Package 'imageRy'

June 25, 2025

Type Package

Title Modify and Share Images

Version 0.1.1

Description Tools for manipulating, visualizing, and exporting raster images in R.
Designed as an educational resource for students learning the basics of remote sensing,
the package provides user-friendly functions to apply color ramps, export RGB composites,
and create multi-frame visualizations. Built on top of the 'terra' and 'ggplot2' packages.
See <https://github.com/ducciorocchini/imageRy> for more details and examples.

Encoding UTF-8

Language en-GB

License GPL (>= 2)

Depends R (>= 4.0.0)

Imports terra, viridis, grDevices, graphics, stats, ggplot2, methods,
rlang

Suggests knitr, testthat (>= 3.0.0), ggridges, devtools

Config/testthat/edition 3

NeedsCompilation no

RoxygenNote 7.3.2

Author Duccio Rocchini [aut],
Ludovico Chieffallo [aut, cre],
Giovanni Andrea Nocera [ctb],
Giacomo Panza [ctb],
Michele Torresani [aut],
Elisa Thouverai [aut]

Maintainer Ludovico Chieffallo <ludovico.chieffallo2@unibo.it>

Repository CRAN

Date/Publication 2025-06-25 16:00:02 UTC

Contents

`im.classify` *Classify a Raster Image Using K-Means Clustering*

Description

This function performs unsupervised classification on a raster image using k-means clustering. It assigns each pixel to a cluster and optionally visualizes the classified image.

Usage

```
im.classify(
  input_image,
  num_clusters = 3,
  seed = NULL,
  do_plot = TRUE,
  custom_colors = NULL,
  num_colors = 100
)
```

Arguments

`input_image`	A 'SpatRaster' object representing the input raster image.
`num_clusters`	An integer specifying the number of clusters (default: 3).
`seed`	An optional integer seed for reproducibility of k-means clustering results (default: NULL).
`do_plot`	A logical value indicating whether to display the classified raster (default: TRUE).
`custom_colors`	A vector of custom colors to be used for classification visualization (default: NULL). If NULL, a predefined set of colors is used.
`num_colors`	The number of colors to interpolate in the visualization palette (default: 100).

Details

The function applies k-means clustering on the pixel values of the raster image. Each pixel is treated as a multi-dimensional point, where each band represents a feature. The classified raster assigns each pixel to a cluster, which can be visualized using a color palette.

- If ‘custom_colors‘ is provided, it is used as the classification color palette. - If ‘seed‘ is provided, it ensures reproducibility of k-means clustering. - If ‘do_plot = TRUE‘, the classified raster is displayed with the chosen color scheme.

Value

A ‘SpatRaster‘ object with cluster assignments, where each pixel belongs to a classified cluster.

References

K-means clustering is a widely used unsupervised classification algorithm. For more information, see: https://en.wikipedia.org/wiki/K-means_clustering

See Also

[im.import()], [im.ridgeline()]

Examples

```
library(terra)

# Load a raster dataset
r <- rast(system.file("ex/elev.tif", package = "terra"))

# Perform k-means classification with 4 clusters
classified_raster <- im.classify(r, num_clusters = 4, seed = 123, do_plot = TRUE)
```

---

im.dvi *Compute the Difference Vegetation Index (DVI)*

Description

This function calculates the Difference Vegetation Index (DVI) from a multispectral raster image. The DVI is computed as the difference between the Near-Infrared (NIR) and Red bands.

Usage

```
im.dvi(x, nir, red)
```

Arguments

x	A 'SpatRaster' object representing the input multispectral image.
nir	An integer specifying the band index of the Near-Infrared (NIR) channel.
red	An integer specifying the band index of the Red channel.

Details

The Difference Vegetation Index (DVI) is a simple vegetation index used to assess plant health. It is calculated as:

$$DVI = NIR - Red$$

Higher values indicate denser and healthier vegetation.

Value

A 'SpatRaster' object containing the computed DVI values.

References

For more details on the DVI index, see: https://en.wikipedia.org/wiki/Vegetation_Index

See Also

[im.classify()], [im.ridgeline()]

Examples

```
library(terra)

# Load a multispectral raster image with 3 bands
r <- rast(system.file("ex/logo.tif", package = "terra"))

# Compute DVI using band 2 (proxy for NIR) and band 1 (Red)
dvi_raster <- im.dvi(r, nir = 2, red = 1)

plot(dvi_raster)
```

---

im.export *Export a Raster to GeoTIFF, PNG, or JPG*

Description

This function saves a 'SpatRaster' object to disk in **GeoTIFF**, **PNG**, or **JPG** format.

Usage

```
im.export(x, filename, overwrite = TRUE)
```

Arguments

x	A 'SpatRaster' object representing the raster to be saved.
filename	A character string specifying the output file path with '.tif', '.png', or '.jpg' extension.
overwrite	A logical value indicating whether to overwrite an existing file (default: TRUE).

Details

- **GeoTIFF ('.tif')**: Uses 'terra::writeRaster()', preserving geospatial information. - **PNG/JPG ('.png', '.jpg', '.jpeg')**: Converts the raster to an image and saves it with 'png()' or 'jpeg()'. - **If the raster has multiple bands**, only the first band is saved in PNG/JPG format.

Value

No return value. The function writes the raster to disk.

See Also

[im.import()], [writeRaster()]

Examples

```
library(terra)

# Create a sample raster
r <- rast(nrows = 10, ncols = 10)
values(r) <- runif(ncell(r))

# Export as GeoTIFF to temporary file
tif_path <- tempfile(fileext = ".tif")
im.export(r, tif_path)

# Export as PNG to temporary file
png_path <- tempfile(fileext = ".png")
im.export(r, png_path)
```

---

im.ggplot *Visualize a Raster Image Using ggplot2*

Description

This function converts a 'SpatRaster' object into a 'ggplot2' visualization, allowing for flexible raster plotting with color interpolation.

Usage

```
im.ggplot(input_raster, layerfill = 1)
```

Arguments

input_raster	A 'SpatRaster' object representing the input raster image.
layerfill	An integer indicating the layer index to be used for coloring the raster (default: 1).

Details

This function extracts raster values, converts them into a data frame, and uses 'ggplot2' to visualize the raster with a viridis color scale.

- If 'layerfill' is not provided, the function defaults to using the first layer. - The function automatically handles coordinate extraction ('x' and 'y' values). - Colors are applied using 'scale_fill_viridis()', ensuring good perceptual readability.

Value

A 'ggplot' object displaying the raster image.

See Also

[im.classify()], [im.dvi()]

Examples

```
library(terra)
library(ggplot2)

# Create a sample raster
r <- rast(nrows = 10, ncols = 10)
values(r) <- matrix(runif(100), nrow = 10)

# Generate a ggplot visualization
im.ggplot(r)
```

---

im.import *Import a raster image*

Description

This function imports a raster image from the package's internal image collection, from a user-specified local path, or from a remote Zenodo repository if the image is not found locally.

Usage

```
im.import(im)
```

Arguments

im	A character string. Either the name of an image included in the package, or a full file path to a user-provided raster image.

Value

A ‘SpatRaster‘ object.

`im.list` *List Available Raster Images in the imageRy Package*

Description

This function lists all raster images stored in the ‘imageRy‘ package.

Usage

```
im.list()
```

Details

The function retrieves the names of all files in the "images" directory of the ‘imageRy‘ package. These files can be imported using ‘im.import()‘.

Value

A character vector containing the names of available raster image files.

See Also

[im.import()]

Examples

```
library(imageRy)

# List available images
im.list()
```

`im.multiframe` *Set Up a Multi-Frame Plot Layout*

Description

This function sets up a multi-frame plotting layout using ‘par(mfrow = c(x, y))‘, allowing multiple plots to be displayed in a grid format.

Usage

```
im.multiframe(x, y)
```

Arguments

`x`	An integer specifying the number of rows in the plot layout.
`y`	An integer specifying the number of columns in the plot layout.

Details

This function changes the ‘mfrow‘ graphical parameter using ‘par()‘, enabling multiple plots to be displayed in a grid layout within the same plotting window. The original plotting parameters are automatically restored when the function exits.

Value

No return value. This function modifies the graphical parameters temporarily.

See Also

[im.ggplot()], [im.import()]

Examples

```
# Set up a 2x2 plotting layout
im.multiframe(2, 2)

# Example plots
plot(1:10, rnorm(10))
plot(1:10, runif(10))
plot(1:10, rpois(10, lambda = 5))
plot(1:10, rbeta(10, shape1 = 2, shape2 = 5))

# Layout is automatically restored after im.multiframe() exits
```

`im.ndvi` *Compute the Normalized Difference Vegetation Index (NDVI)*

Description

This function calculates the Normalized Difference Vegetation Index (NDVI) from a multispectral raster image. NDVI is a widely used vegetation index that assesses plant health by comparing Near-Infrared (NIR) and Red bands.

Usage

```
im.ndvi(x, nir, red)
```

Arguments

`x`	A 'SpatRaster' object representing the input multispectral image.
`nir`	An integer specifying the band index of the Near-Infrared (NIR) channel.
`red`	An integer specifying the band index of the Red channel.

Details

NDVI is calculated as:

$$NDVI = (NIR - Red)/(NIR + Red)$$

where: - **High NDVI values (~1)** indicate healthy, dense vegetation. - **Low NDVI values (~0 or negative)** indicate barren land, water bodies, or unhealthy vegetation.

Important: - Ensure that 'nir' and 'red' correspond to the correct band indices in your raster image. - Pixels with (NIR + Red) = 0 will result in 'NaN' values.

Value

A 'SpatRaster' object containing the computed NDVI values, ranging from -1 to 1.

References

For more details on NDVI, see: https://en.wikipedia.org/wiki/Normalized_difference_vegetation_index

See Also

[im.dvi()], [im.classify()]

Examples

```
library(terra)

# Create a dummy 3-band raster (e.g., NIR = band 3, Red = band 2)
r <- rast(nrows = 10, ncols = 10, nlyrs = 3)
values(r) <- runif(ncell(r) * 3)

# Compute NDVI using bands 3 (NIR) and 2 (Red)
ndvi_raster <- im.ndvi(r, nir = 3, red = 2)

# Plot the result
plot(ndvi_raster)
```

---

im.pca *Perform Principal Component Analysis (PCA) on a Raster Image*

Description

This function applies Principal Component Analysis (PCA) to a multispectral raster image, extracting all available principal components. It reduces dimensionality while preserving the most important variance in the dataset.

Usage

```
im.pca(input_image, n_samples = 100, n_components = 3)
```

Arguments

input_image	A 'SpatRaster' object representing the input multispectral image.
n_samples	An integer specifying the number of random samples used for PCA computation (default: 100).
n_components	Number of principal components to compute.

Details

Principal Component Analysis (PCA) is a statistical technique used to transform correlated raster bands into a set of orthogonal components, capturing the most variance in fewer bands.

- The function **automatically determines** the number of components based on the number of bands. - A sample of 'n_samples' pixels is used to compute the PCA transformation. - The **full image** is then projected onto the principal component space. - The resulting raster contains **all computed principal components**. - The output is visualized using a 'viridis' color scale.

Value

A 'SpatRaster' object containing all computed principal components.

See Also

[im.import()], [im.ggplot()]

Examples

```
library(terra)
library(viridis)

# Create a 3-band raster
r <- rast(nrows = 10, ncols = 10, nlyrs = 3)
values(r) <- runif(ncell(r) * 3)

# Perform PCA
pca_result <- im.pca(r, n_samples = 100)

# Plot the first principal component
plot(pca_result[[1]])
```

---

im.plotRGB *Plot a Raster Image as an RGB Composite with User-Selected Bands*

Description

This function visualizes a multispectral raster image using user-defined bands for the Red, Green, and Blue channels. A linear contrast stretch is applied to enhance visualization.

Usage

```
im.plotRGB(x, r, g, b, title = "")
```

Arguments

x	A ‘SpatRaster‘ object representing the input multispectral image.
r	An integer specifying the band index for the Red channel.
g	An integer specifying the band index for the Green channel.
b	An integer specifying the band index for the Blue channel.
title	A character string specifying the plot title (default: "").

Details

- The function allows users to **manually select bands** for RGB visualization. - It applies ‘stretch="lin"‘ in ‘plotRGB()‘ to enhance contrast. - Axis and label colors are set to white for better contrast with dark backgrounds. - The function supports displaying axes (‘axes = TRUE‘) and sets plot margins.

Value

This function does not return an object. It directly generates a plot.

See Also

[im.plotRGB.auto()], [im.ggplot()]

Examples

```
library(terra)

# Create a 3-band raster
r <- rast(nrows = 10, ncols = 10, nlyrs = 3)
values(r) <- runif(ncell(r) * 3)

# Plot with user-selected bands (3 = Red, 2 = Green, 1 = Blue)
im.plotRGB(r, r = 3, g = 2, b = 1, title = "Custom RGB Visualization")
```

---

im.plotRGB.auto *Automatically Plot a Raster Image as an RGB Composite*

Description

This function visualizes a multispectral raster image using the first three bands as an RGB composite. It applies a linear contrast stretch to enhance visualization.

Usage

```
im.plotRGB.auto(x, title = "Main")
```

Arguments

x	A 'SpatRaster' object representing the input multispectral image.
title	A character string specifying the plot title (default: "Main").

Details

- The function assumes that the **first three bands** of the raster correspond to the Red, Green, and Blue channels. - It uses 'plotRGB()' with 'stretch="lin"' to enhance contrast. - The plot title is customizable via the 'title' parameter. - The axis and label colors are set to white for better contrast with dark backgrounds.

Value

This function does not return an object. It directly generates a plot.

See Also

[im.import()], [im.ggplot()]

Examples

```
library(terra)

# Create a 3-band raster with random values
r <- rast(nrows = 10, ncols = 10, nlyrs = 3)
values(r) <- runif(ncell(r) * 3)

# Plot RGB composite
im.plotRGB.auto(r, title = "RGB Visualization")
```

---

im.print *Print a Message Identifying the imageRy Package*

Description

This function prints a simple message to indicate that the ‘imageRy‘ package is being used.

Usage

```
im.print()
```

Details

This function serves as a basic test function to verify that the ‘imageRy‘ package is loaded.

Value

This function does not return an object. It prints a message to the console.

See Also

[im.list()], [im.import()]

Examples

```
# Print the imageRy message
im.print()
```

`im.ridgeline` *Generate Ridgeline Plots from Satellite Raster Data*

Description

This function generates ridgeline plots from stacked satellite imagery data.

Usage

```
im.ridgeline(
  im,
  scale,
 palette = c("viridis", "magma", "plasma", "inferno", "cividis", "mako", "rocket",
    "turbo")
)
```

Arguments

`im`	A 'SpatRaster' object representing the raster data to be visualized.
`scale`	A numeric value that defines the vertical scale of the ridgeline plot.
`palette`	A character string specifying the 'viridis' color palette option to use. Available options: '"viridis"', '"magma"', '"plasma"', '"inferno"', '"cividis"', '"mako"', '"rocket"', '"turbo"'.

Details

Ridgeline plots are useful for analyzing temporal variations in raster-based satellite imagery. This function extracts raster values and visualizes their distribution across layers.

Value

A 'ggplot' object displaying the ridgeline plot.

References

See also 'im.import()', 'im.ggplot()'.

See Also

[GitHub Repository](https://github.com/ducciorocchini/imageRy/)

Examples

```
library(terra)
library(ggridges)
library(ggplot2)

# Create a 5-layer raster
r <- rast(nrows = 10, ncols = 10, nlyrs = 5)
values(r) <- runif(ncell(r) * 5)

# Generate ridgeline plot
im.ridgeline(r, scale = 2, palette = "viridis") + theme_minimal()
```

Index