

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

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

# A single category defined by a set of rules, not subcategories
cat02 <- pixel_category("Cat02", "#00ae2a", rule01, rule02, rule03)
```

---

pixel_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

```
pixel_rule(rule_name, x_axis, y_axis, line_points, comp_op)

is.rule(x)

## S3 method for class 'pixel_rule'
summary(object, ...)
```

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".

line_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. The coordinates are two-element numeric vectors.
comp_op	a character string containing one of the comparison operators ">", ">=", "<", "<=".
x	the R object being tested
object	an object of class "pixel_category".
...	additional parameters to pass to the function.

Details

This function estimates the slope (a) and intercept (c) of the line y = ax + c from 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 the line is defined by a list containing a couple of points, it is first converted into an object of class "pixel_rule_points" (see place_rule, and the examples below).

The lines are mathematical objects that extend without bound, i.e. all along the x axis. The pair of points do not set the line limits, they only allow the estimation of the line parameters. Therefore, they are not constrained to be inside the triangular area occupied by the pixels, and the points can be selected in the most convenient way, provided that the line divides correctly the categories. Convenience in this context means that the line should seem nice in the plot, if this matters.

Because the variables were transformed into proportions, the pixels in the plot 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 explicitly. In this way, a single line creates two polygons by cutting the triangle in two. Usually, the implicit rules reduce the number of rules to create.

Value

An object of class "pixel_rule" containing these 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 (one of "r", "g" or "b").
- 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

pixel_subcategory, pixel_category, plot_rule, plot_rgb_plane

Examples

```
# Creating the line by passing the coordinates of two points on the line:
rule01 <- pixel_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 will be simplified
rule02 <- pixel_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 <- pixel_rule("rule03", "g", "b", rule_points01,">")

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

## End(Not run)
```

---

pixel_subcategory *Creates a subcategory object*

Description

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

Usage

```
pixel_subcategory(subcat_name, ...)

is.subcategory(x)

## S3 method for class 'pixel_subcategory'
summary(object, ...)
```

Arguments

subcat_name	a character string containing the name of the subcategory.
...	in `pixel_subcategory()` a list of objects of class "`pixel_rule`"; in `summary()`, additional parameters (none needed by now).
x	the R object being tested
object	an object of class "`pixel_subcategory`".

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_subcategory"`, which is a list with these elements:

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

See Also

`pixel_rule`, `pixel_category`

Examples

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

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

---

place_rule *Places a line on the rgb plot*

Description

A wrapper function for `graphics::locator` that helps in creating rules.

Usage

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

is.rule_points(x)
```

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).
x	the R object being tested

Details

This function calls graphics::locator on a previously plotted rgb plane to select two points with the mouse. Then it plots the line joining them and returns an object of class "pixel_rule_object". These objects are passed as parameters to pixel_rule() to create "pixel_rule" objects.

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 to make them the same. 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 the corrected line.

Value

An object of class "pixel_rule_points" containing these elements:

- x_axis: a character string containing the colour variable selected as x axis (one of "r, "g or "b").
- y_axis: a character string containing the colour variable selected as y axis.

See Also

locator, pixel_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 `graphics::points()` for plotting the pixels of an object of class `"pixel_transformed_image"` on an rgb plot.

Usage

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

Arguments

`image_rgb`	an object of class `"pixel_transformed_image"` produced by `read_image()`.
`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 x axis.
`...`	additional graphical parameters to be passed to `graphics::points()`, 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. 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. 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 the plane of the two variables selected by the user, one of (`r`, `g` or `b`), and lines identifying the triangular area that can contain pixels. 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, one of `"r"`, `"g"` or `"b"`,
`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.
`...`	additional graphical parameters passed to the underlying plotting functions.

Details

Because the variables were transformed into proportions, the pixels 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 (red) lines joining this point with the centres of the triangle sides divide the plot in areas where one of the three variables has higher proportions

than the other two. Also, a standard grid can be added. All these lines are visual aids, so, if desired, they can be eliminated using the parameters of the function.

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

Value

The function does not return any value.

See Also

`plot_pixels`, `plot_rule`, `pixel_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 showed in the graph
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 `pixel_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 graphic functions, for example to define the line colour or dashing style.

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. For colours use any character string understood by col2rgb().

A label can be added to the line using label_rule().

Value

The function does not return any value.

See Also

plot_rgb_plane, pixel_rule, label_rule col2rgb

Examples

```
rule_01 <- pixel_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 and transforms 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)

is.transformed_image(x)

## S3 method for class 'pixel_transformed_image'
summary(object, ...)
```

Arguments

file_name	A character string containing the name of the image file.
x	the R object being tested
object	an object of class "pixel_transformed_image".
...	other parameters passed to the function.

Details

This function calls the functions `jpeg::readJPEG()` or `tiff::readTIFF()` to import the image into an R array. Then it transforms the data into proportions

Value

Returns an object of class `"pixel_transformed_image"`, which is an array of dimensions `r x c x 3`, being `r` and `c` the number of rows and columns in the image. The last dimension corresponds to the original `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.

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 JPEG or TIFF format from an object of class `"pixel_classified_image"`.

Usage

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

Arguments

`classified_image`	an object of class `"pixel_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 not used, 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