Package 'sits'

July 7, 2022

Type Package

Version 1.1.0

Title Satellite Image Time Series Analysis for Earth Observation Data Cubes

Maintainer Gilberto Camara <gilberto.camara.inpe@gmail.com>

Description An end-to-end toolkit for land use and land cover classification using big Earth observation data, based on machine learning methods applied to satellite image data cubes, as described in Simoes et al (2021) <doi:10.3390/rs13132428>.
Builds regular data cubes from collections in AWS, Microsoft Planetary Computer, Brazil Data Cube, and Digital Earth Africa using the STAC protocol <https://stacspec.org/> and the 'gdalcubes' R package <doi:10.3390/data4030092>.
Supports visualization methods for images and time series and smoothing filters for dealing with noisy time series.
Includes functions for quality assessment of training samples using self-organized maps as presented by Santos et al (2021) <doi:10.1016/j.isprsjprs.2021.04.014>.
Provides machine learning methods including support vector machines, random forests, extreme gradient boosting, multi-layer perceptrons, temporal convolutional neural networks <doi:10.3390/rs11050523>, residual networks <arxiv:1809.04356>, and temporal attention encoders <arXiv:2007.00586>.
Performs efficient classification of big Earth observation data cubes and includes functions for post-classification smoothing based on Bayesian inference, and methods for uncertainty assessment. Enables best practices for estimating area and assessing accuracy of land change as recommended by Olofsson et al(2014) <doi:10.1016/j.rse.2014.02.015>.
Minimum recommended requirements: 16 GB RAM and 4 CPU dual-core.

Encoding UTF-8

Language en-US

Depends R (>= 4.0.0)

URL https://github.com/e-sensing/sits/,
https://e-sensing.github.io/sitsbook/

BugReports https://github.com/e-sensing/sits/issues

License GPL-2

ByteCompile true

LazyData true

Imports magrittr, yaml, data.table (>= 1.13), dplyr (>= 1.0.0), gdalUtilities, grDevices, ggplot2, graphics, lubridate, parallel (>= 4.0.5), purrr (>= 0.3.0), Rcpp, rstac (>= 0.9.1-5), sf (>= 1.0), slider (>= 0.2.0), stats, terra (>= 1.5-17), tibble (>= 3.1), tidyr (>= 1.2.0), torch (>= 0.7.0), utils

Suggests caret, dendextend, dtwclust, dtwSat (>= 0.2.7), DiagrammeR, digest, e1071, FNN, gdalcubes (>= 0.6.0), geojsonsf, httr, jsonlite, kohonen(>= 3.0.11), leafem (>= 0.2.0), leaflet (>= 2.1.1), luz (>= 0.2.0), methods, mgcv, openxlsx, randomForest, randomForestExplainer, RcppArmadillo (>= 0.11), scales, stars (>= 0.5), testthat (>= 3.1.3), torchopt(>= 0.1.2), xgboost, zoo

Config/testthat/edition 3

Config/testthat/parallel false

Config/testthat/start-first cube, raster, ml

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 7.2.0

Collate 'RcppExports.R' 'data.R' 'pipe.R' 'sits-package.R' 'sits_apply.R' 'sits_accuracy.R' 'sits_active_learning.R' 'sits_bands.R' 'sits_bbox.R' 'sits_classification.R' 'sits_classify_ts.R' 'sits_classify_cube.R' 'sits_compare.R' 'sits_config.R' 'sits_csv.R' 'sits_cube.R' 'sits_cube_aux_functions.R' 'sits_check.R' 'sits_cluster.R' 'sits_debug.R' 'sits_distances.R' 'sits_dt_reference.R' 'sits_factory.R' 'sits_file_info.R' 'sits_filters.R' 'sits_gdalcubes.R' 'sits_geo_dist.R' 'sits_get_data.R' 'sits_imputation.R' 'sits_labels.R' 'sits_label_classification.R' 'sits_lighttae.R' 'sits_machine_learning.R' 'sits_merge.R' 'sits_mixture_model.R' 'sits_mlp.R' 'sits_parallel.R' 'sits_patterns.R' 'sits_plot.R' 'sits_raster_api.R' 'sits_raster_api_terra.R' 'sits_raster_blocks.R' 'sits_raster_data.R' 'sits_raster_sub_image.R' 'sits_regularize.R' 'sits_resnet.R' 'sits_roi.R' 'sits_sample_functions.R' 'sits_select.R' 'sits_sf.R' 'sits_shp.R' 'sits_smooth.R' 'sits_smooth_aux_functions.R' 'sits_som.R' 'sits_source_api.R' 'sits_source_api_aws.R' 'sits_source_api_bdc.R' 'sits_source_api_deafrica.R' 'sits_source_api_local.R' 'sits_source_api_mpc.R' 'sits_source_api_sdc.R' 'sits_source_api_stac.R' 'sits_source_api_usgs.R' 'sits_space_time_operations.R' 'sits_stac.R' 'sits_tae.R' 'sits_tempcnn.R' 'sits_torch_conv1d.R' 'sits_torch_linear.R'

'sits_torch_spatial_encoder.R'
'sits_torch_temporal_attention_encoder.R' 'sits_tibble.R'
'sits_timeline.R' 'sits_train.R' 'sits_tuning.R' 'sits_twdtw.R'
'sits_utils.R' 'sits_uncertainty.R' 'sits_validate.R'
'sits_view.R' 'sits_values.R' 'sits_xlsx.R' 'zzz.R'

NeedsCompilation yes

Author Rolf Simoes [aut],
Gilberto Camara [aut, cre],
Felipe Souza [aut],
Lorena Santos [aut],
Pedro Andrade [aut],
Karine Ferreira [aut],
Alber Sanchez [aut],
Gilberto Queiroz [aut]

Repository CRAN

Date/Publication 2022-07-07 20:00:02 UTC

R topics documented:

sits-package *sits*

Description

Satellite Image Time Series Analysis for Earth Observation Data Cubes

Purpose

The SITS package provides a set of tools for analysis, visualization and classification of satellite image time series. It includes methods for filtering, clustering, classification, and post-processing.

Author(s)

Maintainer: Gilberto Camara <gilberto.camara.inpe@gmail.com>

Authors:

- Rolf Simoes <rolf.simoes@inpe.br>
- Felipe Souza <felipe.carvalho@inpe.br>
- Lorena Santos <lorena.santos@inpe.br>
- Pedro Andrade <pedro.andrade@inpe.br>
- Karine Ferreira <karine.ferreira@inpe.br>
- Alber Sanchez <alber.ipia@inpe.br>
- Gilberto Queiroz <gilberto.queiroz@inpe.br>

See Also

Useful links:

- https://github.com/e-sensing/sits/
- https://e-sensing.github.io/sitsbook/
- Report bugs at https://github.com/e-sensing/sits/issues

.sits_get_top_values *Get top values of a raster.*

Description

Get the top values of a raster as a point ‘sf‘ object. The values locations are guaranteed to be separated by a certain number of pixels.

Usage

```
.sits_get_top_values(r_obj, band, n, sampling_window)
```

Arguments

`r_obj`	A raster object.
`band`	A numeric band index used to read bricks.
`n`	Number of values to extract.
`sampling_window`	Window size to collect a point (in pixels).

Value

A point ‘tibble‘ object.

Author(s)

Alber Sanchez, `<alber.ipia@inpe.br>`

:= *Set by reference in data.table*

Description

Data.table assignment by reference.

Arguments

`lhs, rhs`	A visualization and a function to apply to it.

Value

DT is modified by reference and returned invisibly.

`cerrado_2classes` *Samples of classes Cerrado and Pasture*

Description

A dataset containing a tibble with time series samples for the Cerrado and Pasture areas of the Mato Grosso state. The time series come from MOD13Q1 collection 5 images.

Usage

```
data(cerrado_2classes)
```

Format

A tibble with 736 rows and 7 variables: longitude: East-west coordinate of the time series sample (WGS 84), latitude (North-south coordinate of the time series sample in WGS 84), start_date (initial date of the time series), end_date (final date of the time series), label (the class label associated to the sample), cube (the name of the cube associated with the data), time_series (list containing a tibble with the values of the time series).

`plot` *Plot time series*

Description

This is a generic function. Parameters depend on the specific type of input. See each function description for the required parameters:

- sits tibble: see `plot.sits`
- patterns: see `plot.patterns`
- SOM map: see `plot.som_map`
- SOM evaluate cluster: see `plot.som_evaluate_cluster`
- classified time series: see `plot.predicted`
- raster cube: see `plot.raster_cube`
- random forest model: see `plot.rfor_model`
- xgboost model: see `plot.xgb_model`
- torch ML model: see `plot.torch_model`
- classification probabilities: see `plot.probs_cube`
- model uncertainty: see `plot.uncertainty_cube`
- classified image: see `plot.classified_image`

In the case of time series, the plot function produces different plots based on the input data:

- "all years": Plot all samples from the same location together
- "together": Plot all samples of the same band and label together

The plot.sits function makes an educated guess of what plot is required, based on the input data. If the input data has less than 30 samples, it will default to "all years". If there are more than 30 samples, it will default to "together".

Usage

```
## S3 method for class 'sits'
plot(x, y, ...)
```

Arguments

`x`	Object of class "sits"
`y`	Ignored.
`...`	Further specifications for plot.

Value

A series of plot objects produced by ggplot2 showing all time series associated to each combination of band and label, and including the median, and first and third quartile ranges.

Note

Please refer to the sits documentation available in <https://e-sensing.github.io/sitsbook/> for detailed examples.

if (sits_run_examples()) # plot sets of time series plot(cerrado_2classes)

Author(s)

Gilberto Camara, <gilberto.camara@inpe.br>

`plot.classified_image` *Plot classified images*

Description

plots a classified raster using ggplot.

Usage

```
## S3 method for class 'classified_image'
plot(
  x,
  y,
  ...,
  tiles = NULL,
  title = "Classified Image",
  legend = NULL,
  palette = "Spectral",
  rev = TRUE
)
```

Arguments

`x`	Object of class "classified_image".
`y`	Ignored.
`...`	Further specifications for plot.
`tiles`	Tiles to be plotted.
`title`	Title of the plot.
`legend`	Named vector that associates labels to colors.
`palette`	Alternative palette that uses grDevices::hcl.pals().
`rev`	Invert the order of hcl palette?

Value

A plot object produced by the ggplot2 package with a color maps, where each pixel has the color associated to a label, as defined by the legend parameter.

Author(s)

Gilberto Camara, <gilberto.camara@inpe.br>

Examples

```
if (sits_run_examples()) {
    # select a set of samples
    samples_ndvi <- sits_select(samples_modis_4bands, bands = c("NDVI"))
    # create a random forest model
    rfor_model <- sits_train(samples_ndvi, sits_rfor())
    # create a data cube from local files
    data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
    cube <- sits_cube(
        source = "BDC",
        collection = "MOD13Q1-6",
        data_dir = data_dir,
        delim = "_",
        parse_info = c("X1", "X2", "tile", "band", "date")
```

```
    )
    # classify a data cube
    probs_cube <- sits_classify(data = cube, ml_model = rfor_model)
    # label cube with the most likely class
    label_cube <- sits_label_classification(probs_cube)
    # plot the resulting classified image
    plot(label_cube)
}
```

`plot.geo_distances` *Make a kernel density plot of samples distances.*

Description

Make a kernel density plot of samples distances.

Usage

```
## S3 method for class 'geo_distances'
plot(x, y, ...)
```

Arguments

`x`	Object of class "geo_distances".
`y`	Ignored.
`...`	Further specifications for plot.

Value

A plot showing the sample-to-sample distances and sample-to-prediction distances.

Note

Please refer to the sits documentation available in <https://e-sensing.github.io/sitsbook/> for detailed examples.

Author(s)

Felipe Souza, <`lipecaso@gmail.com`>

Rolf Simoes, <`rolf.simoes@inpe.br`>

Alber Sanchez, <`alber.ipia@inpe.br`>

References

Hanna Meyer and Edzer Pebesma, "Machine learning-based global maps of ecological variables and the challenge of assessing them" Nature Communications, 13,2022. DOI: 10.1038/s41467-022-29838-9.

Examples

```
if (sits_run_examples()) {
    # read a shapefile for the state of Mato Grosso, Brazil
    mt_shp <- system.file("extdata/shapefiles/mato_grosso/mt.shp",
        package = "sits"
    )
    # convert to an sf object
    mt_sf <- sf::read_sf(mt_shp)
    # calculate sample-to-sample and sample-to-prediction distances
    distances <- sits_geo_dist(samples_modis_4bands, mt_sf)
    # plot sample-to-sample and sample-to-prediction distances
    plot(distances)
}
```

`plot.patterns` *Plot patterns that describe classes*

Description

Plots the patterns to be used for classification

Given a sits tibble with a set of patterns, plot them.

Usage

```
## S3 method for class 'patterns'
plot(x, y, ...)
```

Arguments

`x`	Object of class "patterns".
`y`	Ignored.
`...`	Further specifications for plot.

Value

A plot object produced by ggplot2 with one average pattern per label.

Note

Please refer to the sits documentation available in <https://e-sensing.github.io/sitsbook/> for detailed examples. This code is reused from the dtwSat package by Victor Maus.

Author(s)

Gilberto Camara, `<gilberto.camara@inpe.br>`

Victor Maus, `<vwmaus1@gmail.com>`

Examples

```
if (sits_run_examples()) {
    # plot patterns
    plot(sits_patterns(cerrado_2classes))
}
```

---

plot.predicted *Plot time series predictions*

Description

Given a sits tibble with a set of predictions, plot them

Usage

```
## S3 method for class 'predicted'
plot(x, y, ..., bands = "NDVI", palette = "Harmonic")
```

Arguments

x	Object of class "predicted".
y	Ignored.
...	Further specifications for plot.
bands	Bands for visualization.
palette	HCL palette used for visualization in case classes are not in the default sits palette.

Value

A plot object produced by ggplot2 showing the time series and its label.

Note

Please refer to the sits documentation available in <https://e-sensing.github.io/sitsbook/> for detailed examples.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

Gilberto Camara, <gilberto.camara@inpe.br>

Examples

```
if (sits_run_examples()) {
    # Retrieve the samples for Mato Grosso
    # train a tempCNN model
    ml_model <- sits_train(samples_modis_4bands, ml_method = sits_tempcnn)
    # classify the point
    bands_model <- sits_bands(ml_model)
    point_4bands <- sits_select(point_mt_6bands, bands = bands_model)
    point_class <- sits_classify(point_4bands, ml_model)
    plot(point_class)
}
```

`plot.probs_cube` *Plot probability cubes*

Description

plots a probability cube using stars

Usage

```
## S3 method for class 'probs_cube'
plot(
  x,
  ...,
  tiles = NULL,
  labels = NULL,
  breaks = "pretty",
  n_colors = 20,
  palette = "Terrain"
)
```

Arguments

`x`	Object of class "probs_image".
`...`	Further specifications for plot.
`tiles`	Tiles to be plotted.
`labels`	Labels to plot (optional).
`breaks`	Type of class intervals.
`n_colors`	Number of colors to plot.
`palette`	HCL palette used for visualization.

Value

A plot object produced by the stars package containing maps of probabilities associated to each class for each pixel.

Note

Possible class intervals

"sd": intervals based on the average and standard deviation.

- "equal": divides the range of the variable into n parts.
- "pretty": number of breaks likely to be legible.
- "quantile": quantile breaks
- "log": logarithm plot

The function accepts color palettes are defined in grDevices::hcl.pals()

Author(s)

Gilberto Camara, <gilberto.camara@inpe.br>

Examples

```
if (sits_run_examples()) {
    # select a set of samples
    samples_ndvi <- sits_select(samples_modis_4bands, bands = c("NDVI"))
    # create a random forest model
    rfor_model <- sits_train(samples_ndvi, sits_rfor())
    # create a data cube from local files
    data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
    cube <- sits_cube(
        source = "BDC",
        collection = "MOD13Q1-6",
        data_dir = data_dir,
        delim = "_",
        parse_info = c("X1", "X2", "tile", "band", "date")
    )
    # classify a data cube
    probs_cube <- sits_classify(data = cube, ml_model = rfor_model)
    # plot the resulting probability cube
    plot(probs_cube)
}
```

---

plot.raster_cube *Plot RGB data cubes*

Description

Plot RGB raster cube

Usage

```
## S3 method for class 'raster_cube'
plot(
  x,
  ...,
  band = NULL,
  red = NULL,
  green = NULL,
  blue = NULL,
  tile = x$tile[[1]],
  date = NULL
)
```

Arguments

x	Object of class "raster_cube".
...	Further specifications for plot.
band	Band for plotting grey images.
red	Band for red color.
green	Band for green color.
blue	Band for blue color.
tile	Tile to be plotted.
date	Date to be plotted.

Value

A plot object produced by the terra package with an RGB or B/W image.

Author(s)

Gilberto Camara, <gilberto.camara@inpe.br>

Examples

```
if (sits_run_examples()) {
    # create a data cube from local files
    data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
    cube <- sits_cube(
        source = "BDC",
        collection = "MOD13Q1-6",
        data_dir = data_dir,
        delim = "_",
        parse_info = c("X1", "X2", "tile", "band", "date")
    )
    # plot NDVI band of the second date date of the data cube
    plot(cube, band = "NDVI", date = sits_timeline(cube)[2])
}
```

plot.rfor_model *Plot Random Forest model*

Description

Plots the important variables in a random forest model.

Usage

```
## S3 method for class 'rfor_model'
plot(x, y, ...)
```

Arguments

x	Object of class "rf_model".
y	Ignored.
...	Further specifications for plot.

Value

A random forest object.

Note

Please refer to the sits documentation available in <https://e-sensing.github.io/sitsbook/> for detailed examples.

Author(s)

Gilberto Camara, <gilberto.camara@inpe.br>

Examples

```
if (sits_run_examples()) {
    # Retrieve the samples for Mato Grosso
    # train a random forest model
    rf_model <- sits_train(samples_modis_4bands,  ml_method = sits_rfor())
    # plot the model
    plot(rf_model)
}
```

plot.som_evaluate_cluster

Plot confusion between clusters

Description

Plot a bar graph with informations about each cluster. The percentage of mixture between the clusters.

Usage

```
## S3 method for class 'som_evaluate_cluster'
plot(x, y, ..., name_cluster = NULL, title = "Confusion by cluster")
```

Arguments

x	Object of class "plot.som_evaluate_cluster".
y	Ignored.
...	Further specifications for plot.
name_cluster	Choose the cluster to plot.
title	Title of plot.

Value

A plot object produced by the ggplot2 package containing color bars showing the confusion between classes.

Note

Please refer to the sits documentation available in <https://e-sensing.github.io/sitsbook/> for detailed examples.

Author(s)

Lorena Santos <lorena.santos@inpe.br>

Examples

```
if (sits_run_examples()) {
    # create a SOM map
    som_map <- sits_som_map(samples_modis_4bands)
    # evaluate the SOM cluster
    som_clusters <- sits_som_evaluate_cluster(som_map)
    # plot the SOM cluster evaluation
    plot(som_clusters)
}
```

plot.som_map *Plot a SOM map*

Description

plots a SOM map generated by "sits_som_map" The plot function produces different plots based on the input data:

- "codes": Plot the vector weight for in each neuron.
- "mapping": Shows where samples are mapped.

Usage

```
## S3 method for class 'som_map'
plot(x, y, ..., type = "codes", band = 1)
```

Arguments

x	Object of class "som_map".
y	Ignored.
...	Further specifications for plot.
type	Type of plot: "codes" for neuron weight (time series) and "mapping" for the number of samples allocated in a neuron.
band	What band will be plotted.

Value

No return value, called for side effects.

Note

Please refer to the sits documentation available in <https://e-sensing.github.io/sitsbook/> for detailed examples.

Author(s)

Gilberto Camara, <gilberto.camara@inpe.br>

Examples

```
if (sits_run_examples()) {
    # create a SOM map
    som_map <- sits_som_map(samples_modis_4bands)
    # plot the SOM map
    plot(som_map)
}
```

plot.torch_model — *Plot Torch (deep learning) model*

Description

Plots a deep learning model developed using torch.

Usage

```
## S3 method for class 'torch_model'
plot(x, y, ...)
```

Arguments

`x`	Object of class "torch_model".
`y`	Ignored.
`...`	Further specifications for plot.

Value

A plot object produced by the ggplot2 package showing the evolution of the loss and accuracy of the model.

Note

This code has been lifted from the "keras" package.

Please refer to the sits documentation available in <https://e-sensing.github.io/sitsbook/> for detailed examples.

Author(s)

Felipe Souza, <lipecaso@gmail.com>

Rolf Simoes, <rolf.simoes@inpe.br>

Alber Sanchez, <alber.ipia@inpe.br>

Examples

```
if (sits_run_examples()) {
    # Retrieve the samples for Mato Grosso
    # train a tempCNN model
    ml_model <- sits_train(samples_modis_4bands, ml_method = sits_tempcnn)
    # plot the model
    plot(ml_model)
}
```

plot.uncertainty_cube *Plot uncertainty cubes*

Description

plots a probability cube using stars

Usage

```
## S3 method for class 'uncertainty_cube'
plot(
  x,
  ...,
  tiles = NULL,
  n_colors = 14,
  intervals = "log",
  palette = "YlOrRd"
)
```

Arguments

`x`	Object of class "probs_image".
`...`	Further specifications for plot.
`tiles`	Tiles to be plotted.
`n_colors`	Number of colors to plot.
`intervals`	Type of class intervals.
`palette`	HCL palette used for visualization.

Value

A plot object produced by the stars package with a map showing the uncertainty associated to each classified pixel.

Note

Possible class intervals

"sd": intervals based on the average and standard deviation.

- "equal": divides the range of the variable into n parts.
- "quantile": quantile breaks
- "pretty": number of breaks likely to be legible.
- "log" : logarithm plot.

Author(s)

Gilberto Camara, `<gilberto.camara@inpe.br>`

Examples

```
if (sits_run_examples()) {
    # select a set of samples
    samples_ndvi <- sits_select(samples_modis_4bands, bands = c("NDVI"))
    # create a random forest model
    rfor_model <- sits_train(samples_ndvi, sits_rfor())
    # create a data cube from local files
    data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
    cube <- sits_cube(
        source = "BDC",
        collection = "MOD13Q1-6",
        data_dir = data_dir,
        delim = "_",
        parse_info = c("X1", "X2", "tile", "band", "date")
    )
    # classify a data cube
    probs_cube <- sits_classify(data = cube, ml_model = rfor_model)
    # calculate uncertainty
    uncert_cube <- sits_uncertainty(probs_cube)
    # plot the resulting uncertainty cube
    plot(uncert_cube)
}
```

`plot.xgb_model` *Plot XGB model*

Description

Plots the important variables in an extreme gradient boosting.

Usage

```
## S3 method for class 'xgb_model'
plot(x, ..., n_trees = 3)
```

Arguments

`x`	Object of class "xgb_model".
`...`	Further specifications for plot.
`n_trees`	Number of trees to be plotted

Value

A plot object.

Note

Please refer to the sits documentation available in <https://e-sensing.github.io/sitsbook/> for detailed examples.

Author(s)

Gilberto Camara, <gilberto.camara@inpe.br>

Examples

```
if (sits_run_examples()) {
    # Retrieve the samples for Mato Grosso
    # train an extreme gradient boosting
    xgb_model <- sits_train(samples_modis_4bands,
           ml_method = sits_xgboost())
    # plot the model
    plot(xgb_model)
}
```

---

point_mt_6bands *A time series sample with data from 2000 to 2016*

Description

A dataset containing a tibble with one time series samples in the Mato Grosso state of Brazil. The time series comes from MOD13Q1 collection 6 images.

Usage

```
data(point_mt_6bands)
```

Format

A tibble with 1 rows and 7 variables: longitude: East-west coordinate of the time series sample (WGS 84), latitude (North-south coordinate of the time series sample in WGS 84), start_date (initial date of the time series), end_date (final date of the time series), label (the class label associated to the sample), cube (the name of the cube associated with the data), time_series (list containing a tibble with the values of the time series).

---

samples_l8_rondonia_2bands *Samples of Amazon tropical forest biome for deforestation analysis*

Description

A sits tibble with time series samples from Brazilian Amazonia rain forest.

The labels are: "Deforestation", "Forest", "NatNonForest" and "Pasture".

The time series were extracted from the Landsat-8 BDC data cube (collection = "LC8_30_16D_STK-1", tiles = "038047"). These time series comprehends a period of 12 months (25 observations) from "2018-07-12" to "2019-07-28". The extracted bands are NDVI and EVI. Cloudy values were removed and interpolated.

Usage

```
data("samples_l8_rondonia_2bands")
```

Format

A `sits` tibble with 160 samples.

`samples_modis_4bands` *Samples of nine classes for the state of Mato Grosso*

Description

A dataset containing a tibble with time series samples for the Mato Grosso state in Brasil. The time series come from MOD13Q1 collection 6 images. The data set has the following classes: Cerrado(379 samples), Forest (131 samples), Pasture (344 samples), and Soy_Corn (364 samples).

Usage

```
data(samples_modis_4bands)
```

Format

A tibble with 1308 rows and 7 variables: longitude: East-west coordinate of the time series sample (WGS 84), latitude (North-south coordinate of the time series sample in WGS 84), start_date (initial date of the time series), end_date (final date of the time series), label (the class label associated to the sample), cube (the name of the cube associated with the data), time_series (list containing a tibble with the values of the time series).

`sits_accuracy` *Assess classification accuracy (area-weighted method)*

Description

This function calculates the accuracy of the classification result. For a set of time series, it creates a confusion matrix and then calculates the resulting statistics using the R package "caret". The time series needs to be classified using `sits_classify`.

Classified images are generated using `sits_classify` followed by `sits_label_classification`. For a classified image, the function uses an area-weighted technique proposed by Olofsson et al. according to [1-3] to produce more reliable accuracy estimates at 95

In both cases, it provides an accuracy assessment of the classified, including Overall Accuracy, Kappa, User's Accuracy, Producer's Accuracy and error matrix (confusion matrix)

Usage

```
sits_accuracy(data, ...)

## S3 method for class 'sits'
sits_accuracy(data, ...)

## S3 method for class 'classified_image'
sits_accuracy(data, ..., validation_csv)
```

Arguments

`data`	Either a data cube with classified images or a set of time series
`...`	Specific parameters
`validation_csv`	A CSV file path with validation data

Value

A list of lists: The error_matrix, the class_areas, the unbiased estimated areas, the standard error areas, confidence interval 95 and the accuracy (user, producer, and overall), or NULL if the data is empty. A confusion matrix assessment produced by the caret package.

Note

Please refer to the sits documentation available in <https://e-sensing.github.io/sitsbook/> for detailed examples.

Author(s)

Rolf Simoes, <rolf.simoes@inpe.br>

Alber Sanchez, <alber.ipia@inpe.br>

References

[1] Olofsson, P., Foody, G.M., Stehman, S.V., Woodcock, C.E. (2013). Making better use of accuracy data in land change studies: Estimating accuracy and area and quantifying uncertainty using stratified estimation. Remote Sensing of Environment, 129, pp.122-131.

[2] Olofsson, P., Foody G.M., Herold M., Stehman, S.V., Woodcock, C.E., Wulder, M.A. (2014) Good practices for estimating area and assessing accuracy of land change. Remote Sensing of Environment, 148, pp. 42-57.

[3] FAO, Map Accuracy Assessment and Area Estimation: A Practical Guide. National forest monitoring assessment working paper No.46/E, 2016.

Examples

```
if (sits_run_examples()) {
    # show accuracy for a set of samples
    train_data <- sits_sample(samples_modis_4bands, n = 200)
    test_data <- sits_sample(samples_modis_4bands, n = 200)
```

```
    rfor_model <- sits_train(train_data, sits_rfor())
    points_class <- sits_classify(test_data, rfor_model)
    acc <- sits_accuracy(points_class)

    # show accuracy for a data cube classification
    # select a set of samples
    samples_ndvi <- sits_select(samples_modis_4bands, bands = c("NDVI"))
    # create a random forest model
    rfor_model <- sits_train(samples_ndvi, sits_rfor())
    # create a data cube from local files
    data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
    cube <- sits_cube(
        source = "BDC",
        collection = "MOD13Q1-6",
        data_dir = data_dir,
        delim = "_",
        parse_info = c("X1", "X2", "tile", "band", "date")
    )
    # classify a data cube
    probs_cube <- sits_classify(data = cube, ml_model = rfor_model)
    # label the probability cube
    label_cube <- sits_label_classification(probs_cube)
    # obtain the ground truth for accuracy assessment
    ground_truth <- system.file("extdata/samples/samples_sinop_crop.csv",
        package = "sits"
    )
    # make accuracy assessment
    as <- sits_accuracy(label_cube, validation_csv = ground_truth)
}
```

---

sits_apply *Apply a function on a set of time series*

Description

Apply a named expression to a sits cube or a sits tibble to be evaluated and generate new bands (indices). In the case of sits cubes, it materializes a new band in `output_dir` using `gdalcubes`.

Usage

```
sits_apply(data, ...)

## S3 method for class 'sits'
sits_apply(data, ...)

## S3 method for class 'raster_cube'
sits_apply(
  data,
  ...,
```

```
  window_size = 3,
  memsize = 1,
  multicores = 2,
  output_dir = getwd(),
  progress = TRUE
)
```

Arguments

`data`	Valid sits tibble or cube
`...`	Named expressions to be evaluated (see details).
`window_size`	An even number representing the size of the sliding window of sits kernel functions used in expressions (for a list of supported kernel functions, please see details).
`memsize`	Memory available for classification (in GB).
`multicores`	Number of cores to be used for classification.
`output_dir`	Directory where files will be saved.
`progress`	Show progress bar?

Details

`sits_apply()` allow any valid R expression to compute new bands. Use R syntax to pass an expression to this function. Besides arithmetic operators, you can use virtually any R function that can be applied to elements of a matrix (functions that are unaware of matrix sizes, e.g. `sqrt()`, `sin()`, `log()`).

Also, `sits_apply()` accepts a predefined set of kernel functions (see below) that can be applied to pixels considering its neighborhood. `sits_apply()` considers a neighborhood of a pixel as a set of pixels equidistant to it (including itself) according the Chebyshev distance. This neighborhood form a square window (also known as kernel) around the central pixel (Moore neighborhood). Users can set the `window_size` parameter to adjust the size of the kernel window. The image is conceptually mirrored at the edges so that neighborhood including a pixel outside the image is equivalent to take the 'mirrored' pixel inside the edge.

`sits_apply()` applies a function to the kernel and its result is assigned to a corresponding central pixel on a new matrix. The kernel slides throughout the input image and this process generates an entire new matrix, which is returned as a new band to the cube. The kernel functions ignores any `NA` values inside the kernel window. Central pixel is `NA` just only all pixels in the window are `NA`.

Kernel functions

Value

A sits tibble or a sits cube with new bands, produced according to the requested expression.

Summarizing kernel functions

- `w_median()`: returns the median of the neighborhood's values.
- `w_sum()`: returns the sum of the neighborhood's values.

- `w_mean()`: returns the mean of the neighborhood's values.
- `w_sd()`: returns the standard deviation of the neighborhood's values.
- `w_var()`: returns the variance of the neighborhood's values.
- `w_min()`: returns the minimum of the neighborhood's values.
- `w_max()`: returns the maximum of the neighborhood's values.

Author(s)

Rolf Simoes, <rolf.simoes@inpe.br>

Felipe Carvalho, <felipe.carvalho@inpe.br>

Gilberto Camara, <gilberto.camara@inpe.br>

Examples

```
# Get a time series
# Apply a normalization function

point2 <-
    sits_select(point_mt_6bands, "NDVI") %>%
    sits_apply(NDVI_norm = (NDVI - min(NDVI)) / (max(NDVI) - min(NDVI)))
```

---

sits_as_sf — *Return a sits_tibble or sits_cube as an sf object.*

Description

Return a sits_tibble or sits_cube as an sf object.

Usage

```
sits_as_sf(data, ..., crs)

## S3 method for class 'sits'
sits_as_sf(data, ..., crs = 4326)

## S3 method for class 'raster_cube'
sits_as_sf(data, ...)
```

Arguments

`data`	A sits tibble or sits cube.
`...`	Additional parameters.
`crs`	A coordinate reference system of samples.

Value

An sf object of point or polygon geometry.

Author(s)

Felipe Carvalho, <felipe.carvalho@inpe.br>

Alber Sanchez, <alber.ipia@inpe.br>

Examples

```
if (sits_run_examples()) {
   # convert sits tibble to an sf object (point)
   sf_object <- sits_as_sf(cerrado_2classes)

   # convert sits cube to an sf object (polygon)
   data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
   cube <- sits_cube(
       source = "BDC",
       collection = "MOD13Q1-6",
       data_dir = data_dir,
       delim = "_",
       parse_info = c("X1", "X2", "tile", "band", "date")
   )
   sf_objet <- sits_as_sf(cube)
}
```

`sits_bands` *Get the names of the bands*

Description

Finds the names of the bands of a set of time series or of a data cube

Usage

```
sits_bands(x)

## S3 method for class 'sits'
sits_bands(x)

## S3 method for class 'sits_cube'
sits_bands(x)

## S3 method for class 'patterns'
sits_bands(x)

## S3 method for class 'sits_model'
sits_bands(x)
```

Arguments

x	Valid sits tibble (time series or a cube)

Value

A vector with the names of the bands.

Author(s)

Gilberto Camara, <gilberto.camara@inpe.br>

Rolf Simoes, <rolf.simoes@inpe.br>

Examples

```
bands <- sits_bands(samples_modis_4bands)
```

`sits_bbox` *Get the bounding box of the data*

Description

Obtain a vector of limits (either on lat/long for time series or in projection coordinates in the case of cubes)

Usage

```
sits_bbox(data, wgs84 = FALSE, ...)

## S3 method for class 'sits'
sits_bbox(data, ...)

## S3 method for class 'sits_cube'
sits_bbox(data, wgs84 = FALSE, ...)
```

Arguments

`data`	Valid sits tibble (time series or a cube).
`wgs84`	Reproject bbox to WGS84 (EPSG:4326)?
`...`	Additional parameters (not implemented).

Value

Bounding box in WGS84 for time series or on the cube projection for a data cube unless wgs84 parameter is TRUE.

Author(s)

Gilberto Camara, <gilberto.camara@inpe.br>

Rolf Simoes, <rolf.simoes@inpe.br>

Examples

```
bbox <- sits_bbox(samples_modis_4bands)
```

---

sits_classify *Classify time series or data cubes*

Description

This function classifies a set of time series or data cube given a trained model prediction model created by sits_train.

SITS supports the following models:

- support vector machines: see sits_svm
- random forests: see sits_rfor
- extreme gradient boosting: see sits_xgboost
- multi-layer perceptrons: see sits_mlp
- 1D CNN: see sits_tempcnn
- deep residual networks:see sits_resnet
- self-attention encoders:see sits_lighttae

Usage

```
sits_classify(data, ml_model, ...)

## S3 method for class 'sits'
sits_classify(data, ml_model, ..., filter_fn = NULL, multicores = 2)

## S3 method for class 'raster_cube'
sits_classify(
  data,
  ml_model,
  ...,
  roi = NULL,
  filter_fn = NULL,
  impute_fn = sits_impute_linear(),
  start_date = NULL,
  end_date = NULL,
  memsize = 8,
  multicores = 2,
  output_dir = ".",
  version = "v1",
  verbose = FALSE,
  progress = FALSE
)
```

Arguments

`data`	Data cube.
`ml_model`	R model trained by `sits_train`.
`...`	Other parameters for specific functions.
`filter_fn`	Smoothing filter to be applied (if desired).
`multicores`	Number of cores to be used for classification.
`roi`	Region of interest (see below)
`impute_fn`	Impute function to replace NA.
`start_date`	Start date for the classification.
`end_date`	End date for the classification.
`memsize`	Memory available for classification (in GB).
`output_dir`	Directory for output file.
`version`	Version of the output (for multiple classifications).
`verbose`	Print information about processing time?
`progress`	Show progress bar?

Value

Predicted data (classified time series) or a data cube with probabilities for each class.

Note

The "roi" parameter defines a region of interest. It can be an sf_object, a shapefile, or a bounding box vector with named XY values ("xmin", "xmax", "ymin", "ymax") or named lat/long values ("lon_min", "lat_min", "lon_max", "lat_max")

The "filter_fn" parameter specifies a smoothing filter to be applied to time series for reducing noise. Currently, options include Savitzky-Golay (see `sits_sgolay`) and Whittaker (see `sits_whittaker`).

The "impute_fn" function is used to remove invalid or cloudy pixels from time series. The default is a linear interpolator, available in `sits_impute_linear`. Users can add their custom functions.

The "memsize" and "multicores" parameters are used for multiprocessing. The "multicores" parameter defines the number of cores used for processing. The "memsize" parameter controls the amount of memory available for classification. We recommend using a 4:1 relation between "memsize" and "multicores".

Please refer to the sits documentation available in <https://e-sensing.github.io/sitsbook/> for detailed examples.

Author(s)

Rolf Simoes, <rolf.simoes@inpe.br>

Gilberto Camara, <gilberto.camara@inpe.br>

Examples

```
if (sits_run_examples()) {
    # Example of classification of a time series
    # Retrieve the samples for Mato Grosso
    # select the NDVI band
    samples_ndvi <- sits_select(samples_modis_4bands, bands = c("NDVI"))
    # train a random forest model
    rf_model <- sits_train(samples_ndvi, ml_method = sits_rfor)

    # classify the point
    point_ndvi <- sits_select(point_mt_6bands, bands = c("NDVI"))
    point_class <- sits_classify(point_ndvi, rf_model)
    plot(point_class)

    # Example of classification of a data cube
    # create a data cube from local files
    data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
    cube <- sits_cube(
        source = "BDC",
        collection = "MOD13Q1-6",
        data_dir = data_dir,
        delim = "_",
        parse_info = c("X1", "X2", "tile", "band", "date")
    )
    # classify a data cube
    probs_cube <- sits_classify(data = cube, ml_model = rf_model)
    # label the probability cube
    label_cube <- sits_label_classification(probs_cube)
    # plot the classified image
    plot(label_cube)
}
```

`sits_clustering` *Find clusters in time series samples*

Description

These functions support hierarchical agglomerative clustering in sits. They provide support from creating a dendrogram and using it for cleaning samples.

`sits_cluster_dendro()` takes a tibble containing time series and produces a sits tibble with an added "cluster" column. The function first calculates a dendrogram and obtains a validity index for best clustering using the adjusted Rand Index. After cutting the dendrogram using the chosen validity index, it assigns a cluster to each sample.

`sits_cluster_frequency()` computes the contingency table between labels and clusters and produces a matrix. It needs as input a tibble produced by `sits_cluster_dendro()`.

`sits_cluster_clean()` takes a tibble with time series that has an additional ‘cluster‘ produced by `sits_cluster_dendro()` and removes labels that are minority in each cluster.

Usage

```
sits_cluster_dendro(
  samples = NULL,
  bands = NULL,
  dist_method = "dtw_basic",
  linkage = "ward.D2",
  k = NULL,
  palette = "RdYlGn",
  .plot = TRUE,
  ...
)
```

Arguments

`samples`	Tibble with input set of time series.
`bands`	Bands to be used in the clustering.
`dist_method`	Distance method.
`linkage`	Agglomeration method. Can be any ‘hclust‘ method (see ‘hclust‘). Default is ’ward.D2’.
`k`	Desired number of clusters (overrides default value)
`palette`	Color palette as per ‘grDevices::hcl.pals()‘ function.
`.plot`	Plot the dendrogram?
`...`	Additional parameters to be passed to dtwclust::tsclust() function.

Value

Tibble with added "cluster" column.

Note

Please refer to the sits documentation available in <https://e-sensing.github.io/sitsbook/> for detailed examples.

Author(s)

Rolf Simoes, `<rolf.simoes@inpe.br>`

References

"dtwclust" package (https://CRAN.R-project.org/package=dtwclust)

Examples

```
if (sits_run_examples()) {
    clusters <- sits_cluster_dendro(cerrado_2classes)
}
```

sits_cluster_clean *Removes labels that are minority in each cluster.*

Description

Takes a tibble with time series that has an additional ‘cluster‘ produced by `sits_cluster_dendro()` and removes labels that are minority in each cluster.

Usage

```
sits_cluster_clean(samples)
```

Arguments

`samples`	Tibble with input set of time series with additional cluster information produced by `sits::sits_cluster_dendro()`.

Value

Tibble with time series where clusters have been cleaned of labels that were in a minority at each cluster.

Author(s)

Rolf Simoes, <rolf.simoes@inpe.br>

Examples

```
if (sits_run_examples()) {
    clusters <- sits_cluster_dendro(cerrado_2classes)
    freq1 <- sits_cluster_frequency(clusters)
    freq1
    clean_clusters <- sits_cluster_clean(clusters)
    freq2 <- sits_cluster_frequency(clean_clusters)
    freq2
}
```

sits_cluster_frequency *Show label frequency in each cluster produced by dendrogram analysis*

Description

Show label frequency in each cluster produced by dendrogram analysis

Usage

```
sits_cluster_frequency(samples)
```

Arguments

`samples`	Tibble with input set of time series with additional cluster information produced by `sits::sits_cluster_dendro`.

Value

A matrix containing frequencies of labels in clusters.

Author(s)

Rolf Simoes, <rolf.simoes@inpe.br>

Examples

```
if (sits_run_examples()) {
    clusters <- sits_cluster_dendro(cerrado_2classes)
    freq <- sits_cluster_frequency(clusters)
    freq
}
```

---

sits_confidence_sampling

Suggest high confidence samples to increase the training set.

Description

Suggest points for increasing the training set. These points are labelled with high confidence so they can be added to the training set. They need to have a satisfactory margin of confidence to be selected. The input is a probability cube. For each label, the algorithm finds out location where the machine learning model has high confidence in choosing this label compared to all others. The algorithm also considers a minimum distance between new labels, to minimize spatial autocorrelation effects.

This function is best used in the following context

- 1. Select an initial set of samples.
- 2. Train a machine learning model.
- 3. Build a data cube and classify it using the model.
- 4. Run a Bayesian smoothing in the resulting probability cube.
- 5. Create an uncertainty cube.
- 6. Perform confidence sampling.

The Bayesian smoothing procedure will reduce the classification outliers and thus increase the likelihood that the resulting pixels with provide good quality samples for each class.

Usage

```
sits_confidence_sampling(
  probs_cube,
  n = 20,
  min_margin = 0.9,
  sampling_window = 10
)
```

Arguments

probs_cube	A probability cube. See sits_classify.
n	Number of suggested points per class.
min_margin	Minimum margin of confidence to select a sample
sampling_window	Window size for collecting points (in pixels). The minimum window size is 10.

Value

A tibble with longitude and latitude in WGS84 with locations which have high uncertainty and meet the minimum distance criteria.

Author(s)

Alber Sanchez, <alber.ipia@inpe.br>

Rolf Simoes, <rolf.simoes@inpe.br>

Felipe Carvalho, <felipe.carvalho@inpe.br>

Gilberto Camara, <gilberto.camara@inpe.br>

Examples

```
if (sits_run_examples()) {
    # create a data cube
    data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
    cube <- sits_cube(
        source = "BDC",
        collection = "MOD13Q1-6",
        data_dir = data_dir,
        delim = "_",
        parse_info = c("X1", "X2", "tile", "band", "date")
    )
    # build a random forest model
    samples_ndvi <- sits_select(samples_modis_4bands, bands = c("NDVI"))
    rfor_model <- sits_train(samples_ndvi, ml_method = sits_rfor())
    # classify the cube
    probs_cube <- sits_classify(data = cube, ml_model = rfor_model)
    # obtain a new set of samples for active learning
    # the samples are located in uncertain places
    new_samples <- sits_confidence_sampling(probs_cube)
}
```

sits_configuration   *Configure parameters for sits package*

Description

These functions load and show sits configurations.

The ‘sits‘ package uses a configuration file that contains information on parameters required by different functions. This includes information about the image collections handled by ‘sits‘.

`sits_config()` loads the default configuration file and the user provided configuration file. The final configuration is obtained by overriding the options by the values provided in `processing_bloat`, `rstac_pagination_limit`, `raster_api_package`, and `gdal_creation_options` parameters.

`sits_config_show()` prints the current sits configuration options. To show specific configuration options for a source, a collection, or a palette, users can inform the corresponding keys to `source`, `collection`, and `palette` parameters.

`sits_list_collections()` prints the collections available in each cloud service supported by sits. Users can select to get information only for a single service by using the `source` parameter.

Usage

```
sits_config(
  run_tests = NULL,
  run_examples = NULL,
  processing_bloat = NULL,
  rstac_pagination_limit = NULL,
  raster_api_package = NULL,
  gdal_creation_options = NULL,
  gdalcubes_chunk_size = NULL,
  leaflet_max_megabytes = NULL,
  leaflet_comp_factor = NULL,
  reset = FALSE
)

sits_config_show(source = NULL, collection = NULL, colors = FALSE)

sits_list_collections(source = NULL)
```

Arguments

`run_tests` Should tests be run?

`run_examples` Should examples be run?

`processing_bloat`
Estimated growth size of R memory relative to block size.

`rstac_pagination_limit`
Limit of number of items returned by STAC.

`raster_api_package`
: Supported raster handling package.

`gdal_creation_options`
: GDAL creation options for GeoTiff.

`gdalcubes_chunk_size`
: Chunk size to be used by gdalcubes

`leaflet_max_megabytes`
: Max image size of an image for leaflet (in MB)

`leaflet_comp_factor`
: Compression factor for leaflet RGB display.

`reset`
: Should current configuration options be cleaned before loading config files? Default is `FALSE`.

`source`
: Data source to be shown in detail.

`collection`
: Collection key entry to be shown in detail.

`colors`
: Show colors?

Details

Users can provide additional configuration files, by specifying the location of their file in the environmental variable `SITS_CONFIG_USER_FILE`.

To see the key entries and contents of the current configuration values, use `sits_config_show()`.

Value

`sits_config()` returns a `list` containing the final configuration options.

A `list` containing the respective configuration printed in the console.

Prints collections available in each cloud service supported by sits.

Author(s)

Gilberto Camara, `<gilberto.camara@inpe.br>`

Rolf Simoes, `<rolf.simoes@inpe.br>`

Examples

```
current_config <- sits_config()
sits_config_show()
```

sits_cube *Create data cubes from image collections*

Description

Creates a data cube based on spatial and temporal restrictions in collections available in cloud services or local repositories. The following cloud providers are supported, based on the STAC protocol:

- "AWS": Amazon Web Services (AWS), see https://registry.opendata.aws/
- "BDC": Brazil Data Cube (BDC), see http://brazildatacube.org/
- "DEAFRICA": Digital Earth Africa, see https://www.digitalearthafrica.org/
- "MPC": Microsoft Planetary Computer, see https://planetarycomputer.microsoft.com/
- "USGS":USGS LANDSAT collection, see https://registry.opendata.aws/usgs-landsat/

Data cubes can also be created using local files (see details).

Usage

```
sits_cube(source, collection, ..., data_dir = NULL)

## S3 method for class 'stac_cube'
sits_cube(
  source,
  collection,
  ...,
  data_dir = NULL,
  bands = NULL,
  tiles = NULL,
  roi = NULL,
  start_date = NULL,
  end_date = NULL,
  platform = NULL
)

## S3 method for class 'local_cube'
sits_cube(
  source,
  collection,
  data_dir,
  ...,
  bands = NULL,
  start_date = NULL,
  end_date = NULL,
  labels = NULL,
  parse_info = NULL,
```

```
  delim = "_",
  multicores = 2,
  progress = TRUE
)
```

Arguments

`source`	Data source (one of `"AWS"`, `"BDC"`, `"DEAFRICA"`, `"MPC"`, `"USGS"`).
`collection`	Image collection in data source (To find out the supported collections, use `sits_list_collections()`).
`...`	Other parameters to be passed for specific types.
`data_dir`	Local directory where images are stored (for local cubes).
`bands`	Spectral bands and indices to be included in the cube (optional).
`tiles`	Tiles from the collection to be included in the cube (see details below).
`roi`	Filter collection by region of interest (see details below).
`start_date, end_date`	Initial and final dates to include images from the collection in the cube (optional).
`platform`	Optional parameter specifying the platform in case of collections that include more than one satellite.
`labels`	Labels associated to the classes (only for result cubes).
`parse_info`	Parsing information for local files.
`delim`	Delimiter for parsing local files.
`multicores`	Number of workers for parallel processing
`progress`	Show a progress bar?

Details

To create cubes from cloud providers, users need to inform:

- `source`: One of `"AWS"`, `"BDC"`, `"DEAFRICA"`, `"MPC"`, `"USGS"`.
- `collection`: Use `sits_list_collections()` to see which collections are supported.
- `tiles`: A set of tiles defined according to the collection tiling grid.
- `roi`: Region of interest in WGS84 coordinates.

Either `tiles` or `roi` must be informed. The parameters `bands`, `start_date`, and `end_date` are optional for cubes created from cloud providers.

The `roi` parameter allows a selection of an area of interest, either using a named `vector` (`"lon_min"`, `"lat_min"`, `"lon_max"`, `"lat_max"`) in WGS84, a `sfc` or `sf` object from sf package in WGS84 projection. GeoJSON geometries (RFC 7946) and shapefiles should be converted to `sf` objects before being used to define a region of interest. This parameter does not crop a region; it only selects images that intersect the `roi`.

To create a cube from local files, users need to inform:

- `source`: Provider from where the data has been downloaded (e.g, "BDC", "AWS").

- `collection`:Collection where the data has been extracted from.
- `data_dir`: Local directory where images are stored.
- `parse_info`: Parsing information for files (see below).
- `delim`: Delimiter character for parsing files (see below).

To create a cube from local files, all images should have the same spatial resolution and projection and each file should contain a single image band for a single date. Files can belong to different tiles of a spatial reference system and file names need to include tile, date, and band information. For example: `"CBERS-4_022024_B13_2018-02-02.tif"` and `"cube_20LKP_B02_2018-07-18.jp2"` are accepted names. The user has to provide parsing information to allow `sits` to extract values of tile, band, and date. In the examples above, the parsing info is `c("X1", "tile", "band", "date")` and the delimiter is `"_"`.

It is also possible to create result cubes; these are local cubes that have been produced by classification or post-classification algorithms. In this case, there are more parameters that are required (see below) and the parameter `parse_info` is specified differently:

- `band`: The band name is associated to the type of result. Use `"probs"`, for probability cubes produced by `sits_classify()`; `"bayes"`, or `"bilat"` (bilateral) according to the function selected when using `sits_smooth()`; `"entropy"` when using `sits_uncertainty()`, or `"class"` for cubes produced by `sits_label_classification()`.
- `labels`: Labels associated to the classification results.
- `parse_info`: File name parsing information has to allow `sits` to deduce the values of "tile", "start_date", "end_date" from the file name. Default is `c("X1", "X2", "tile", "start_date", "end_date", "band")`. Note that, unlike non-classified image files, cubes with results have both "start_date" and "end_date".

Value

A `tibble` describing the contents of a data cube.

Note

In MPC, sits can access are two open data collections: `"SENTINEL-S2-L2A"` for Sentinel-2/2A images, and `"LANDSAT-C2-L2"` for the Landsat-4/5/7/8/9 collection. (requester-pays) and `"SENTINEL-S2-L2A-COGS"` (open data).

Sentinel-2/2A level 2A files in MPC are organized by sensor resolution. The bands in 10m resolution are `"B02"`, `"B03"`, `"B04"`, and `"B08"`. The 20m bands are `"B05"`, `"B06"`, `"B07"`, `"B8A"`, `"B11"`, and `"B12"`. Bands `"B01"` and `"B09"` are available at 60m resolution. The `"CLOUD"` band is also available.

All Landsat-4/5/7/8/9 images in MPC have bands with 30 meter resolution. To account for differences between the different sensors, Landsat bands in this collection have been renamed `"BLUE"`, `"GREEN"`, `"RED"`, `"NIR08"`, `"SWIR16"` and `"SWIR22"`. The `"CLOUD"` band is also available.

In AWS, there are two types of collections: open data and requester-pays. Currently, `sits` supports collection `"SENTINEL-S2-L2A"` (requester-pays) and `"SENTINEL-S2-L2A-COGS"` (open data). There is no need to provide AWS credentials to access open data collections. For requester-pays data, users need to provide their access codes as environment variables, as follows: `Sys.setenv(`

AWS_ACCESS_KEY_ID = <your_access_key>, AWS_SECRET_ACCESS_KEY = <your_secret_access_key>
)

Sentinel-2/2A level 2A files in AWS are organized by sensor resolution. The AWS bands in 10m resolution are "B02", "B03", "B04", and "B08". The 20m bands are "B05", "B06", "B07", "B8A", "B11", and "B12". Bands "B01" and "B09" are available at 60m resolution.

For DEAFRICA, sits currently works with collection "S2_L2A" (open data). This collection is the same as AWS collection "SENTINEL-S2-L2A-COGS", and is located in Africa (Capetown) for faster access to African users. No payment for access is required.

For USGS, sits currently works with collection "LANDSAT-C2L2-SR", which corresponds to Landsat Collection 2 Level-2 surface reflectance data, covering Landsat-8 dataset. This collection is requester-pays and requires payment for accessing.

All BDC collections are regularized. BDC users need to provide their credentials using environment variables. To create your credentials, please see <brazil-data-cube.github.io/applications/dc_explorer/token-module.html>. Accessing data in the BDC is free. After obtaining the BDC access key, please include it as an environment variable, as follows: Sys.setenv(BDC_ACCESS_KEY = <your_bdc_access_key>
)

Please refer to the sits documentation available in <https://e-sensing.github.io/sitsbook/> for detailed examples.

References

rstac package (https://github.com/brazil-data-cube/rstac)

Examples

```
if (sits_run_examples()) {

    # --- Access to the Brazil Data Cube
    # Provide your BDC credentials as environment variables
    bdc_access_key <- Sys.getenv("BDC_ACCESS_KEY")
    if (nchar(bdc_access_key) == 0) {
        stop("No BDC_ACCESS_KEY defined in environment.")
    }

    # create a raster cube file based on the information in the BDC
    cbers_tile <- sits_cube(
        source = "BDC",
        collection = "CB4_64_16D_STK-1",
        bands = c("NDVI", "EVI"),
        tiles = "022024",
        start_date = "2018-09-01",
        end_date = "2019-08-28"
    )

    # --- Access to Digital Earth Africa
    # create a raster cube file based on the information about the files
    # DEAFRICA does not support definition of tiles
    cube_dea <- sits_cube(
        source = "DEAFRICA",
```

```
    collection = "s2_l2a",
    bands = c("B04", "B08"),
    roi = c(
        "lat_min" = 17.379,
        "lon_min" = 1.1573,
        "lat_max" = 17.410,
        "lon_max" = 1.1910
    ),
    start_date = "2019-01-01",
    end_date = "2019-10-28"
)

# --- Access to AWS open data Sentinel 2/2A level 2 collection
s2_cube <- sits_cube(
    source = "AWS",
    collection = "sentinel-s2-l2a-cogs",
    tiles = c("20LKP", "20LLP"),
    bands = c("B04", "B08", "B11"),
    start_date = "2018-07-18",
    end_date = "2019-07-23"
)

# --- Access to USGS Landsat cubes (requester pays)
# --- Need to provide AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY
usgs_cube <- sits_cube(
    source = "USGS",
    collection = "landsat-c2l2-sr",
    bands = c("B04", "CLOUD"),
    roi = c(
        "xmin" = -50.379,
        "ymin" = -10.1573,
        "xmax" = -50.410,
        "ymax" = -10.1910
    ),
    start_date = "2019-01-01",
    end_date = "2019-10-28"
)


# -- Creating Sentinel cube from MPC"
s2_cube <- sits_cube(
    source = "MPC",
    collection = "sentinel-2-l2a",
    tiles = "20LKP",
    bands = c("B05", "CLOUD"),
    start_date = "2018-07-18",
    end_date = "2018-08-23"
)

# -- Creating Landsat cube from MPC"
mpc_cube <- sits_cube(
    source = "MPC",
    collection = "LANDSAT-C2-L2",
```

```
        bands = c("BLUE", "RED", "CLOUD"),
        roi = c(
            "xmin" = -50.379,
            "ymin" = -10.1573,
            "xmax" = -50.410,
            "ymax" = -10.1910
        ),
        start_date = "2005-01-01",
        end_date = "2006-10-28"
    )

    # --- Create a cube based on a local MODIS data
    data_dir <- system.file("extdata/raster/mod13q1", package = "sits")

    modis_cube <- sits_cube(
        source = "BDC",
        collection = "MOD13Q1-6",
        data_dir = data_dir,
        delim = "_"
    )
}
```

`sits_filters` *Filter time series and data cubes*

Description

Filtering functions should be used with ‘sits_filter()‘. The following filtering functions is supported by ‘sits‘:

‘sits_whittaker()‘: The algorithm searches for an optimal warping polynomial. The degree of smoothing depends on smoothing factor lambda (usually from 0.5 to 10.0). Use lambda = 0.5 for very slight smoothing and lambda = 5.0 for strong smoothing.

‘sits_filter()‘: applies a filter to all bands.

‘sits_sgolay()‘: An optimal polynomial for warping a time series. The degree of smoothing depends on the filter order (usually 3.0). The order of the polynomial uses the parameter ‘order‘ (default = 3), the size of the temporal window uses the parameter ‘length‘ (default = 5).

Usage

```
sits_whittaker(data = NULL, lambda = 0.5)

sits_filter(data, filter = sits_whittaker())

sits_sgolay(data = NULL, order = 3, length = 5)
```

Arguments

`data`	Time series or matrix.
`lambda`	Smoothing factor to be applied (default 0.5).
`filter`	a filter function such as ‘sits_whittaker()‘ or ‘sits_sgolay()‘.
`order`	Filter order (integer).
`length`	Filter length (must be odd).

Value

Filtered time series

Author(s)

Rolf Simoes, `<rolf.simoes@inpe.br>`

Gilberto Camara, `<gilberto.camara@inpe.br>`

Felipe Carvalho, `<felipe.carvalho@inpe.br>`

References

Francesco Vuolo, Wai-Tim Ng, Clement Atzberger, "Smoothing and gap-filling of high resolution multi-spectral time series: Example of Landsat data", Int Journal of Applied Earth Observation and Geoinformation, vol. 57, pg. 202-213, 2107.

A. Savitzky, M. Golay, "Smoothing and Differentiation of Data by Simplified Least Squares Procedures". Analytical Chemistry, 36 (8): 1627–39, 1964.

See Also

sits_apply

Examples

```
# Retrieve a time series with values of NDVI
point_ndvi <- sits_select(point_mt_6bands, bands = "NDVI")

# Filter the point using the Whittaker smoother
point_whit <- sits_filter(point_ndvi, sits_whittaker(lambda = 3.0))
# Merge time series
point_ndvi <- sits_merge(point_ndvi, point_whit, suffix = c("", ".WHIT"))

# Plot the two points to see the smoothing effect
plot(point_ndvi)

# Retrieve a time series with values of NDVI
point_ndvi <- sits_select(point_mt_6bands, bands = "NDVI")

# Filter the point using the Savitzky-Golay smoother
point_sg <- sits_filter(point_ndvi,
    filter = sits_sgolay(order = 3, length = 5)
```

```
)
# Merge time series
point_ndvi <- sits_merge(point_ndvi, point_sg, suffix = c("", ".SG"))

# Plot the two points to see the smoothing effect
plot(point_ndvi)
```

`sits_formula_linear` *Define a linear formula for classification models*

Description

Provides a symbolic description of a fitting model. Tells the model to do a linear transformation of the input values. The ‘predictors_index‘ parameter informs the positions of fields corresponding to formula independent variables. If no value is given, that all fields will be used as predictors.

Usage

```
sits_formula_linear(predictors_index = -2:0)
```

Arguments

`predictors_index`
: Index of the valid columns whose names are used to compose formula (default: -2:0).

Value

A function that computes a valid formula using a linear function.

Author(s)

Gilberto Camara, <gilberto.camara@inpe.br>

Alexandre Ywata de Carvalho, <alexandre.ywata@ipea.gov.br>

Rolf Simoes, <rolf.simoes@inpe.br>

Examples

```
if (sits_run_examples()) {
    # Example of training a model for time series classification
    # Retrieve the samples for Mato Grosso
    # train an SVM model
    ml_model <- sits_train(samples_modis_4bands,
        ml_method = sits_svm(formula = sits_formula_logref()))
    # select the bands to classify the point
    sample_bands <- sits_bands(samples_modis_4bands)
    point_4bands <- sits_select(point_mt_6bands, bands = sample_bands)
    # classify the point
    point_class <- sits_classify(point_4bands, ml_model)
```

```
    plot(point_class)
}
```

sits_formula_logref   *Define a loglinear formula for classification models*

Description

A function to be used as a symbolic description of some fitting models such as svm and random forest. This function tells the models to do a log transformation of the inputs. The ‘predictors_index‘ parameter informs the positions of ‘tb‘ fields corresponding to formula independent variables. If no value is given, the default is NULL, a value indicating that all fields will be used as predictors.

Usage

```
sits_formula_logref(predictors_index = -2:0)
```

Arguments

`predictors_index`
: Index of the valid columns to compose formula (default: -2:0).

Value

A function that computes a valid formula using a log function.

Author(s)

Alexandre Ywata de Carvalho, <alexandre.ywata@ipea.gov.br>

Rolf Simoes, <rolf.simoes@inpe.br>

Examples

```
if (sits_run_examples()) {
    # Example of training a model for time series classification
    # Retrieve the samples for Mato Grosso
    # train an SVM model
    ml_model <- sits_train(samples_modis_4bands,
        ml_method = sits_svm(formula = sits_formula_logref()))
    # select the bands to classify the point
    sample_bands <- sits_bands(samples_modis_4bands)
    point_4bands <- sits_select(point_mt_6bands, bands = sample_bands)
    # classify the point
    point_class <- sits_classify(point_4bands, ml_model)
    plot(point_class)
}
```

sits_geo_dist — *Compute the minimum distances among samples and prediction points.*

Description

Compute the minimum distances among samples and samples to prediction points, following the approach proposed by Meyer and Pebesma(2022).

Usage

```
sits_geo_dist(samples, roi = NULL, n = 1000)
```

Arguments

`samples`	A 'sits' tibble with time series samples.
`roi`	A 'sf' object (polygon) with a region of interest for prediction.
`n`	Maximum number of samples to consider.

Value

A tibble with sample-to-sample and sample-to-prediction distances.

Author(s)

Alber Sanchez, <alber.ipia@inpe.br>

Rolf Simoes, <rolf.simoes@inpe.br>

Felipe Carvalho, <felipe.carvalho@inpe.br>

Gilberto Camara, <gilberto.camara@inpe.br>

References

Meyer, H., Pebesma, E. "Machine learning-based global maps of ecological variables and the challenge of assessing them", Nature Communications 13, 2208 (2022). https://doi.org/10.1038/s41467-022-29838-9

Examples

```
if (sits_run_examples()) {
    # read a shapefile for the state of Mato Grosso, Brazil
    mt_shp <- system.file("extdata/shapefiles/mato_grosso/mt.shp",
        package = "sits"
    )
    # convert to an sf object
    mt_sf <- sf::read_sf(mt_shp)
    # calculate sample-to-sample and sample-to-prediction distances
    distances <- sits_geo_dist(samples_modis_4bands, mt_sf)
```

```
    # plot sample-to-sample and sample-to-prediction distances
    plot(distances)
}
```

sits_get_data *Get time series from data cubes and cloud services*

Description

Retrieve a set of time series from a data cube or from a time series service. Data cubes and puts it in a "sits tibble". Sits tibbles are the main structures of sits package. They contain both the satellite image time series and their metadata.

Usage

```
sits_get_data(
  cube,
  samples,
  ...,
  start_date = as.Date(sits_timeline(cube)[1]),
  end_date = as.Date(sits_timeline(cube)[length(sits_timeline(cube))]),
  label = "NoClass",
  bands = sits_bands(cube),
  crs = 4326,
  impute_fn = sits_impute_linear(),
  label_attr = NULL,
  n_sam_pol = 30,
  pol_avg = FALSE,
  pol_id = NULL,
  multicores = 2,
  output_dir = ".",
  progress = FALSE
)

## Default S3 method:
sits_get_data(cube, samples, ...)

## S3 method for class 'csv'
sits_get_data(
  cube,
  samples,
  ...,
  bands = sits_bands(cube),
  crs = 4326,
  impute_fn = sits_impute_linear(),
  multicores = 2,
  output_dir = ".",
```

```
  progress = FALSE
)

## S3 method for class 'shp'
sits_get_data(
  cube,
  samples,
  ...,
  label = "NoClass",
  start_date = as.Date(sits_timeline(cube)[1]),
  end_date = as.Date(sits_timeline(cube)[length(sits_timeline(cube))]),
  bands = sits_bands(cube),
  impute_fn = sits_impute_linear(),
  label_attr = NULL,
  n_sam_pol = 30,
  pol_avg = FALSE,
  pol_id = NULL,
  multicores = 2,
  output_dir = ".",
  progress = FALSE
)

## S3 method for class 'sf'
sits_get_data(
  cube,
  samples,
  ...,
  bands = sits_bands(cube),
  start_date = as.Date(sits_timeline(cube)[1]),
  end_date = as.Date(sits_timeline(cube)[length(sits_timeline(cube))]),
  impute_fn = sits_impute_linear(),
  label = "NoClass",
  label_attr = NULL,
  n_sam_pol = 30,
  pol_avg = FALSE,
  pol_id = NULL,
  multicores = 2,
  output_dir = ".",
  progress = FALSE
)

## S3 method for class 'sits'
sits_get_data(
  cube,
  samples,
  ...,
  bands = sits_bands(cube),
  impute_fn = sits_impute_linear(),
```

```
  multicores = 2,
  output_dir = ".",
  progress = FALSE
)

## S3 method for class 'data.frame'
sits_get_data(
  cube,
  samples,
  ...,
  start_date = as.Date(sits_timeline(cube)[1]),
  end_date = as.Date(sits_timeline(cube)[length(sits_timeline(cube))]),
  label = "NoClass",
  bands = sits_bands(cube),
  crs = 4326,
  impute_fn = sits_impute_linear(),
  multicores = 2,
  output_dir = ".",
  progress = FALSE
)
```

Arguments

cube	Data cube from where data is to be retrieved.
samples	Samples location (sits, sf, or data.frame).
...	Specific parameters for specific cases.
start_date	Start of the interval for the time series in "YYYY-MM-DD" format (optional).
end_date	End of the interval for the time series in "YYYY-MM-DD" format (optional).
label	Label to be assigned to the time series (optional).
bands	Bands to be retrieved (optional).
crs	A coordinate reference system of samples. The provided crs could be a character (e.g, "EPSG:4326" or "WGS84" or a proj4string), or a a numeric with the EPSG code (e.g. 4326). This parameter only works for 'csv' or data.frame' samples. Default is 4326.
impute_fn	Imputation function for NA values.
label_attr	Attribute in the shapefile or sf object to be used as a polygon label.
n_sam_pol	Number of samples per polygon to be read (for POLYGON or MULTIPOLYGON shapefile).
pol_avg	Summarize samples for each polygon?
pol_id	ID attribute for polygons.
multicores	Number of threads to process the time series.
output_dir	Directory where the time series will be saved as rds. Default is the current path.
progress	A logical value indicating if a progress bar should be shown. Default is `FALSE`.

Value

A tibble with the metadata and data for each time series <longitude, latitude, start_date, end_date, label, cube, time_series>.

Note

There are four ways of specifying data to be retrieved using the "samples" parameter:

- CSV file: Provide a CSV file with columns "longitude", "latitude", "start_date", "end_date" and "label" for each sample
- SHP file: Provide a shapefile in POINT or POLYGON geometry containing the location of the samples and an attribute to be used as label. Also, provide start and end date for the time series.
- sits object: A sits tibble.
- sf object: An "sf" object with POINT or POLYGON geometry.
- data.frame: A data.frame with with mandatory columns "longitude", "latitude".

Please refer to the sits documentation available in <https://e-sensing.github.io/sitsbook/> for detailed examples.

Author(s)

Gilberto Camara

Examples

```
if (sits_run_examples()) {
    # reading a lat/long from a local cube
    # create a cube from local files
    data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
    raster_cube <- sits_cube(
        source = "BDC",
        collection = "MOD13Q1-6",
        data_dir = data_dir,
        delim = "_",
        parse_info = c("X1", "X2", "tile", "band", "date")
    )
    samples <- tibble::tibble(longitude = -55.66738, latitude = -11.76990)
    point_ndvi <- sits_get_data(raster_cube, samples)
    #
    # reading samples from a cube based on a  CSV file
    csv_file <- system.file("extdata/samples/samples_sinop_crop.csv",
        package = "sits"
    )
    points <- sits_get_data(cube = raster_cube, samples = csv_file)

    # reading a shapefile from BDC (Brazil Data Cube)
    # needs a BDC access key that can be obtained
    # for free by registering in the BDC website
    if (nchar(Sys.getenv("BDC_ACCESS_KEY")) > 0) {
```

```
        # create a data cube from the BDC
        bdc_cube <- sits_cube(
            source = "BDC",
            collection = "CB4_64_16D_STK-1",
            bands = c("NDVI", "EVI"),
            tiles = c("022024", "022025"),
            start_date = "2018-09-01",
            end_date = "2018-10-28"
        )
        # define a shapefile to be read from the cube
        shp_file <- system.file("extdata/shapefiles/bdc-test/samples.shp",
            package = "sits"
        )
        # get samples from the BDC based on the shapefile
        time_series_bdc <- sits_get_data(
            cube = bdc_cube,
            samples = shp_file)
    }
}
```

`sits_impute_linear` *Replace NA values with linear interpolation*

Description

Remove NA by linear interpolation

Usage

```
sits_impute_linear(data = NULL)
```

Arguments

`data`	A time series vector or matrix

Value

A set of filtered time series using the imputation function.

Author(s)

Gilberto Camara, <gilberto.camara@inpe.br>

Examples

```
if (sits_run_examples()) {
    # reading a lat/long from a local cube
    # create a cube from local files
    data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
    raster_cube <- sits_cube(
        source = "BDC",
        collection = "MOD13Q1-6",
        data_dir = data_dir,
        delim = "_",
        parse_info = c("X1", "X2", "tile", "band", "date")
    )
    samples <- tibble::tibble(longitude = -55.66738, latitude = -11.76990)
    point_ndvi <- sits_get_data(
                  cube = raster_cube,
                  samples = samples,
                  impute_fn = sits_impute_linear())
    #
    # reading samples from a cube based on a  CSV file
    csv_file <- system.file("extdata/samples/samples_sinop_crop.csv",
        package = "sits"
    )
    points <- sits_get_data(cube = raster_cube, samples = csv_file)
}
```

`sits_kfold_validate` *Cross-validate time series samples*

Description

Splits the set of time series into training and validation and perform k-fold cross-validation. Cross-validation is a technique for assessing how the results of a statistical analysis will generalize to an independent data set. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform. One round of cross-validation involves partitioning a sample of data into complementary subsets, performing the analysis on one subset (called the training set), and validating the analysis on the other subset (called the validation set or testing set).

The k-fold cross validation method involves splitting the dataset into k-subsets. For each subset is held out while the model is trained on all other subsets. This process is completed until accuracy is determine for each instance in the dataset, and an overall accuracy estimate is provided.

This function returns the confusion matrix, and Kappa values.

Usage

```
sits_kfold_validate(
  samples,
  folds = 5,
  ml_method = sits_rfor(),
```

```
  multicores = 2
)
```

Arguments

samples	Time series.
folds	Number of partitions to create.
ml_method	Machine learning method.
multicores	Number of cores to process in parallel.

Value

A caret::confusionMatrix object to be used for validation assessment.

Note

Please refer to the sits documentation available in <https://e-sensing.github.io/sitsbook/> for detailed examples.

Author(s)

Rolf Simoes, <rolf.simoes@inpe.br>

Gilberto Camara, <gilberto.camara@inpe.br>

Examples

```
if (sits_run_examples()) {
    # A dataset containing a tibble with time series samples
    # for the Mato Grosso state in Brasil
    # create a list to store the results
    results <- list()

    # accuracy assessment lightTAE
    acc_ltae <- sits_kfold_validate(samples_modis_4bands,
        folds = 5,
        ml_method = sits_lighttae()
    )
    # use a name
    acc_ltae$name <- "LightTAE"
    # put the result in a list
    results[[length(results) + 1]] <- acc_ltae

    # Deep Learning - ResNet
    acc_rn <- sits_kfold_validate(samples_modis_4bands,
        folds = 5,
        ml_method = sits_resnet()
    )
    acc_rn$name <- "ResNet"
    # put the result in a list
    results[[length(results) + 1]] <- acc_rn
```

```
    # save to xlsx file
    sits_to_xlsx(results, file = "./accuracy_mato_grosso_dl.xlsx")
}
```

---

sits_labels *Get labels associated to a data set*

Description

Finds labels in a sits tibble or data cube

Usage

```
sits_labels(data)

## S3 method for class 'sits'
sits_labels(data)

## S3 method for class 'sits_cube'
sits_labels(data)

## S3 method for class 'patterns'
sits_labels(data)

## S3 method for class 'sits_model'
sits_labels(data)
```

Arguments

data	Time series or a cube.

Value

The labels associated to a set of time series or to a data cube.

Author(s)

Rolf Simoes, <rolf.simoes@inpe.br>

Examples

```
# read a tibble with 400 samples of Cerrado and 346 samples of Pasture
data(cerrado_2classes)
# print the labels
sits_labels(cerrado_2classes)
```

sits_labels_summary *Inform label distribution of a set of time series*

Description

Describes labels in a sits tibble

Usage

```
sits_labels_summary(data)

## S3 method for class 'sits'
sits_labels_summary(data)
```

Arguments

data	Valid sits tibble

Value

A tibble with the frequency of each label.

Author(s)

Rolf Simoes, <rolf.simoes@inpe.br>

Examples

```
# read a tibble with 400 samples of Cerrado and 346 samples of Pasture
data(cerrado_2classes)
# print the labels
sits_labels_summary(cerrado_2classes)
```

sits_label_classification *Build a labelled image from a probability cube*

Description

Takes a set of classified raster layers with probabilities, and label them based on the maximum probability for each pixel.

Usage

```
sits_label_classification(
  cube,
  multicores = 2,
  memsize = 4,
  output_dir = ".",
  version = "v1"
)
```

Arguments

cube	Classified image data cube.
multicores	Number of workers to label the classification in parallel.
memsize	maximum overall memory (in GB) to label the classification.
output_dir	Output directory for classified files.
version	Version of resulting image (in the case of multiple runs).

Value

A data cube with an image with the classified map.

Note

Please refer to the sits documentation available in <https://e-sensing.github.io/sitsbook/> for detailed examples.

Author(s)

Rolf Simoes, <rolf.simoes@inpe.br>

Examples

```
if (sits_run_examples()) {
    # select a set of samples
    samples_ndvi <- sits_select(samples_modis_4bands, bands = c("NDVI"))
    # create a random forest model
    rfor_model <- sits_train(samples_ndvi, sits_rfor())
    # create a data cube from local files
    data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
    cube <- sits_cube(
        source = "BDC",
        collection = "MOD13Q1-6",
        data_dir = data_dir,
        delim = "_",
        parse_info = c("X1", "X2", "tile", "band", "date")
    )
    # classify a data cube
    probs_cube <- sits_classify(data = cube, ml_model = rfor_model)
    # plot the probability cube
```

```
    plot(probs_cube)
    # smooth the probability cube using Bayesian statistics
    bayes_cube <- sits_smooth(probs_cube)
    # plot the smoothed cube
    plot(bayes_cube)
    # label the probability cube
    label_cube <- sits_label_classification(bayes_cube)
    # plot the labelled cube
    plot(label_cube)
}
```

`sits_lighttae` *Train a model using Lightweight Temporal Self-Attention Encoder*

Description

Implementation of Light Temporal Attention Encoder (L-TAE) for satellite image time series

This function is based on the paper by Vivien Garnot referenced below and code available on github at https://github.com/VSainteuf/lightweight-temporal-attention-pytorch If you use this method, please cite the original TAE and the LTAE paper.

We also used the code made available by Maja Schneider in her work with Marco Körner referenced below and available at https://github.com/maja601/RC2020-psetae.

Usage

```
sits_lighttae(
  samples = NULL,
  samples_validation = NULL,
  epochs = 150,
  batch_size = 128,
  validation_split = 0.2,
  optimizer = torchopt::optim_adamw,
  opt_hparams = list(lr = 0.005, eps = 1e-08, weight_decay = 1e-06),
  lr_decay_epochs = 50,
  lr_decay_rate = 1,
  patience = 20,
  min_delta = 0.01,
  verbose = FALSE
)
```

Arguments

`samples` Time series with the training samples.

`samples_validation`
Time series with the validation samples. if the `samples_validation` parameter is provided, the `validation_split` parameter is ignored.

`epochs` Number of iterations to train the model.

batch_size	Number of samples per gradient update.
validation_split	Fraction of training data to be used as validation data.
optimizer	Optimizer function to be used.
opt_hparams	Hyperparameters for optimizer: lr : Learning rate of the optimizer eps: Term added to the denominator to improve numerical stability. weight_decay: L2 regularization
lr_decay_epochs	Number of epochs to reduce learning rate.
lr_decay_rate	Decay factor for reducing learning rate.
patience	Number of epochs without improvements until training stops.
min_delta	Minimum improvement in loss function to reset the patience counter.
verbose	Verbosity mode (TRUE/FALSE). Default is FALSE.

Value

A fitted model to be used for classification of data cubes.

Note

Please refer to the sits documentation available in <https://e-sensing.github.io/sitsbook/> for detailed examples.

Author(s)

Charlotte Pelletier, <charlotte.pelletier@univ-ubs.fr>

Gilberto Camara, <gilberto.camara@inpe.br>

Rolf Simoes, <rolf.simoes@inpe.br>

References

Vivien Garnot, Loic Landrieu, Sebastien Giordano, and Nesrine Chehata, "Satellite Image Time Series Classification with Pixel-Set Encoders and Temporal Self-Attention", 2020 Conference on Computer Vision and Pattern Recognition. pages 12322-12331. DOI: 10.1109/CVPR42600.2020.01234

Vivien Garnot, Loic Landrieu, "Lightweight Temporal Self-Attention for Classifying Satellite Images Time Series", arXiv preprint arXiv:2007.00586, 2020.

Schneider, Maja; Körner, Marco, "[Re] Satellite Image Time Series Classification with Pixel-Set Encoders and Temporal Self-Attention." ReScience C 7 (2), 2021. DOI: 10.5281/zenodo.4835356

Examples

```
if (sits_run_examples()) {
    # select a set of samples
    samples_ndvi <- sits_select(samples_modis_4bands, bands = c("NDVI"))
    # create a lightTAE model
    torch_model <- sits_train(samples_ndvi, sits_lighttae())
```

```
    # plot the model
    plot(torch_model)
    # create a data cube from local files
    data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
    cube <- sits_cube(
        source = "BDC",
        collection = "MOD13Q1-6",
        data_dir = data_dir,
        delim = "_",
        parse_info = c("X1", "X2", "tile", "band", "date")
    )
    # classify a data cube
    probs_cube <- sits_classify(data = cube, ml_model = torch_model)
    # plot the probability cube
    plot(probs_cube)
    # smooth the probability cube using Bayesian statistics
    bayes_cube <- sits_smooth(probs_cube)
    # plot the smoothed cube
    plot(bayes_cube)
    # label the probability cube
    label_cube <- sits_label_classification(bayes_cube)
    # plot the labelled cube
    plot(label_cube)
}
```

sits_merge — *Merge two data sets (time series or cubes)*

Description

To merge two series, we consider that they contain different attributes but refer to the same data cube, and spatiotemporal location. This function is useful to merge different bands of the same locations. For example, one may want to put the raw and smoothed bands for the same set of locations in the same tibble.

To merge data cubes, they should share the same sensor, resolution, bounding box, timeline, and have different bands.

Usage

```
sits_merge(data1, data2, ..., suffix = c(".1", ".2"))

## S3 method for class 'sits'
sits_merge(data1, data2, ..., suffix = c(".1", ".2"))

## S3 method for class 'raster_cube'
sits_merge(data1, data2, ..., suffix = c(".1", ".2"))
```

Arguments

data1	Time series or cube to be merged.
data2	Time series or cube to be merged.
...	Additional parameters
suffix	If there are duplicate bands in data1 and data2 these suffixes will be added.

Value

merged data sets

Author(s)

Gilberto Camara, <gilberto.camara@inpe.br>

Examples

```
if (sits_run_examples()) {
    # Retrieve a time series with values of NDVI
    point_ndvi <- sits_select(point_mt_6bands, bands = "NDVI")

    # Filter the point using the Whittaker smoother
    point_whit <- sits_filter(point_ndvi, sits_whittaker(lambda = 3.0))
    # Merge time series
    point_ndvi <- sits_merge(point_ndvi, point_whit, suffix = c("", ".WHIT"))

    # Plot the two points to see the smoothing effect
    plot(point_ndvi)
}
```

---

sits_mixture_model *Multiple endmember spectral mixture analysis*

Description

Create a multiple endmember spectral mixture analyses fractions images. To calculate the fraction of each endmember, the non-negative least squares (NNLS) solver is used. The NNLS implementation was made by Jakob Schwalb-Willmann in RStoolbox package (licensed as GPL>=3).

Usage

```
sits_mixture_model(
  cube,
  endmembers_spectra,
  memsize = 1,
  multicores = 2,
  output_dir = getwd(),
  rmse_band = TRUE,
```

```
  remove_outliers = TRUE,
  progress = TRUE
)
```

Arguments

- `cube` — A sits data cube.
- `endmembers_spectra` — Reference endmembers spectra in a tibble format. (see details below).
- `memsize` — Memory available for mixture model (in GB).
- `multicores` — Number of cores to be used for generate the mixture model.
- `output_dir` — Directory for output file.
- `rmse_band` — A boolean indicating whether the error associated with the linear model should be generated. If true, a new band with the errors for each pixel is generated using the root mean square measure (RMSE). Default is TRUE.
- `remove_outliers` — A boolean indicating whether values larger and smaller than the limits in the image metadata, and missing values should be marked as NA. This parameter can be used when the cloud component is added to the mixture model. Default is TRUE.
- `progress` — Show progress bar? Default is TRUE.

Value

a sits cube with the generated fractions.

Note

The `endmembers_spectra` parameter should be a tibble, csv or a shapefile. `endmembers_spectra` must have the following columns: `type`, which defines the endmembers that will be created and the columns corresponding to the bands that will be used in the mixture model.

Author(s)

Felipe Carvalho, <felipe.carvalho@inpe.br>

Felipe Carlos, <efelipecarlos@gmail.com>

Rolf Simoes, <rolf.simoes@inpe.br>

Alber Sanchez, <alber.ipia@inpe.br>

References

`RStoolbox` package (https://github.com/bleutner/RStoolbox/)

Examples

```
if (sits_run_examples()) {
   # --- Create a cube based on a local MODIS data
   data_dir <- system.file("extdata/raster/mod13q1", package = "sits")

   modis_cube <- sits_cube(
       source = "BDC",
       collection = "MOD13Q1-6",
       data_dir = data_dir,
       delim = "_"
   )

   endmembers_spectra <- tibble::tibble(
       type = c("vegetation", "not-vegetation"),
       NDVI = c(8500, 3400)
   )

   mixture_cube <- sits_mixture_model(
       cube = modis_cube,
       endmembers_spectra = endmembers_spectra,
       memsize = 4,
       multicores = 2,
       output_dir = tempdir()
   )
}
```

---

sits_mlp *Train multi-layer perceptron models using torch*

Description

Use a multi-layer perceptron algorithm to classify data. This function uses the R "torch" and "luz" packages. Please refer to the documentation of those package for more details.

Usage

```
sits_mlp(
  samples = NULL,
  samples_validation = NULL,
  layers = c(512, 512, 512),
  dropout_rates = c(0.2, 0.3, 0.4),
  optimizer = torchopt::optim_adamw,
  opt_hparams = list(lr = 0.001, eps = 1e-08, weight_decay = 1e-06),
  epochs = 100,
  batch_size = 64,
  validation_split = 0.2,
  patience = 20,
```

```
  min_delta = 0.01,
  verbose = FALSE
)
```

Arguments

samples — Time series with the training samples.

samples_validation — Time series with the validation samples. if the samples_validation parameter is provided, the validation_split parameter is ignored.

layers — Vector with number of hidden nodes in each layer.

dropout_rates — Vector with the dropout rates (0,1) for each layer.

optimizer — Optimizer function to be used.

opt_hparams — Hyperparameters for optimizer: lr : Learning rate of the optimizer eps: Term added to the denominator to improve numerical stability.. weight_decay: L2 regularization

epochs — Number of iterations to train the model.

batch_size — Number of samples per gradient update.

validation_split — Number between 0 and 1. Fraction of the training data for validation. The model will set apart this fraction and will evaluate the loss and any model metrics on this data at the end of each epoch.

patience — Number of epochs without improvements until training stops.

min_delta — Minimum improvement in loss function to reset the patience counter.

verbose — Verbosity mode (TRUE/FALSE). Default is FALSE.

Value

A torch mlp model to be used for classification.

Note

The parameters for the MLP have been chosen based on the work by Wang et al. 2017 that takes multilayer perceptrons as the baseline for time series classifications: (a) Three layers with 512 neurons each, specified by the parameter 'layers'; (b) dropout rates of 10 (c) the "optimizer_adam" as optimizer (default value); (d) a number of training steps ('epochs') of 100; (e) a 'batch_size' of 64, which indicates how many time series are used for input at a given steps; (f) a validation percentage of 20 will be randomly set side for validation. (g) The "relu" activation function.

#' @references

Zhiguang Wang, Weizhong Yan, and Tim Oates, "Time series classification from scratch with deep neural networks: A strong baseline", 2017 international joint conference on neural networks (IJCNN).

Please refer to the sits documentation available in <https://e-sensing.github.io/sitsbook/> for detailed examples.

Author(s)

Gilberto Camara, <gilberto.camara@inpe.br>

Rolf Simoes, <rolf.simoes@inpe.br>

Felipe Souza, <lipecaso@gmail.com>

Alber Sanchez, <alber.ipia@inpe.br>

Examples

```
if (sits_run_examples()) {
    # select a set of samples
    samples_ndvi <- sits_select(samples_modis_4bands, bands = c("NDVI"))
    # create an MLP model
    torch_model <- sits_train(samples_ndvi, sits_mlp())
    # plot the model
    plot(torch_model)
    # create a data cube from local files
    data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
    cube <- sits_cube(
        source = "BDC",
        collection = "MOD13Q1-6",
        data_dir = data_dir,
        delim = "_",
        parse_info = c("X1", "X2", "tile", "band", "date")
    )
    # classify a data cube
    probs_cube <- sits_classify(data = cube, ml_model = torch_model)
    # plot the probability cube
    plot(probs_cube)
    # smooth the probability cube using Bayesian statistics
    bayes_cube <- sits_smooth(probs_cube)
    # plot the smoothed cube
    plot(bayes_cube)
    # label the probability cube
    label_cube <- sits_label_classification(bayes_cube)
    # plot the labelled cube
    plot(label_cube)
}
```

---

sits_patterns *Find temporal patterns associated to a set of time series*

Description

This function takes a set of time series samples as input estimates a set of patterns. The patterns are calculated using a GAM model. The idea is to use a formula of type y ~ s(x), where x is a temporal reference and y if the value of the signal. For each time, there will be as many predictions as there are sample values. The GAM model predicts a suitable approximation that fits the assumptions of the statistical model, based on a smooth function.

This method is based on the "createPatterns" method of the dtwSat package, which is also described in the reference paper.

Usage

```
sits_patterns(data = NULL, freq = 8, formula = y ~ s(x), ...)
```

Arguments

`data`	Time series.
`freq`	Interval in days for estimates.
`formula`	Formula to be applied in the estimate.
`...`	Any additional parameters.

Value

Time series with patterns.

Note

Please refer to the sits documentation available in <https://e-sensing.github.io/sitsbook/> for detailed examples.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

Gilberto Camara, <gilberto.camara@inpe.br>

Rolf Simoes, <rolf.simoes@inpe.br>

References

Maus V, Camara G, Cartaxo R, Sanchez A, Ramos F, Queiroz GR. A Time-Weighted Dynamic Time Warping Method for Land-Use and Land-Cover Mapping. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 9(8):3729-3739, August 2016. ISSN 1939-1404. doi:10.1109/JSTARS.2016.2517118.

Examples

```
if (sits_run_examples()) {
   patterns <- sits_patterns(cerrado_2classes)
   plot(patterns)
}
```

`sits_reduce_imbalance` *Reduce imbalance in a set of samples*

Description

Takes a sits tibble with different labels and returns a new tibble. Deals with class imbalance using the synthetic minority oversampling technique (SMOTE) for oversampling. Undersampling is done using the SOM methods available in the sits package.

Usage

```
sits_reduce_imbalance(
  samples,
  n_samples_over = 200,
  n_samples_under = 400,
  multicores = 2
)
```

Arguments

`samples`	Sample set to rebalance
`n_samples_over`	Number of samples to oversample for classes with samples less than this number (use n_samples_over = NULL to avoid oversampling).
`n_samples_under`	Number of samples to undersample for classes with samples more than this number (use n_samples_over = NULL to avoid oversampling).
`multicores`	Number of cores to process the data (default 2).

Value

A sits tibble with reduced sample imbalance.

Note

Please refer to the sits documentation available in <https://e-sensing.github.io/sitsbook/> for detailed examples.

Author(s)

Gilberto Camara, <gilberto.camara@inpe.br>

References

The reference paper on SMOTE is N. V. Chawla, K. W. Bowyer, L. O.Hall, W. P. Kegelmeyer, “SMOTE: synthetic minority over-sampling technique,” Journal of artificial intelligence research, 321-357, 2002.

Undersampling uses the SOM map developed by Lorena Santos and co-workers and used in the sits_som_map() function. The SOM map technique is described in the paper: Lorena Santos, Karine Ferreira, Gilberto Camara, Michelle Picoli, Rolf Simoes, "Quality control and class noise reduction of satellite image time series". ISPRS Journal of Photogrammetry and Remote Sensing, vol. 177, pp 75-88, 2021. https://doi.org/10.1016/j.isprsjprs.2021.04.014.

Examples

```
if (sits_run_examples()) {
    # print the labels summary for a sample set
    sits_labels_summary(samples_modis_4bands)
    # reduce the sample imbalance
    new_samples <- sits_reduce_imbalance(samples_modis_4bands,
        n_samples_over = 200,
        n_samples_under = 200,
        multicores = 1
    )
    # print the labels summary for the rebalanced set
    sits_labels_summary(new_samples)
}
```

---

sits_regularize — *Build a regular data cube from an irregular one*

Description

Produces regular data cubes for analysis-ready data (ARD) image collections. Analysis-ready data (ARD) collections available in AWS, MPC, USGS and DEAfrica are not regular in space and time. Bands may have different resolutions, images may not cover the entire time, and time intervals are not regular. For this reason, subsets of these collection need to be converted to regular data cubes before further processing and data analysis.

This function requires users to include the cloud band in their ARD-based data cubes.

Usage

```
sits_regularize(
  cube,
  period,
  res,
  roi = NULL,
  output_dir,
  multicores = 1,
  memsize = 4,
  progress = TRUE
)
```

Arguments

cube	`sits_cube` object whose observation period and/or spatial resolution is not constant.
period	ISO8601-compliant time period for regular data cubes, with number and unit, where "D", "M" and "Y" stand for days, month and year; e.g., "P16D" for 16 days.
res	Spatial resolution of regularized images (in meters).
roi	A named `numeric` vector with a region of interest. See more above.
output_dir	Valid directory for storing regularized images.
multicores	Number of cores used for regularization; used for parallel processing of input.
memsize	Memory available for regularization (in GB).
progress	show progress bar?

Value

A `sits_cube` object with aggregated images.

Note

Please refer to the sits documentation available in <https://e-sensing.github.io/sitsbook/> for detailed examples.

The "roi" parameter defines a region of interest. It can be an sf_object, a shapefile, or a bounding box vector with named XY values ("xmin", "xmax", "ymin", "ymax") or named lat/long values ("lat_min", "lat_max", "long_min", "long_max"). The `sits_regularize` function will crop the images that contain the roi region.

The aggregation method used in `sits_regularize` sorts the images based on cloud cover, where images with the fewest clouds at the top of the stack. Once the stack of images is sorted, the method uses the first valid value to create the temporal aggregation.

The input (non-regular) ARD cube needs to include the cloud band for the regularization to work.

References

Appel, Marius; Pebesma, Edzer. On-demand processing of data cubes from satellite image collections with the gdalcubes library. Data, v. 4, n. 3, p. 92, 2019. DOI: 10.3390/data4030092.

Examples

```
if (sits_run_examples()) {
    # define a non-regular Sentinel-2 cube in AWS
    s2_cube_open <- sits_cube(
        source = "AWS",
        collection = "SENTINEL-S2-L2A-COGS",
        tiles = c("20LKP", "20LLP"),
        bands = c("B8A", "SCL"),
        start_date = "2018-10-01",
        end_date = "2018-11-01"
```

```
    )
    # create a directory to store the regularized images
    dir_images <- paste0(".", "/images_regcube/")
    if (!dir.exists(dir_images)) {
        dir.create(dir_images)
    }
    # regularize the cube
    rg_cube <- sits_regularize(
        cube = s2_cube_open,
        output_dir = dir_images,
        res = 60,
        period = "P16D",
        multicores = 2,
        memsize = 16
    )
}
```

`sits_resnet` *Train ResNet classification models*

Description

Use a ResNet architecture for classifying image time series. The ResNet (or deep residual network) was proposed by a team in Microsoft Research for 2D image classification. ResNet tries to address the degradation of accuracy in a deep network. The idea is to replace a deep network with a combination of shallow ones. In the paper by Fawaz et al. (2019), ResNet was considered the best method for time series classification, using the UCR dataset. Please refer to the paper for more details.

The R-torch version is based on the code made available by Zhiguang Wang, author of the original paper. The code was developed in python using keras.

https://github.com/cauchyturing (repo: UCR_Time_Series_Classification_Deep_Learning_Baseline)

The R-torch version also considered the code by Ignacio Oguiza, whose implementation is available at https://github.com/timeseriesAI/tsai/blob/main/tsai/models/ResNet.py.

There are differences between Wang's Keras code and Oguiza torch code. In this case, we have used Wang's keras code as the main reference.

Usage

```
sits_resnet(
  samples = NULL,
  samples_validation = NULL,
  blocks = c(64, 128, 128),
  kernels = c(7, 5, 3),
  epochs = 100,
  batch_size = 64,
  validation_split = 0.2,
  optimizer = torchopt::optim_adamw,
```

```
  opt_hparams = list(lr = 0.001, eps = 1e-08, weight_decay = 1e-06),
  lr_decay_epochs = 1,
  lr_decay_rate = 0.95,
  patience = 20,
  min_delta = 0.01,
  verbose = FALSE
)
```

Arguments

- `samples` — Time series with the training samples.
- `samples_validation` — Time series with the validation samples. if the `samples_validation` parameter is provided, the `validation_split` parameter is ignored.
- `blocks` — Number of 1D convolutional filters for each block of three layers.
- `kernels` — Size of the 1D convolutional kernels
- `epochs` — Number of iterations to train the model. for each layer of each block.
- `batch_size` — Number of samples per gradient update.
- `validation_split` — Fraction of training data to be used as validation data.
- `optimizer` — Optimizer function to be used.
- `opt_hparams` — Hyperparameters for optimizer: lr : Learning rate of the optimizer eps: Term added to the denominator to improve numerical stability. weight_decay: L2 regularization
- `lr_decay_epochs` — Number of epochs to reduce learning rate.
- `lr_decay_rate` — Decay factor for reducing learning rate.
- `patience` — Number of epochs without improvements until training stops.
- `min_delta` — Minimum improvement in loss function to reset the patience counter.
- `verbose` — Verbosity mode (TRUE/FALSE). Default is FALSE.

Value

A fitted model to be used for classification.

Note

Please refer to the sits documentation available in <https://e-sensing.github.io/sitsbook/> for detailed examples.

Author(s)

Gilberto Camara, `<gilberto.camara@inpe.br>`

Rolf Simoes, `<rolf.simoes@inpe.br>`

Felipe Souza, `<lipecaso@gmail.com>`

Alber Sanchez, <alber.ipia@inpe.br>

Charlotte Pelletier, <charlotte.pelletier@univ-ubs.fr>

Daniel Falbel, <dfalbel@gmail.com>

References

Hassan Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller, "Deep learning for time series classification: a review", Data Mining and Knowledge Discovery, 33(4): 917–963, 2019.

Zhiguang Wang, Weizhong Yan, and Tim Oates, "Time series classification from scratch with deep neural networks: A strong baseline", 2017 international joint conference on neural networks (IJCNN).

Examples

```
if (sits_run_examples()) {
    # select a set of samples
    samples_ndvi <- sits_select(samples_modis_4bands, bands = c("NDVI"))
    # create a ResNet model
    torch_model <- sits_train(samples_ndvi, sits_resnet())
    # plot the model
    plot(torch_model)
    # create a data cube from local files
    data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
    cube <- sits_cube(
        source = "BDC",
        collection = "MOD13Q1-6",
        data_dir = data_dir,
        delim = "_",
        parse_info = c("X1", "X2", "tile", "band", "date")
    )
    # classify a data cube
    probs_cube <- sits_classify(data = cube, ml_model = torch_model)
    # plot the probability cube
    plot(probs_cube)
    # smooth the probability cube using Bayesian statistics
    bayes_cube <- sits_smooth(probs_cube)
    # plot the smoothed cube
    plot(bayes_cube)
    # label the probability cube
    label_cube <- sits_label_classification(bayes_cube)
    # plot the labelled cube
    plot(label_cube)
}
```

sits_rfor *Train random forest models*

Description

Use Random Forest algorithm to classify samples. This function is a front-end to the "randomForest" package. Please refer to the documentation in that package for more details.

Usage

```
sits_rfor(samples = NULL, num_trees = 120, mtry = NULL, ...)
```

Arguments

`samples`	Time series with the training samples.
`num_trees`	Number of trees to grow. This should not be set to too small a number, to ensure that every input row gets predicted at least a few times (default: 120).
`mtry`	Number of variables randomly sampled as candidates at each split (default: NULL - use default value of `randomForest::randomForest()` function, i.e. `floor(sqrt(features))`).
`...`	Other parameters to be passed to ‘randomForest::randomForest‘ function.

Value

Model fitted to input data (to be passed to `sits_classify`).

Note

Please refer to the sits documentation available in <https://e-sensing.github.io/sitsbook/> for detailed examples.

Author(s)

Alexandre Ywata de Carvalho, `<alexandre.ywata@ipea.gov.br>`

Rolf Simoes, `<rolf.simoes@inpe.br>`

Gilberto Camara, `<gilberto.camara@inpe.br>`

Examples

```
if (sits_run_examples()) {
    # Example of training a model for time series classification
    # Retrieve the samples for Mato Grosso
    # train a random forest model
    rf_model <- sits_train(samples_modis_4bands,
                           ml_method = sits_rfor(mtry = 20))
    # select the bands to classify the point
    sample_bands <- sits_bands(samples_modis_4bands)
```

```
    point_4bands <- sits_select(point_mt_6bands, bands = sample_bands)
    # classify the point
    point_class <- sits_classify(point_4bands, rf_model)
    plot(point_class)
}
```

sits_run_examples *Informs if sits examples should run*

Description

This function informs if sits examples should run. This is useful to avoid running slow examples in CRAN environment.

Usage

```
sits_run_examples()
```

Value

A logical value

Examples

```
if (sits_run_examples()) {
# set examples to FALSE
sits_config(run_examples = FALSE)
isFALSE(sits_run_examples())
# recover config state
sits_config(run_examples = TRUE)
}
```

sits_run_tests *Informs if sits tests should run*

Description

This function informs if sits test should run. Useful to avoid running slow tests in CRAN environment. Behaviour controlled by environmental variable R_CONFIG_ACTIVE_TESTS

Usage

```
sits_run_tests()
```

Value

TRUE/FALSE

Examples

```
if (sits_run_examples()) {
# recover config state
config_tests <- sits_run_tests()
# set active tests to FALSE
sits_config(run_tests = FALSE)
isFALSE(sits_run_tests())
# recover config state
# set active tests
sits_config(run_tests = TRUE)
# result should be true
isTRUE(sits_run_tests())
# restore previous state
sits_config(run_tests = config_tests)
}
```

---

sits_sample *Sample a percentage of a time series*

Description

Takes a sits tibble with different labels and returns a new tibble. For a given field as a group criterion, this new tibble contains a given number or percentage of the total number of samples per group. Parameter n: number of random samples. Parameter frac: a fraction of random samples. If n is greater than the number of samples for a given label, that label will be sampled with replacement. Also, if frac > 1 , all sampling will be done with replacement.

Usage

```
sits_sample(data, n = NULL, frac = NULL, oversample = TRUE)
```

Arguments

`data`	Input sits tibble.
`n`	Number of samples to pick from each group of data.
`frac`	Percentage of samples to pick from each group of data.
`oversample`	Oversample classes with small number of samples?

Value

A sits tibble with a fixed quantity of samples.

Author(s)

Rolf Simoes, <rolf.simoes@inpe.br>

Examples

```
# Retrieve a set of time series with 2 classes
data(cerrado_2classes)
# Print the labels of the resulting tibble
sits_labels(cerrado_2classes)
# Samples the data set
data <- sits_sample(cerrado_2classes, n = 10)
# Print the labels of the resulting tibble
sits_labels(data)
```

`sits_select` *Filter bands on a data set (tibble or cube)*

Description

Filter only the selected bands from a tibble or a data cube.

Usage

```
sits_select(data, bands, ...)

## S3 method for class 'sits'
sits_select(data, bands, ...)

## S3 method for class 'sits_cube'
sits_select(data, bands, ..., tiles = NULL)

## S3 method for class 'patterns'
sits_select(data, bands, ...)
```

Arguments

data	A sits tibble or data cube.
bands	Character vector with the names of the bands.
...	Additional parameters to be provided in the select function.
tiles	Character vector with the names of the tiles.

Value

For sits tibble, returns a sits tibble with the selected bands. For data cube, a data cube with the selected bands.

Author(s)

Rolf Simoes, <rolf.simoes@inpe.br>

Examples

```
# Retrieve a set of time series with 2 classes
data(cerrado_2classes)
# Print the original bands
sits_bands(cerrado_2classes)
# Select only the NDVI band
data <- sits_select(cerrado_2classes, bands = c("NDVI"))
# Print the labels of the resulting tibble
sits_bands(data)
```

---

sits_smooth *Smooth probability cubes with spatial predictors*

Description

Takes a set of classified raster layers with probabilities, whose metadata is]created by sits_cube, and applies a smoothing function. There are three options, defined by the "type" parameter:

- "bayes": Use a bayesian smoother
- "bilateral: Use a bilateral smoother

Usage

```
sits_smooth(cube, type = "bayes", ...)

## S3 method for class 'bayes'
sits_smooth(
  cube,
  type = "bayes",
  ...,
  window_size = 5,
  smoothness = 20,
  covar = FALSE,
  multicores = 2,
  memsize = 4,
  output_dir = ".",
  version = "v1"
)

## S3 method for class 'bilateral'
sits_smooth(
  cube,
```

```
    type = "bilateral",
    ...,
    window_size = 5,
    sigma = 8,
    tau = 0.1,
    multicores = 2,
    memsize = 4,
    output_dir = ".",
    version = "v1"
  )
```

Arguments

cube	Probability data cube
type	Type of smoothing
...	Parameters for specific functions
window_size	Size of the neighbourhood.
smoothness	Estimated variance of logit of class probabilities (Bayesian smoothing parameter). It can be either a matrix or a scalar.
covar	a logical argument indicating if a covariance matrix must be computed as the prior covariance for bayesian smoothing.
multicores	Number of cores to run the smoothing function
memsize	Maximum overall memory (in GB) to run the smoothing.
output_dir	Output directory for image files
version	Version of resulting image (in the case of multiple tests)
sigma	Standard deviation of the spatial Gaussian kernel (for bilateral smoothing)
tau	Standard deviation of the class probs value (for bilateral smoothing)

Value

A tibble with metadata about the output raster objects.

Note

Please refer to the sits documentation available in <https://e-sensing.github.io/sitsbook/> for detailed examples.

Author(s)

Gilberto Camara, <gilberto.camara@inpe.br>

Rolf Simoes, <rolf.simoes@inpe.br>

References

K. Schindler, "An Overview and Comparison of Smooth Labeling Methods for Land-Cover Classification", IEEE Transactions on Geoscience and Remote Sensing, 50 (11), 4534-4545, 2012 (for gaussian and bilateral smoothing)

Examples

```
if (sits_run_examples()) {
    # select a set of samples
    samples_ndvi <- sits_select(samples_modis_4bands, bands = c("NDVI"))
    # create a ResNet model
    torch_model <- sits_train(samples_ndvi, sits_resnet())
    # plot the model
    plot(torch_model)
    # create a data cube from local files
    data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
    cube <- sits_cube(
        source = "BDC",
        collection = "MOD13Q1-6",
        data_dir = data_dir,
        delim = "_",
        parse_info = c("X1", "X2", "tile", "band", "date")
    )
    # classify a data cube
    probs_cube <- sits_classify(data = cube, ml_model = torch_model)
    # plot the probability cube
    plot(probs_cube)
    # smooth the probability cube using Bayesian statistics
    bayes_cube <- sits_smooth(probs_cube)
    # plot the smoothed cube
    plot(bayes_cube)
    # label the probability cube
    label_cube <- sits_label_classification(bayes_cube)
    # plot the labelled cube
    plot(label_cube)
}
```

---

sits_som   *Use SOM for quality analysis of time series samples*

Description

These function use self-organized maps to perform quality analysis in satellite image time series

`sits_som_map()` creates a SOM map, where high-dimensional data is mapped into a two dimensional map, keeping the topological relations between data patterns. Each sample is assigned to a neuron, and neurons are placed in the grid based on similarity.

`sits_som_evaluate_cluster()` analyses the neurons of the SOM map, and builds clusters based on them. Each cluster is a neuron or a set of neuron categorized with same label. It produces a tibble with the percentage of mixture of classes in each cluster.

`sits_som_clean_samples()` evaluates the quality of the samples based on the results of the SOM map. The algorithm identifies noisy samples, using ‘prior_threshold‘ for the prior probability and ‘posterior_threshold‘ for the posterior probability. Each sample receives an evaluation tag, according to the following rule: (a) If the prior probability is < ‘prior_threshold‘, the sample is tagged as "remove"; (b) If the prior probability is >= ‘prior_threshold‘ and the posterior probability is

>=‘posterior_threshold‘, the sample is tagged as "clean"; (c) If the prior probability is >= ‘posterior_threshold‘ and the posterior probability is < ‘posterior_threshold‘, the sample is tagged as "analyze" for further inspection. The user can define which tagged samples will be returned using the "keep" parameter, with the following options: "clean", "analyze", "remove".

Usage

```
sits_som_map(
  data,
  grid_xdim = 10,
  grid_ydim = 10,
  alpha = 1,
  rlen = 100,
  distance = "euclidean",
  som_radius = 2,
  mode = "online"
)

sits_som_clean_samples(
  som_map,
  prior_threshold = 0.6,
  posterior_threshold = 0.6,
  keep = c("clean", "analyze")
)

sits_som_evaluate_cluster(som_map)
```

Arguments

`data`	A tibble with samples to be clustered.
`grid_xdim`	X dimension of the SOM grid (default = 25).
`grid_ydim`	Y dimension of the SOM grid.
`alpha`	Starting learning rate (decreases according to number of iterations).
`rlen`	Number of iterations to produce the SOM.
`distance`	The type of similarity measure (distance).
`som_radius`	Radius of SOM neighborhood.
`mode`	Type of learning algorithm (default = "online").
`som_map`	Object returned by `sits_som_map`.
`prior_threshold`	Threshold of conditional probability (frequency of samples assigned to the same SOM neuron).
`posterior_threshold`	Threshold of posterior probability (influenced by the SOM neighborhood).
`keep`	Which types of evaluation to be maintained in the data.

Value

`sits_som_map()` produces a list with three members: (1) the samples tibble, with one additional column indicating to which neuron each sample has been mapped; (2) the Kohonen map, used for plotting and cluster quality measures; (3) a tibble with the labelled neurons, where each class of each neuron is associated to two values: (a) the prior probability that this class belongs to a cluster based on the frequency of samples of this class allocated to the neuron; (b) the posterior probability that this class belongs to a cluster, using data for the neighbours on the SOM map.

`sits_som_clean_samples()` produces a sits tibble with an two additional columns.The first indicates if each sample is clean, should be analyzed or should be removed. The second indicates the posterior probability of the sample

`sits_som_evaluate_cluster()` produces a tibble with the clusters found by the SOM map. For each cluster, ir provides the percentage of classes inside it.

Note

Please refer to the sits documentation available in <https://e-sensing.github.io/sitsbook/> for detailed examples.

Author(s)

Lorena Alves, `<lorena.santos@inpe.br>`

Karine Ferreira. `<karine.ferreira@inpe.br>`

References

Lorena Santos, Karine Ferreira, Gilberto Camara, Michelle Picoli, Rolf Simoes, “Quality control and class noise reduction of satellite image time series”. ISPRS Journal of Photogrammetry and Remote Sensing, vol. 177, pp 75-88, 2021. https://doi.org/10.1016/j.isprsjprs.2021.04.014.

Examples

```
if (sits_run_examples()) {
    # create a som map
    som_map <- sits_som_map(samples_modis_4bands)
    # plot the som map
    plot(som_map)
    # evaluate the som map and create clusters
    clusters_som <- sits_som_evaluate_cluster(som_map)
    # plot the cluster evaluation
    plot(clusters_som)
    # clean the samples
    new_samples <- sits_som_clean_samples(som_map)
}
```

sits_svm *Train support vector machine models*

Description

This function receives a tibble with a set of attributes X for each observation Y. These attributes are the values of the time series for each band. The SVM algorithm is used for multiclass-classification. For this purpose, it uses the "one-against-one" approach, in which k(k-1)/2 binary classifiers are trained; the appropriate class is found by a voting scheme. This function is a front-end to the "svm" method in the "e1071" package. Please refer to the documentation in that package for more details.

Usage

```
sits_svm(
  samples = NULL,
  formula = sits_formula_linear(),
  scale = FALSE,
  cachesize = 1000,
  kernel = "radial",
  degree = 3,
  coef0 = 0,
  cost = 10,
  tolerance = 0.001,
  epsilon = 0.1,
  cross = 10,
  ...
)
```

Arguments

samples	Time series with the training samples.
formula	Symbolic description of the model to be fit. (default: sits_formula_linear).
scale	Logical vector indicating the variables to be scaled.
cachesize	Cache memory in MB (default = 1000).
kernel	Kernel used in training and predicting. options: "linear", "polynomial", "radial", "sigmoid" (default: "radial").
degree	Exponential of polynomial type kernel (default: 3).
coef0	Parameter needed for kernels of type polynomial and sigmoid (default: 0).
cost	Cost of constraints violation (default: 10).
tolerance	Tolerance of termination criterion (default: 0.001).
epsilon	Epsilon in the insensitive-loss function (default: 0.1).
cross	Number of cross validation folds applied to assess the quality of the model (default: 10).
...	Other parameters to be passed to e1071::svm function.

Value

Model fitted to input data (to be passed to `sits_classify`)

Note

Please refer to the sits documentation available in <https://e-sensing.github.io/sitsbook/> for detailed examples.

Author(s)

Alexandre Ywata de Carvalho, <alexandre.ywata@ipea.gov.br>

Rolf Simoes, <rolf.simoes@inpe.br>

Gilberto Camara, <gilberto.camara@inpe.br>

Examples

```
if (sits_run_examples()) {
    # Example of training a model for time series classification
    # Retrieve the samples for Mato Grosso
    # train an SVM model
    ml_model <- sits_train(samples_modis_4bands, ml_method = sits_svm)
    # select the bands to classify the point
    sample_bands <- sits_bands(samples_modis_4bands)
    point_4bands <- sits_select(point_mt_6bands, bands = sample_bands)
    # classify the point
    point_class <- sits_classify(point_4bands, ml_model)
    plot(point_class)
}
```

`sits_tae` *Train a model using Temporal Self-Attention Encoder*

Description

Implementation of Temporal Attention Encoder (TAE) for satellite image time series classification.

This function is based on the paper by Vivien Garnot referenced below and code available on github at https://github.com/VSainteuf/pytorch-psetae.

We also used the code made available by Maja Schneider in her work with Marco Körner referenced below and available at https://github.com/maja601/RC2020-psetae.

If you use this method, please cite Garnot's and Schneider's work.

Usage

```
sits_tae(
  samples = NULL,
  samples_validation = NULL,
  epochs = 150,
  batch_size = 64,
  validation_split = 0.2,
  optimizer = torchopt::optim_adamw,
  opt_hparams = list(lr = 0.001, eps = 1e-08, weight_decay = 1e-06),
  lr_decay_epochs = 1,
  lr_decay_rate = 0.95,
  patience = 20,
  min_delta = 0.01,
  verbose = FALSE
)
```

Arguments

`samples`	Time series with the training samples.
`samples_validation`	Time series with the validation samples. if the `samples_validation` parameter is provided, the `validation_split` parameter is ignored.
`epochs`	Number of iterations to train the model.
`batch_size`	Number of samples per gradient update.
`validation_split`	Number between 0 and 1. Fraction of training data to be used as validation data.
`optimizer`	Optimizer function to be used.
`opt_hparams`	Hyperparameters for optimizer: lr : Learning rate of the optimizer eps: Term added to the denominator to improve numerical stability. weight_decay: L2 regularization
`lr_decay_epochs`	Number of epochs to reduce learning rate.
`lr_decay_rate`	Decay factor for reducing learning rate.
`patience`	Number of epochs without improvements until training stops.
`min_delta`	Minimum improvement to reset the patience counter.
`verbose`	Verbosity mode (TRUE/FALSE). Default is FALSE.

Value

A fitted model to be used for classification.

Note

Please refer to the sits documentation available in <https://e-sensing.github.io/sitsbook/> for detailed examples.

Author(s)

Charlotte Pelletier, <charlotte.pelletier@univ-ubs.fr>

Gilberto Camara, <gilberto.camara@inpe.br>

Rolf Simoes, <rolf.simoes@inpe.br>

References

Vivien Garnot, Loic Landrieu, Sebastien Giordano, and Nesrine Chehata, "Satellite Image Time Series Classification with Pixel-Set Encoders and Temporal Self-Attention", 2020 Conference on Computer Vision and Pattern Recognition. pages 12322-12331. DOI: 10.1109/CVPR42600.2020.01234

Schneider, Maja; Körner, Marco, "[Re] Satellite Image Time Series Classification with Pixel-Set Encoders and Temporal Self-Attention." ReScience C 7 (2), 2021. DOI: 10.5281/zenodo.4835356

Examples

```
if (sits_run_examples()) {
    # select a set of samples
    samples_ndvi <- sits_select(samples_modis_4bands, bands = c("NDVI"))
    # create a TAE model
    torch_model <- sits_train(samples_ndvi, sits_tae())
    # plot the model
    plot(torch_model)
    # create a data cube from local files
    data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
    cube <- sits_cube(
        source = "BDC",
        collection = "MOD13Q1-6",
        data_dir = data_dir,
        delim = "_",
        parse_info = c("X1", "X2", "tile", "band", "date")
    )
    # classify a data cube
    probs_cube <- sits_classify(data = cube, ml_model = torch_model)
    # plot the probability cube
    plot(probs_cube)
    # smooth the probability cube using Bayesian statistics
    bayes_cube <- sits_smooth(probs_cube)
    # plot the smoothed cube
    plot(bayes_cube)
    # label the probability cube
    label_cube <- sits_label_classification(bayes_cube)
    # plot the labelled cube
    plot(label_cube)
}
```

sits_tempcnn *Train temporal convolutional neural network models*

Description

Use a TempCNN algorithm to classify data, which has two stages: a 1D CNN and a multi-layer perceptron. Users can define the depth of the 1D network, as well as the number of perceptron layers.

This function is based on the paper by Charlotte Pelletier referenced below. If you use this method, please cite the original tempCNN paper.

The torch version is based on the code made available by the BreizhCrops team: Marc Russwurm, Charlotte Pelletier, Marco Korner, Maximilian Zollner. The original python code is available at the website https://github.com/dl4sits/BreizhCrops. This code is licensed as GPL-3.

Usage

```
sits_tempcnn(
  samples = NULL,
  samples_validation = NULL,
  cnn_layers = c(128, 128, 128),
  cnn_kernels = c(7, 7, 7),
  cnn_dropout_rates = c(0.2, 0.2, 0.2),
  dense_layer_nodes = 256,
  dense_layer_dropout_rate = 0.5,
  epochs = 150,
  batch_size = 64,
  validation_split = 0.2,
  optimizer = torchopt::optim_adamw,
  opt_hparams = list(lr = 0.005, eps = 1e-08, weight_decay = 1e-06),
  lr_decay_epochs = 1,
  lr_decay_rate = 0.95,
  patience = 20,
  min_delta = 0.01,
  verbose = FALSE
)
```

Arguments

`samples` Time series with the training samples.

`samples_validation` Time series with the validation samples. if the `samples_validation` parameter is provided, the `validation_split` parameter is ignored.

`cnn_layers` Number of 1D convolutional filters per layer

`cnn_kernels` Size of the 1D convolutional kernels.

`cnn_dropout_rates` Dropout rates for 1D convolutional filters.

dense_layer_nodes
: Number of nodes in the dense layer.

dense_layer_dropout_rate
: Dropout rate (0,1) for the dense layer.

epochs
: Number of iterations to train the model.

batch_size
: Number of samples per gradient update.

validation_split
: Fraction of training data to be used for validation.

optimizer
: Optimizer function to be used.

opt_hparams
: Hyperparameters for optimizer: lr : Learning rate of the optimizer eps: Term added to the denominator to improve numerical stability. weight_decay: L2 regularization

lr_decay_epochs
: Number of epochs to reduce learning rate.

lr_decay_rate
: Decay factor for reducing learning rate.

patience
: Number of epochs without improvements until training stops.

min_delta
: Minimum improvement in loss function to reset the patience counter.

verbose
: Verbosity mode (TRUE/FALSE). Default is FALSE.

Value

A fitted model to be used for classification.

Note

Please refer to the sits documentation available in <https://e-sensing.github.io/sitsbook/> for detailed examples.

Author(s)

Charlotte Pelletier, <charlotte.pelletier@univ-ubs.fr>

Gilberto Camara, <gilberto.camara@inpe.br>

Rolf Simoes, <rolf.simoes@inpe.br>

Felipe Souza, <lipecaso@gmail.com>

References

Charlotte Pelletier, Geoffrey Webb and François Petitjean, "Temporal Convolutional Neural Network for the Classification of Satellite Image Time Series", Remote Sensing, 11,523, 2019. DOI: 10.3390/rs11050523.

Examples

```
if (sits_run_examples()) {
    # select a set of samples
    samples_ndvi <- sits_select(samples_modis_4bands, bands = c("NDVI"))
    # create a TempCNN model
    torch_model <- sits_train(samples_ndvi, sits_tempcnn())
    # plot the model
    plot(torch_model)
    # create a data cube from local files
    data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
    cube <- sits_cube(
        source = "BDC",
        collection = "MOD13Q1-6",
        data_dir = data_dir,
        delim = "_",
        parse_info = c("X1", "X2", "tile", "band", "date")
    )
    # classify a data cube
    probs_cube <- sits_classify(data = cube, ml_model = torch_model)
    # plot the probability cube
    plot(probs_cube)
    # smooth the probability cube using Bayesian statistics
    bayes_cube <- sits_smooth(probs_cube)
    # plot the smoothed cube
    plot(bayes_cube)
    # label the probability cube
    label_cube <- sits_label_classification(bayes_cube)
    # plot the labelled cube
    plot(label_cube)
}
```

---

sits_timeline — *Get timeline of a cube or a set of time series*

Description

This function returns the timeline for a given data set, either a set of time series, a data cube, or a trained model.

Usage

```
sits_timeline(data)
```

Arguments

`data`	either a sits tibble, a data cube, or a trained model.

Value

Timeline of sample set or of data cube.

Author(s)

Gilberto Camara, <gilberto.camara@inpe.br>

Examples

```
sits_timeline(samples_modis_4bands)
```

`sits_time_series` *Get the time series for a row of a sits tibble*

Description

Returns the time series associated to a row of the a sits tibble

Usage

```
sits_time_series(data)
```

Arguments

`data`	A sits tibble with one or more time series.

Value

A tibble in sits format with the time series.

Author(s)

Gilberto Camara, <gilberto.camara@inpe.br>

Examples

```
sits_time_series(cerrado_2classes)
```

sits_to_csv — *Export a sits tibble metadata to the CSV format*

Description

Converts metadata from a sits tibble to a CSV file. The CSV file will not contain the actual time series. Its columns will be the same as those of a CSV file used to retrieve data from ground information ("latitude", "longitude", "start_date", "end_date", "cube", "label").

Usage

```
sits_to_csv(data, file)
```

Arguments

data	Time series.
file	Name of the exported CSV file.

Value

No return value, called for side effects.

Author(s)

Gilberto Camara, <gilberto.camara@inpe.br>

Examples

```
csv_file <- paste0(tempdir(), "/cerrado_2classes.csv")
sits_to_csv(cerrado_2classes, file = csv_file)
```

sits_to_xlsx — *Save accuracy assessments as Excel files*

Description

Saves confusion matrices as Excel spreadsheets. This function takes the a list of accuracy assessments generated by sits_accuracy and saves them in an Excel spreadsheet.

Usage

```
sits_to_xlsx(acc_lst, file, data = NULL)
```

Arguments

`acc_lst`	A list of accuracy statistics
`file`	The file where the XLSX data is to be saved.
`data`	(optional) Print information about the samples

Value

No return value, called for side effects.

Note

Please refer to the sits documentation available in <https://e-sensing.github.io/sitsbook/> for detailed examples.

Author(s)

Gilberto Camara, <gilberto.camara@inpe.br>

Examples

```
if (sits_run_examples()) {
    # A dataset containing a tibble with time series samples
    # for the Mato Grosso state in Brasil
    # create a list to store the results
    results <- list()

    # accuracy assessment lightTAE
    acc_ltae <- sits_kfold_validate(samples_modis_4bands,
        folds = 5,
        multicores = 1,
        ml_method = sits_lighttae()
    )
    # use a name
    acc_ltae$name <- "LightTAE"

    # put the result in a list
    results[[length(results) + 1]] <- acc_ltae

    # save to xlsx file
    sits_to_xlsx(results, file = "./accuracy_mato_grosso_dl.xlsx")
}
```

---

sits_train *Train classification models*

Description

Given a tibble with a set of distance measures, returns trained models. Currently, sits supports the following models: 'svm' (see sits_svm), random forests (see sits_rfor), extreme gradient boosting (see sits_xgboost), and different deep learning functions, including multi-layer perceptrons (see sits_mlp), 1D convolution neural networks sits_tempcnn, deep residual networks sits_resnet and self-attention encoders sits_lighttae

Usage

```
sits_train(samples, ml_method = sits_svm())
```

Arguments

samples	Time series with the training samples.
ml_method	Machine learning method.

Value

Model fitted to input data to be passed to sits_classify

Author(s)

Rolf Simoes, <rolf.simoes@inpe.br>

Gilberto Camara, <gilberto.camara@inpe.br>

Alexandre Ywata de Carvalho, <alexandre.ywata@ipea.gov.br>

Examples

```
# Retrieve the set of samples for Mato Grosso (provided by EMBRAPA)
# fit a training model (RFOR model)
samples <- sits_select(samples_modis_4bands, bands = c("NDVI"))
ml_model <- sits_train(samples, sits_rfor(num_trees = 50))
# get a point and classify the point with the ml_model
point_ndvi <- sits_select(point_mt_6bands, bands = "NDVI")
class <- sits_classify(point_ndvi, ml_model)
```

---

sits_tuning *Tuning machine learning models hyper-parameters*

Description

Machine learning models use stochastic gradient descent (SGD) techniques to find optimal solutions. To perform SGD, models use optimization algorithms which have hyperparameters that have to be adjusted to achieve best performance for each application.

This function performs a random search on values of selected hyperparameters. Instead of performing an exhaustive test of all parameter combinations, it selecting them randomly. Validation is done using an independent set of samples or by a validation split. The function returns the best hyper-parameters in a list.

hyper-parameters passed to `params` parameter should be passed by calling `sits_tuning_hparams()` function.

Usage

```
sits_tuning(
  samples,
  samples_validation = NULL,
  validation_split = 0.2,
  ml_method = sits_tempcnn(),
 params = sits_tuning_hparams(optimizer = torchopt::optim_adamw, opt_hparams = list(lr
    = beta(0.3, 5))),
  trials = 30,
  multicores = 2,
  progress = FALSE
)
```

Arguments

`samples`	Time series set to be validated.
`samples_validation`	Time series set used for validation.
`validation_split`	Percent of original time series set to be used for validation (if samples_validation is NULL)
`ml_method`	Machine learning method.
`params`	List with hyper parameters to be passed to `ml_method`. User can use `uniform`, `choice`, `randint`, `normal`, `lognormal`, `loguniform`, and `beta` distribution functions to randomize parameters.
`trials`	Number of random trials to perform the random search.
`multicores`	Number of cores to process in parallel
`progress`	Show progress bar?

Value

A tibble containing all parameters used to train on each trial ordered by accuracy

Note

Please refer to the sits documentation available in <https://e-sensing.github.io/sitsbook/> for detailed examples.

References

James Bergstra, Yoshua Bengio, "Random Search for Hyper-Parameter Optimization". Journal of Machine Learning Research. 13: 281–305, 2012.

Examples

```
if (sits_run_examples()) {
    # find best learning rate parameters for TempCNN
    tuned <- sits_tuning(
        samples_modis_4bands,
        ml_method = sits_tempcnn(),
        params = sits_tuning_hparams(
            optimizer = choice(
                torchopt::optim_adamw
            ),
            opt_hparams = list(
                lr = beta(0.3, 5)
            )
        ),
        trials = 4,
        multicores = 2,
        progress = FALSE
    )
    # obtain best accuracy, kappa and best_lr
    accuracy <- tuned$accuracy[[1]]
    kappa <- tuned$kappa[[1]]
    best_lr <- tuned$opt_hparams[[1]]$lr
}
```

`sits_tuning_hparams` *Tuning machine learning models hyper-parameters*

Description

This function allow user building the hyper-parameters space used by `sits_tuning()` function search randomly the best parameter combination.

User should pass the possible values for hyper-parameters as constant or by calling the following random functions:

- `uniform(min = 0, max = 1, n = 1)`: returns random numbers from a uniform distribution with parameters min and max.
- `choice(..., replace = TRUE, n = 1)`: returns random objects passed to `...` with replacement or not (parameter `replace`).
- `randint(min, max, n = 1)`: returns random integers from a uniform distribution with parameters min and max.
- `normal(mean = 0, sd = 1, n = 1)`: returns random numbers from a normal distribution with parameters min and max.
- `lognormal(meanlog = 0, sdlog = 1, n = 1)`: returns random numbers from a lognormal distribution with parameters min and max.
- `loguniform(minlog = 0, maxlog = 1, n = 1)`: returns random numbers from a loguniform distribution with parameters min and max.

- `beta(shape1, shape2, n = 1)`: returns random numbers from a beta distribution with parameters min and max.

These functions accepts `n` parameter to indicate how many values should be returned.

Usage

```
sits_tuning_hparams(...)
```

Arguments

`...`	Used to prepare hyper-parameter space

Value

A list containing the hyper-parameter space to be passed to `sits_tuning()`'s `params` parameter.

Examples

```
if (sits_run_examples()) {
    # find best learning rate parameters for TempCNN
    tuned <- sits_tuning(
        samples_modis_4bands,
        ml_method = sits_tempcnn(),
        params = sits_tuning_hparams(
            optimizer = choice(
                torchopt::optim_adamw,
                torchopt::optim_yogi
            ),
            opt_hparams = list(
                lr = beta(0.3, 5)
            )
        ),
        trials = 4,
        multicores = 2,
        progress = FALSE
    )
}
```

`sits_twdtw_classify` *Find matches between patterns and time series using TWDTW*

Description

Returns the results of the TWDTW matching function. The TWDTW matching function compares the values of a satellite image time series with the values of known patters and tries to match each pattern to a part of the time series

The TWDTW (time-weighted dynamical time warping) is a version of the Dynamic Time Warping method for LUCC mapping using a sequence of multi-band satellite images. Methods based on dynamic time warping are flexible to handle irregular sampling and out-of-phase time series, and they have achieved significant results in time series analysis. In contrast to standard DTW, the TWDTW method is sensitive to seasonal changes of natural and cultivated vegetation types. It also considers inter-annual climatic and seasonal variability.

Usage

```
sits_twdtw_classify(
  samples,
  patterns,
  bands = NULL,
  dist_method = "euclidean",
  alpha = -0.1,
  beta = 100,
  theta = 0.5,
  span = 0,
  keep = FALSE,
  start_date = NULL,
  end_date = NULL,
  interval = "12 month",
  overlap = 0.5,
  .plot = TRUE
)
```

Arguments

`samples`	A sits tibble to be classified using TWDTW.
`patterns`	Patterns to be used for classification.
`bands`	Names of the bands to be used for classification.
`dist_method`	Name of the method to derive the local cost matrix.
`alpha`	Steepness of the logistic function used for temporal weighting (a double value).
`beta`	Midpoint (in days) of the logistic function.
`theta`	Relative weight of the time distance compared to the dtw distance.
`span`	Minimum number of days between two matches of the same pattern in the time series (approximate).
`keep`	Keep internal values for plotting matches?
`start_date`	Start date of the classification period.
`end_date`	End date of the classification period.
`interval`	Period between two classifications in months.
`overlap`	Minimum overlapping between one match and the interval of classification.
`.plot`	Plot the output?

Value

A dtwSat S4 object with the matches.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

Gilberto Camara, <gilberto.camara@inpe.br>

References

Maus V, Camara G, Cartaxo R, Sanchez A, Ramos F, Queiroz G (2016). A Time-Weighted Dynamic Time Warping Method for Land-Use and Land-Cover Mapping. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 9(8):3729-3739, August 2016. ISSN 1939-1404. doi:10.1109/JSTARS.2016.2517118.

Examples

```
if (sits_run_examples()){
# Retrieve the set of samples for the Mato Grosso region
samples <- sits_select(samples_modis_4bands, bands = c("NDVI", "EVI"))

# get a point and classify the point with the ml_model
point <- sits_select(point_mt_6bands, bands = c("NDVI", "EVI"))

# plot the series
plot(point)

# obtain a set of patterns for these samples
patterns <- sits_patterns(samples)
plot(patterns)

# find the matches between the patterns and the time series
# using the TWDTW algorithm
# (uses the dtwSat R package)
matches <- sits_twdtw_classify(point, patterns,
    bands = c("NDVI", "EVI"),
    alpha = -0.1, beta = 100, theta = 0.5, keep = TRUE
)
}
```

---

sits_uncertainty *Estimate classification uncertainty based on probs cube*

Description

Calculate the uncertainty cube based on the probabilities produced by the classifier. Takes a probability cube as input. The uncertainty measure is relevant in the context of active leaning, and helps to increase the quantity and quality of training samples by providing information about the confidence of the model. The supported types of uncertainty are 'entropy', 'least', 'margin' and 'ratio'.

'entropy' is the difference between all predictions expressed as entropy, 'least' is the difference between 100 prediction, 'margin' is the difference between the two most confident predictions, and 'ratio' is the ratio between the two most confident predictions.

Usage

```
sits_uncertainty(
  cube,
  type = "least",
  ...,
  multicores = 2,
  memsize = 8,
  output_dir = ".",
  version = "v1"
)

## S3 method for class 'entropy'
sits_uncertainty(
  cube,
  type = "entropy",
  ...,
  window_size = 5,
  window_fn = "median",
  multicores = 2,
  memsize = 4,
  output_dir = ".",
  version = "v1"
)

## S3 method for class 'least'
sits_uncertainty(
  cube,
  type = "least",
  ...,
  window_size = 5,
  window_fn = "median",
  multicores = 2,
  memsize = 4,
  output_dir = ".",
  version = "v1"
)

## S3 method for class 'margin'
sits_uncertainty(
  cube,
  type = "margin",
  ...,
  window_size = 5,
  window_fn = "median",
```

```
  multicores = 2,
  memsize = 4,
  output_dir = ".",
  version = "v1"
)

## S3 method for class 'ratio'
sits_uncertainty(
  cube,
  type = "ratio",
  ...,
  window_size = 5,
  window_fn = "median",
  multicores = 2,
  memsize = 4,
  output_dir = ".",
  version = "v1"
)
```

Arguments

cube	Probability data cube.
type	Method to measure uncertainty. See details.
...	Parameters for specific functions.
multicores	Number of cores to run the function.
memsize	Maximum overall memory (in GB) to run the function.
output_dir	Output directory for image files.
version	Version of resulting image. (in the case of multiple tests)
window_size	Size of neighborhood to calculate entropy.
window_fn	Function to be applied in entropy calculation.

Value

An uncertainty data cube

Note

Please refer to the sits documentation available in <https://e-sensing.github.io/sitsbook/> for detailed examples.

Author(s)

Gilberto Camara, <gilberto.camara@inpe.br>

Rolf Simoes, <rolf.simoes@inpe.br>

Alber Sanchez, <alber.ipia@inpe.br>

References

Monarch, Robert Munro. Human-in-the-Loop Machine Learning: Active learning and annotation for human-centered AI. Simon and Schuster, 2021.

Examples

```
if (sits_run_examples()) {
    # select a set of samples
    samples_ndvi <- sits_select(samples_modis_4bands, bands = c("NDVI"))
    # create a random forest model
    rfor_model <- sits_train(samples_ndvi, sits_rfor())
    # create a data cube from local files
    data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
    cube <- sits_cube(
        source = "BDC",
        collection = "MOD13Q1-6",
        data_dir = data_dir,
        delim = "_",
        parse_info = c("X1", "X2", "tile", "band", "date")
    )
    # classify a data cube
    probs_cube <- sits_classify(data = cube, ml_model = rfor_model)
    # calculate uncertainty
    uncert_cube <- sits_uncertainty(probs_cube)
    # plot the resulting uncertainty cube
    plot(uncert_cube)
}
```

---

sits_uncertainty_sampling

Suggest samples for enhancing classification accuracy

Description

Suggest samples for regions of high uncertainty as predicted by the model. The function selects data points that have confused an algorithm. These points don't have labels and need be manually labelled by experts and then used to increase the classification's training set.

This function is best used in the following context

- 1. Select an initial set of samples.
- 2. Train a machine learning model.
- 3. Build a data cube and classify it using the model.
- 4. Run a Bayesian smoothing in the resulting probability cube.
- 5. Create an uncertainty cube.
- 6. Perform uncertainty sampling.

The Bayesian smoothing procedure will reduce the classification outliers and thus increase the likelihood that the resulting pixels with high uncertainty have meaningful information.

Usage

```
sits_uncertainty_sampling(
  uncert_cube,
  n = 100,
  min_uncert = 0.4,
  sampling_window = 10
)
```

Arguments

uncert_cube	An uncertainty cube. See `sits_uncertainty`.
n	Number of suggested points.
min_uncert	Minimum uncertainty value to select a sample.
sampling_window	Window size for collecting points (in pixels). The minimum window size is 10.

Value

A tibble with longitude and latitude in WGS84 with locations which have high uncertainty and meet the minimum distance criteria.

Author(s)

Alber Sanchez, <alber.ipia@inpe.br>

Rolf Simoes, <rolf.simoes@inpe.br>

Felipe Carvalho, <felipe.carvalho@inpe.br>

Gilberto Camara, <gilberto.camara@inpe.br>

References

Robert Monarch, "Human-in-the-Loop Machine Learning: Active learning and annotation for human-centered AI". Manning Publications, 2021.

Examples

```
if (sits_run_examples()) {
    # create a data cube
    data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
    cube <- sits_cube(
        source = "BDC",
        collection = "MOD13Q1-6",
        data_dir = data_dir,
        delim = "_",
        parse_info = c("X1", "X2", "tile", "band", "date")
    )
    # build a random forest model
    samples_ndvi <- sits_select(samples_modis_4bands, bands = c("NDVI"))
    rfor_model <- sits_train(samples_ndvi, ml_method = sits_rfor())
```

```
    # classify the cube
    probs_cube <- sits_classify(data = cube, ml_model = rfor_model)
    # create an uncertainty cube
    uncert_cube <- sits_uncertainty(probs_cube)
    # obtain a new set of samples for active learning
    # the samples are located in uncertain places
    new_samples <- sits_uncertainty_sampling(uncert_cube)
}
```

---

sits_validate *Validate time series samples*

Description

One round of cross-validation involves partitioning a sample of data into complementary subsets, performing the analysis on one subset (called the training set), and validating the analysis on the other subset (called the validation set or testing set).

The function takes two arguments: a set of time series with a machine learning model and another set with validation samples. If the validation sample set is not provided, The sample dataset is split into two parts, as defined by the parameter validation_split. The accuracy is determined by the result of the validation test set.

This function returns the confusion matrix, and Kappa values.

Usage

```
sits_validate(
  samples,
  samples_validation = NULL,
  validation_split = 0.2,
  ml_method = sits_rfor()
)
```

Arguments

`samples` Time series set to be validated.

`samples_validation` Time series set used for validation.

`validation_split` Percent of original time series set to be used for validation (if samples_validation is NULL)

`ml_method` Machine learning method.

Value

A `caret::confusionMatrix` object to be used for validation assessment.

Author(s)

Rolf Simoes, <rolf.simoes@inpe.br>

Gilberto Camara, <gilberto.camara@inpe.br>

Examples

```
if (sits_run_examples()){
   conf_matrix <- sits_validate(cerrado_2classes)
}
```

---

sits_values *Return the values of a set of time series*

Description

This function returns the values of a sits tibble (according a specified format). This function is useful to use packages such as ggplot2, dtwclust, or kohonen that require values that are rowwise or colwise organized.

Usage

```
sits_values(data, bands = NULL, format = "cases_dates_bands")
```

Arguments

data	A sits tibble with time series for different bands.
bands	Bands whose values are to be extracted.
format	A string with either "cases_dates_bands" or "bands_cases_dates" or "bands_dates_cases".

Value

A matrix with values.

Author(s)

Rolf Simoes, <rolf.simoes@inpe.br>

Examples

```
# Retrieve a set of time series with 2 classes
data(cerrado_2classes)
# retrieve the values split by bands and dates
ls1 <- sits_values(cerrado_2classes[1:2, ], format = "bands_dates_cases")
# retrieve the values split by cases (occurences)
ls2 <- sits_values(cerrado_2classes[1:2, ], format = "cases_dates_bands")
#' # retrieve the values split by bands and cases (occurences)
ls3 <- sits_values(cerrado_2classes[1:2, ], format = "bands_cases_dates")
```

sits_view *View data cubes and samples in leaflet*

Description

Uses leaflet to visualize time series, raster cube and classified images

Usage

```
sits_view(x, ...)

## S3 method for class 'sits'
sits_view(x, ..., legend = NULL, palette = "Harmonic")

## S3 method for class 'som_map'
sits_view(
  x,
  ...,
  label,
  prob_max = 1,
  prob_min = 0.7,
  legend = NULL,
  palette = "Harmonic"
)

## S3 method for class 'raster_cube'
sits_view(
  x,
  ...,
  band = NULL,
  red = NULL,
  green = NULL,
  blue = NULL,
  tiles = NULL,
  dates = NULL,
  class_cube = NULL,
  legend = NULL,
  palette = "default"
)

## S3 method for class 'classified_image'
sits_view(x, ..., tiles = NULL, legend = NULL, palette = "default")

## S3 method for class 'probs_cube'
sits_view(x, ...)

## Default S3 method:
```

```
sits_view(x, ...)
```

Arguments

`x`	Object of class "sits", "raster_cube" or "classified image".
`...`	Further specifications for sits_view.
`legend`	Named vector that associates labels to colors.
`palette`	Palette provided in the configuration file.
`label`	Label from the SOM map to be shown.
`prob_max`	Maximum a posteriori probability for SOM neuron samples to be shown
`prob_min`	Minimum a posteriori probability for SOM neuron samples to be shown
`band`	For plotting grey images.
`red`	Band for red color.
`green`	Band for green color.
`blue`	Band for blue color.
`tiles`	Tiles to be plotted (in case of a multi-tile cube).
`dates`	Dates to be plotted.
`class_cube`	Classified cube to be overlayed on top on image.

Value

A leaflet object containing either samples or data cubes embedded in a global map that can be visualized directly in an RStudio viewer.

Note

Please refer to the sits documentation available in <https://e-sensing.github.io/sitsbook/> for detailed examples.

Author(s)

Gilberto Camara, <gilberto.camara@inpe.br>

Examples

```
if (sits_run_examples()) {
    sits_view(cerrado_2classes)

    data_dir <- system.file("extdata/raster/mod13q1", package = "sits")

    modis_cube <- sits_cube(
        source = "BDC",
        collection = "MOD13Q1-6",
        data_dir = data_dir,
        parse_info = c("X1", "X2", "tile", "band", "date")
    )
    # get the timeline
```

```
    timeline <- sits_timeline(modis_cube)
    # view the data cube
    sits_view(modis_cube,
        band = "NDVI",
        dates = timeline[[1]]
    )

    samples_ndvi <- sits_select(samples_modis_4bands,
        bands = c("NDVI")
    )
    rf_model <- sits_train(samples_ndvi, sits_rfor())

    modis_probs <- sits_classify(
        data = modis_cube,
        ml_model = rf_model,
        output_dir = tempdir(),
        memsize = 4,
        multicores = 1
    )
    modis_label <- sits_label_classification(modis_probs,
        output_dir = tempdir()
    )

    sits_view(modis_label)

    sits_view(modis_cube,
        band = "NDVI",
        class_cube = modis_label,
        dates = sits_timeline(modis_cube)[[1]]
    )
}
```

`sits_xgboost` *Train extreme gradient boosting models*

Description

This function uses the extreme gradient boosting algorithm. Boosting iteratively adds basis functions in a greedy fashion so that each new basis function further reduces the selected loss function. This function is a front-end to the methods in the "xgboost" package. Please refer to the documentation in that package for more details.

Usage

```
sits_xgboost(
  samples = NULL,
  learning_rate = 0.15,
  min_split_loss = 1,
  max_depth = 5,
```

```
  min_child_weight = 1,
  max_delta_step = 1,
  subsample = 0.8,
  nfold = 5,
  nrounds = 100,
  early_stopping_rounds = 20,
  verbose = FALSE
)
```

Arguments

`samples`	Time series with the training samples.
`learning_rate`	Learning rate: scale the contribution of each tree by a factor of 0 < lr < 1 when it is added to the current approximation. Used to prevent overfitting. Default: 0.15
`min_split_loss`	Minimum loss reduction to make a further partition of a leaf. Default: 1.
`max_depth`	Maximum depth of a tree. Increasing this value makes the model more complex and more likely to overfit. Default: 5.
`min_child_weight`	If the leaf node has a minimum sum of instance weights lower than min_child_weight, tree splitting stops. The larger min_child_weight is, the more conservative the algorithm is. Default: 1.
`max_delta_step`	Maximum delta step we allow each leaf output to be. If the value is set to 0, there is no constraint. If it is set to a positive value, it can help making the update step more conservative. Default: 1.
`subsample`	Percentage of samples supplied to a tree. Default: 0.8.
`nfold`	Number of the subsamples for the cross-validation.
`nrounds`	Number of rounds to iterate the cross-validation (default: 100)
`early_stopping_rounds`	Training with a validation set will stop if the performance doesn't improve for k rounds.
`verbose`	Print information on statistics during the process

Value

Model fitted to input data (to be passed to `sits_classify`)

Note

Please refer to the sits documentation available in <https://e-sensing.github.io/sitsbook/> for detailed examples.

Author(s)

Rolf Simoes, `<rolf.simoes@inpe.br>`

Gilberto Camara, `<gilberto.camara@inpe.br>`

References

Tianqi Chen, Carlos Guestrin, "XGBoost : Reliable Large-scale Tree Boosting System", SIG KDD 2016.

Examples

```
if (sits_run_examples()) {
    # Example of training a model for time series classification
    # Retrieve the samples for Mato Grosso
    # train a xgboost model
    ml_model <- sits_train(samples_modis_4bands, ml_method = sits_xgboost)
    # select the bands to classify the point
    sample_bands <- sits_bands(samples_modis_4bands)
    point_4bands <- sits_select(point_mt_6bands, bands = sample_bands)
    # classify the point
    point_class <- sits_classify(point_4bands, ml_model)
    plot(point_class)
}
```

`%>%` *Pipe*

Description

Magrittr compound assignment pipe-operator.

Arguments

`lhs, rhs`	A visualization and a function to apply to it.

Value

Apply lhs as input to rhs function

`‘sits_labels<-‘` *Change the labels of a set of time series*

Description

Given a sits tibble with a set of labels, renames the labels to the specified in value.

Usage

```
sits_labels(data) <- value

## S3 replacement method for class 'sits'
sits_labels(data) <- value

## S3 replacement method for class 'probs_cube'
sits_labels(data) <- value
```

Arguments

`data`	Data cube or time series.
`value`	A character vector used to convert labels. Labels will be renamed to the respective value positioned at the labels order returned by `sits_labels`.

Value

A sits tibble with modified labels.

A sits tibble with modified labels.

A probs cube with modified labels.

Author(s)

Rolf Simoes, <rolf.simoes@inpe.br>

Examples

```
# show original samples ("Cerrado" and "Pasture")
sits_labels(cerrado_2classes)
# rename label samples to "Savanna" and "Grasslands"
sits_labels(cerrado_2classes) <-  c("Savanna", "Grasslands")
# see the change
sits_labels(cerrado_2classes)
```

Index