

Package ‘sits’

September 3, 2025

Type Package

Version 1.5.3-1

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. Builds satellite image data cubes from cloud collections. Supports visualization methods for images and time series and smoothing filters for dealing with noisy time series. Enables merging of multi-source imagery (SAR, optical, DEM). Includes functions for quality assessment of training samples using self-organized maps and to reduce training samples imbalance. Provides machine learning algorithms including support vector machines, random forests, extreme gradient boosting, multi-layer perceptrons, temporal convolution neural networks, and temporal attention encoders. Performs efficient classification of big Earth observation data cubes and includes functions for post-classification smoothing based on Bayesian inference. Enables best practices for estimating area and assessing accuracy of land change. Includes object-based spatio-temporal segmentation for space-time OBIA. Minimum recommended requirements: 16 GB RAM and 4 CPU dual-core.

Encoding UTF-8

Language en-US

Depends R (>= 4.1.0)

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

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

License GPL-2

ByteCompile true

LazyData true

Imports yaml (>= 2.3.0), dplyr (>= 1.1.0), grDevices, graphics, httr2 (>= 1.1.0), leafgl, leaflet (>= 2.2.2), lubridate, luz (>= 0.4.0), parallel, purrr (>= 1.0.2), randomForest, Rcpp (>= 1.1.0), rstac (>= 1.0.1), sf (>= 1.0-19), slider (>= 0.2.0), stats, terra (>= 1.8-54), tibble (>= 3.3.0), tidyr (>= 1.3.0), tmap (>= 4.1), torch (>= 0.15.0), units, utils

Suggests aws.s3, caret, cli, cols4all (>= 0.8.0), covr, dendextend, dtwclust, DiagrammeR, digest, e1071, exactextractr, FNN, gdalcubes (>= 0.7.0), geojsonsf, ggplot2, jsonlite, kohonen (>= 3.0.11), lightgbm, methods, mgcv, nnet, openxlsx, parallelly, proxy, randomForestExplainer, RColorBrewer, RcppArmadillo (>= 14.0.0), scales, spdep, stars, stringr, supercells (>= 1.0.0), testthat (>= 3.1.3), tools, xgboost

Config/testthat/edition 3

Config/testthat/parallel false

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

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 7.3.2

Collate 'api_accessors.R' 'api_accuracy.R' 'api_apply.R' 'api_band.R' 'api_bayts.R' 'api_bbox.R' 'api_block.R' 'api_check.R' 'api_chunks.R' 'api_classify.R' 'api_clean.R' 'api_cluster.R' 'api_colors.R' 'api_combine_predictions.R' 'api_comp.R' 'api_conf.R' 'api_crop.R' 'api_csv.R' 'api_cube.R' 'api_data.R' 'api_debug.R' 'api_detect_change.R' 'api_download.R' 'api_dtw.R' 'api_environment.R' 'api_factory.R' 'api_file_info.R' 'api_file.R' 'api_gdal.R' 'api_gdalcubes.R' 'api_grid.R' 'api_jobs.R' 'api_kohonen.R' 'api_label_class.R' 'api_mask.R' 'api_merge.R' 'api_mixture_model.R' 'api_ml_model.R' 'api_message.R' 'api_mosaic.R' 'api_opensearch.R' 'api_parallel.R' 'api_patterns.R' 'api_period.R' 'api_plot_time_series.R' 'api_plot_raster.R' 'api_plot_vector.R' 'api_point.R' 'api_predictors.R' 'api_raster.R' 'api_reclassify.R' 'api_reduce.R' 'api_regularize.R' 'api_request.R' 'api_request_httr2.R' 'api_roi.R' 'api_samples.R' 'api_segments.R' 'api_select.R' 'api_sf.R' 'api_shp.R' 'api_signal.R' 'api_smooth.R' 'api_smote.R' 'api_som.R' 'api_source.R' 'api_source_aws.R' 'api_source_bdc.R' 'api_source_cdse.R' 'api_source_cdse_os.R' 'api_source_deafrica.R' 'api_source_deaustralia.R' 'api_source_hls.R' 'api_source_local.R' 'api_source_mpc.R' 'api_source_ogh.R' 'api_source_sdc.R' 'api_source_stac.R' 'api_source_terrascope.R' 'api_source_usgs.R' 'api_space_time_operations.R' 'api_stac.R' 'api_stats.R' 'api_summary.R' 'api_texture.R' 'api_tibble.R' 'api_tile.R' 'api_timeline.R' 'api_tmap.R' 'api_torch.R' 'api_torch_psetae.R' 'api_ts.R' 'api_tuning.R'

'api_uncertainty.R' 'api_utils.R' 'api_validate.R'
 'api_values.R' 'api_variance.R' 'api_vector.R'
 'api_vector_info.R' 'api_view.R' 'RcppExports.R' 'data.R'
 'sits-package.R' 'sits_add_base_cube.R' 'sits_apply.R'
 'sits_accuracy.R' 'sits_bands.R' 'sits_bayts.R' 'sits_bbox.R'
 'sits_classify.R' 'sits_colors.R' 'sits_combine_predictions.R'
 'sits_config.R' 'sits_csv.R' 'sits_cube.R' 'sits_cube_copy.R'
 'sits_cube_local.R' 'sits_clean.R' 'sits_cluster.R'
 'sits_detect_change.R' 'sits_detect_change_method.R'
 'sits_dtw.R' 'sits_factory.R' 'sits_filters.R'
 'sits_geo_dist.R' 'sits_get_data.R' 'sits_get_class.R'
 'sits_get_probs.R' 'sits_grid_systems.R' 'sits_histogram.R'
 'sits_imputation.R' 'sits_labels.R'
 'sits_label_classification.R' 'sits_lighttae.R'
 'sits_lstm_fcn.R' 'sits_machine_learning.R' 'sits_merge.R'
 'sits_mixture_model.R' 'sits_mlp.R' 'sits_mosaic.R'
 'sits_model_export.R' 'sits_patterns.R' 'sits_plot.R'
 'sits_predictors.R' 'sits_reclassify.R' 'sits_reduce.R'
 'sits_reduce_imbalance.R' 'sits_regularize.R' 'sits_resnet.R'
 'sits_sample_functions.R' 'sits_segmentation.R' 'sits_select.R'
 'sits_sf.R' 'sits_smooth.R' 'sits_som.R' 'sits_stars.R'
 'sits_summary.R' 'sits_tae.R' 'sits_tempcnn.R' 'sits_terra.R'
 'sits_texture.R' 'sits_timeline.R' 'sits_train.R'
 'sits_tuning.R' 'sits_utils.R' 'sits_uncertainty.R'
 'sits_validate.R' 'sits_view.R' 'sits_variance.R' 'sits_xlsx.R'
 'zzz.R'

NeedsCompilation yes

Author Rolf Simoes [aut],
 Gilberto Camara [aut, cre, ths],
 Felipe Souza [aut],
 Felipe Carlos [aut],
 Lorena Santos [ctb],
 Charlotte Pelletier [ctb],
 Estefania Pizarro [ctb],
 Karine Ferreira [ctb, ths],
 Alber Sanchez [ctb],
 Alexandre Assuncao [ctb],
 Daniel Falbel [ctb],
 Gilberto Queiroz [ctb],
 Johannes Reiche [ctb],
 Pedro Andrade [ctb],
 Pedro Brito [ctb],
 Renato Assuncao [ctb],
 Ricardo Cartaxo [ctb]

Repository CRAN

Date/Publication 2025-09-03 18:20:02 UTC

Contents

sits-package	7
cerrado_2classes	9
hist.probs_cube	9
hist.raster_cube	10
hist.sits	11
hist.uncertainty_cube	12
impute_linear	13
plot	14
plot.class_cube	15
plot.class_vector_cube	17
plot.dem_cube	18
plot.geo_distances	20
plot.patterns	21
plot.predicted	22
plot.probs_cube	23
plot.probs_vector_cube	25
plot.raster_cube	26
plot.rfor_model	29
plot.sar_cube	30
plot.sits_accuracy	32
plot.sits_cluster	33
plot.som_clean_samples	34
plot.som_evaluate_cluster	35
plot.som_map	36
plot.torch_model	37
plot.uncertainty_cube	38
plot.uncertainty_vector_cube	39
plot.variance_cube	41
plot.vector_cube	43
plot.xgb_model	45
point_mt_6bands	46
samples_l8_rondonia_2bands	46
samples_modis_ndvi	47
sits_accuracy	47
sits_add_base_cube	50
sits_apply	51
sits_as_sf	54
sits_as_stars	55
sits_as_terra	56
sits_bands	58
sits_bbox	59
sits_classify	61
sits_classify.raster_cube	63
sits_classify.segs_cube	65
sits_classify.sits	68
sits_clean	70

sits_cluster_clean	72
sits_cluster_dendro	73
sits_cluster_frequency	74
sits_colors	75
sits_colors_qgis	76
sits_colors_reset	77
sits_colors_set	77
sits_colors_show	79
sits_combine_predictions	79
sits_confidence_sampling	82
sits_config	83
sits_config_show	84
sits_config_user_file	85
sits_cube	85
sits_cube.local_cube	88
sits_cube.results_cube	90
sits_cube.stac_cube	94
sits_cube.vector_cube	98
sits_cube_copy	101
sits_factory_function	103
sits_filter	105
sits_formula_linear	105
sits_formula_logref	106
sits_geo_dist	107
sits_get_class	109
sits_get_data	110
sits_get_data.csv	113
sits_get_data.data.frame	114
sits_get_data.sf	115
sits_get_data.shp	117
sits_get_data.sits	119
sits_get_probs	120
sits_impute	122
sits_kfold_validate	123
sits_labels	124
sits_labels<-	126
sits_labels<-.class_cube	127
sits_labels<-.default	127
sits_labels<-.probs_cube	128
sits_labels<-.sits	128
sits_labels_summary	129
sits_label_classification	129
sits_lightgbm	132
sits_lighttae	133
sits_list_collections	136
sits_lstm_fcn	137
sits_merge	138
sits_mgrs_to_roi	140

sits_mixture_model	140
sits_mlp	143
sits_model_export	145
sits_mosaic	146
sits_patterns	148
sits_predictors	150
sits_pred_features	151
sits_pred_normalize	152
sits_pred_references	153
sits_pred_sample	153
sits_reclassify	154
sits_reduce	157
sits_reduce_imbalance	159
sits_regularize	161
sits_resnet	166
sits_rfor	168
sits_roi_to_mgrs	170
sits_roi_to_tiles	171
sits_run_examples	172
sits_run_tests	172
sits_sample	173
sits_sampling_design	174
sits_segment	175
sits_select	178
sits_sgolay	179
sits_slic	180
sits_smooth	182
sits_som_clean_samples	185
sits_som_evaluate_cluster	187
sits_som_map	188
sits_som_remove_samples	190
sits_stats	191
sits_stratified_sampling	191
sits_svm	193
sits_tae	195
sits_tempcnn	197
sits_texture	200
sits_tiles_to_roi	203
sits_timeline	203
sits_timeseries_to_csv	204
sits_to_csv	205
sits_to_xlsx	206
sits_train	207
sits_tuning	209
sits_tuning_hparams	211
sits_uncertainty	212
sits_uncertainty_sampling	214
sits_validate	216

sits_variance	218
sits_view	219
sits_whittaker	225
sits_xgboost	226
summary.class_cube	228
summary.raster_cube	229
summary.sits	230
summary.sits_accuracy	230
summary.sits_area_accuracy	231
summary.variance_cube	232
Index	235

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.

Note

The main sits classification workflow has the following steps:

1. [sits_cube](#): selects a ARD image collection from a cloud provider.
2. [sits_cube_copy](#): copies an ARD image collection from a cloud provider to a local directory for faster processing.
3. [sits_regularize](#): create a regular data cube from an ARD image collection.
4. [sits_apply](#): create new indices by combining bands of a regular data cube (optional).
5. [sits_get_data](#): extract time series from a regular data cube based on user-provided labelled samples.
6. [sits_train](#): train a machine learning model based on image time series.
7. [sits_classify](#): classify a data cube using a machine learning model and obtain a probability cube.
8. [sits_smooth](#): post-process a probability cube using a spatial smoother to remove outliers and increase spatial consistency.
9. [sits_label_classification](#): produce a classified map by selecting the label with the highest probability from a smoothed cube.

Author(s)

Maintainer: Gilberto Camara <gilberto.camara.inpe@gmail.com> [thesis advisor]

Authors:

- Rolf Simoes <rolfsimoes@gmail.com>
- Felipe Souza <felipe.carvalho@inpe.br>
- Felipe Carlos <efelipecarlos@gmail.com>

Other contributors:

- Lorena Santos <lorena.santos@inpe.br> [contributor]
- Charlotte Pelletier <charlotte.pelletier@univ-ubs.fr> [contributor]
- Estefania Pizarro <eapizarro@ine.gob.cl> [contributor]
- Karine Ferreira <karine.ferreira@inpe.br> [contributor, thesis advisor]
- Alber Sanchez <alber.ipia@inpe.br> [contributor]
- Alexandre Assuncao <alexcarssuncao@gmail.com> [contributor]
- Daniel Falbel <dfalbel@gmail.com> [contributor]
- Gilberto Queiroz <gilberto.queiroz@inpe.br> [contributor]
- Johannes Reiche <johannes.reiche@wur.nl> [contributor]
- Pedro Andrade <pedro.andrade@inpe.br> [contributor]
- Pedro Brito <pedro_brito1997@hotmail.com> [contributor]
- Renato Assuncao <assuncaoest@gmail.com> [contributor]
- Ricardo Cartaxo <rcartaxoms@gmail.com> [contributor]

See Also

Useful links:

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

cerrado_2classes	<i>Samples of classes Cerrado and Pasture</i>
------------------	---

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).

hist.probs_cube	<i>histogram of prob cubes</i>
-----------------	--------------------------------

Description

This is a generic function. Parameters depend on the specific type of input.

Usage

```
## S3 method for class 'probs_cube'
hist(x, ..., tile = x[["tile"]][[1L]], label = NULL, size = 100000L)
```

Arguments

- x Object of classes "raster_cube".
- ... Further specifications for [summary](#).
- tile Tile to be shown
- label Label to be shown
- size Number of cells to be sampled

Value

A histogram of one label of a probability cube.

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")
  modis_cube <- sits_cube(
    source = "BDC",
    collection = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  rfor_model <- sits_train(samples_modis_ndvi, sits_rfor())
  probs_cube <- sits_classify(
    data = modis_cube,
    ml_model = rfor_model,
    output_dir = tempdir()
  )
  hist(probs_cube, label = "Forest")
}
```

hist.raster_cube	<i>histogram of data cubes</i>
------------------	--------------------------------

Description

This is a generic function. Parameters depend on the specific type of input.

Usage

```
## S3 method for class 'raster_cube'
hist(
  x,
  ...,
  tile = x[["tile"]][[1L]],
  date = NULL,
  band = NULL,
  size = 100000L
)
```

Arguments

x	Object of classes "raster_cube".
...	Further specifications for summary .
tile	Tile to be shown

date	Date to be shown
band	Band to be shown
size	Number of cells to be sampled

Value

A histogram of one band of data cube.

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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  hist(cube)
}
```

hist.sits

Histogram

Description

This is a generic function. Parameters depend on the specific type of input.

Usage

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

Arguments

x	Object of classes "sits".
...	Further specifications for hist .

Value

A summary of the sits tibble.

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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  hist(cube)
}
```

hist.uncertainty_cube *Histogram uncertainty cubes*

Description

This is a generic function. Parameters depend on the specific type of input.

Usage

```
## S3 method for class 'uncertainty_cube'
hist(x, ..., tile = x[["tile"]][[1L]], size = 100000L)
```

Arguments

x	Object of class "variance_cube"
...	Further specifications for hist .
tile	Tile to be summarized
size	Sample size

Value

A histogram of a uncertainty cube

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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # create a random forest model
  rfor_model <- sits_train(samples_modis_ndvi, sits_rfor())
  # classify a data cube
  probs_cube <- sits_classify(
    data = cube, ml_model = rfor_model, output_dir = tempdir()
  )
  uncert_cube <- sits_uncertainty(
    cube = probs_cube,
    output_dir = tempdir()
  )
  hist(uncert_cube)
}

```

impute_linear

*Replace NA values by linear interpolation***Description**

Remove NA by linear interpolation

Usage

```
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>

plot

*Plot time series and data cubes***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](#)
- classified time series: see [plot.predicted](#)
- raster cube: see [plot.raster_cube](#)
- SAR cube: see [plot.sar_cube](#)
- DEM cube: see [plot.dem_cube](#)
- vector cube: see [plot.vector_cube](#)
- classification probabilities: see [plot.probs_cube](#)
- classification uncertainty: see [plot.uncertainty_cube](#)
- uncertainty of vector cubes: see [plot.uncertainty_vector_cube](#)
- classified cube: see [plot.class_cube](#)
- classified vector cube: see [plot.class_vector_cube](#)
- dendrogram cluster: see [plot.sits_cluster](#)
- SOM map: see [plot.som_map](#)
- SOM evaluate cluster: see [plot.som_evaluate_cluster](#)
- geo-distances: see [plot.geo_distances](#)
- random forest model: see [plot.rfor_model](#)
- xgboost model: see [plot.xgb_model](#)
- torch ML model: see [plot.torch_model](#)

Plots the time series to be used for classification

Usage

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

Arguments

<code>x</code>	Object of class "sits".
<code>y</code>	Ignored.
<code>...</code>	Further specifications for plot .
<code>together</code>	A logical value indicating whether the samples should be plotted together.

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.

Author(s)

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

Examples

```
if (sits_run_examples()) {
  # plot sets of time series
  plot(cerrado_2classes)
}
```

plot.class_cube	<i>Plot classified images</i>
-----------------	-------------------------------

Description

plots a classified raster using tmap.

Usage

```
## S3 method for class 'class_cube'
plot(
  x,
  y,
  ...,
  tile = x[["tile"]][[1L]],
  roi = NULL,
  legend = NULL,
  palette = "Spectral",
  scale = 1,
  max_cog_size = 1024L,
  legend_position = "outside"
)
```

Arguments

x	Object of class "class_cube".
y	Ignored.
...	Further specifications for plot .
tile	Tile to be plotted.

roi	Spatial extent to plot in WGS 84 - named vector with either (lon_min, lon_max, lat_min, lat_max) or (xmin, xmax, ymin, ymax)
legend	Named vector that associates labels to colors.
palette	Alternative RColorBrewer palette
scale	Relative scale (0.4 to 1.0) of plot text
max_cog_size	Maximum size of COG overviews (lines or columns)
legend_position	Where to place the legend (default = "outside")

Value

A color map, where each pixel has the color associated to a label, as defined by the legend parameter.

Note

The following optional parameters are available to allow for detailed control over the plot output:

- graticules_labels_size: size of coord labels (default = 0.8)
- legend_title_size: relative size of legend title (default = 1.0)
- legend_text_size: relative size of legend text (default = 1.0)
- legend_bg_color: color of legend background (default = "white")
- legend_bg_alpha: legend opacity (default = 0.5)

Author(s)

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

Examples

```
if (sits_run_examples()) {
  # create a random forest model
  rfor_model <- sits_train(samples_modis_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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # classify a data cube
  probs_cube <- sits_classify(
    data = cube, ml_model = rfor_model, output_dir = tempdir()
  )
  # label cube with the most likely class
  label_cube <- sits_label_classification(
    probs_cube,
    output_dir = tempdir()
  )
  # plot the resulting classified image
```



```

    plot(label_cube)
}

```

plot.class_vector_cube

Plot Segments

Description

Plot vector classified cube

Usage

```

## S3 method for class 'class_vector_cube'
plot(
  x,
  ...,
  tile = x[["tile"]][[1L]],
  legend = NULL,
  seg_color = "black",
  line_width = 0.5,
  palette = "Spectral",
  scale = 1,
  legend_position = "outside"
)

```

Arguments

x	Object of class "segments".
...	Further specifications for plot .
tile	Tile to be plotted.
legend	Named vector that associates labels to colors.
seg_color	Segment color.
line_width	Segment line width.
palette	Alternative RColorBrewer palette
scale	Scale to plot map (0.4 to 1.0)
legend_position	Where to place the legend (default = "outside")

Value

A plot object with an RGB image or a B/W image on a color scale using the chosen palette

Note

To see which color palettes are supported, please run

Author(s)

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

Examples

```
if (sits_run_examples()) {
  data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
  cube <- sits_cube(
    source = "BDC",
    collection = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # segment the image
  segments <- sits_segment(
    cube = cube,
    output_dir = tempdir()
  )
  # create a classification model
  rfor_model <- sits_train(samples_modis_ndvi, sits_rfor())
  # classify the segments
  probs_segs <- sits_classify(
    data = segments,
    ml_model = rfor_model,
    output_dir = tempdir()
  )
  #
  # Create a classified vector cube
  class_segs <- sits_label_classification(
    cube = probs_segs,
    output_dir = tempdir(),
    multicores = 2,
    memsize = 4
  )
  # plot the segments
  plot(class_segs)
}
```

plot.dem_cube

Plot DEM cubes

Description

Plot RGB raster cube

Usage

```
## S3 method for class 'dem_cube'
plot(
  x,
```

```

...,
band = "ELEVATION",
tile = x[["tile"]][[1L]],
roi = NULL,
palette = "Spectral",
rev = TRUE,
scale = 1,
max_cog_size = 1024L,
legend_position = "inside"
)

```

Arguments

x	Object of class "dem_cube".
...	Further specifications for plot .
band	Band for plotting grey images.
tile	Tile to be plotted.
roi	Spatial extent to plot in WGS 84 - named vector with either (lon_min, lon_max, lat_min, lat_max) or (xmin, xmax, ymin, ymax)
palette	An RColorBrewer palette
rev	Reverse the color order in the palette?
scale	Scale to plot map (0.4 to 1.0)
max_cog_size	Maximum size of COG overviews (lines or columns)
legend_position	Where to place the legend (default = "inside")

Value

A plot object with a DEM cube or a B/W image on a color scale

Note

Use scale parameter for general output control.

The following optional parameters are available to allow for detailed control over the plot output:

- graticules_labels_size: size of coord labels (default = 0.7)
- legend_title_size: relative size of legend title (default = 0.7)
- legend_text_size: relative size of legend text (default = 0.7)
- legend_bg_color: color of legend background (default = "white")
- legend_bg_alpha: legend opacity (default = 0.3)

Author(s)

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

Examples

```

if (sits_run_examples()) {
  # obtain the DEM cube
  dem_cube_19HBA <- sits_cube(
    source = "MPC",
    collection = "COP-DEM-GLO-30",
    bands = "ELEVATION",
    tiles = "19HBA"
  )
  # plot the DEM reversing the palette
  plot(dem_cube_19HBA, band = "ELEVATION")
}

```

plot.geo_distances	<i>Make a kernel density plot of samples distances.</i>
--------------------	---

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.

Author(s)

Felipe Souza, <lipecaso@gmail.com>
 Rolf Simoes, <rolfsimoes@gmail.com>
 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/s41467022-298389](https://doi.org/10.1038/s41467022-298389).

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_ndvi, mt_sf)
  # plot sample-to-sample and sample-to-prediction distances
  plot(distances)
}

```

plot.patterns

Plot patterns that describe classes

Description

Plots the patterns (one plot per band/class combination) Useful to understand the trends of time series.

Usage

```

## S3 method for class 'patterns'
plot(x, y, ..., bands = NULL, year_grid = FALSE)

```

Arguments

x	Object of class "patterns".
y	Ignored.
...	Further specifications for plot .
bands	Bands to be viewed (optional).
year_grid	Plot a grid of panels using labels as columns and years as rows. Default is FALSE.

Value

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

Note

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	<i>Plot time series predictions</i>
----------------	-------------------------------------

Description

Given a sits tibble with a set of predictions, plot them. Useful to show multi-year predictions for a time series.

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

This code is reused from the dtwSat package by Victor Maus.

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 an svm model
  ml_model <- sits_train(samples_modis_ndvi, ml_method = sits_svm)
  # classify the point
  point_ndvi <- sits_select(point_mt_6bands, bands = "NDVI")
  point_class <- sits_classify(
    data = point_ndvi, ml_model = ml_model
  )
  plot(point_class)
}

```

plot.probs_cube	<i>Plot probability cubes</i>
-----------------	-------------------------------

Description

plots a probability cube

Usage

```

## S3 method for class 'probs_cube'
plot(
  x,
  ...,
  tile = x[["tile"]][[1L]],
  roi = NULL,
  labels = NULL,
  palette = "YlGn",
  rev = FALSE,
  quantile = NULL,
  scale = 1,
  max_cog_size = 512L,
  legend_position = "outside",
  legend_title = "probs"
)

```

Arguments

x	Object of class "probs_cube".
...	Further specifications for plot .
tile	Tile to be plotted.
roi	Spatial extent to plot in WGS 84 - named vector (see notes below)
labels	Labels to plot.
palette	RColorBrewer palette

<code>rev</code>	Reverse order of colors in palette?
<code>quantile</code>	Minimum quantile to plot
<code>scale</code>	Scale to plot map (0.4 to 1.0)
<code>max_cog_size</code>	Maximum size of COG overviews (lines or columns)
<code>legend_position</code>	Where to place the legend (default = "outside")
<code>legend_title</code>	Title of legend (default = "probs")

Value

A plot containing probabilities associated to each class for each pixel.

To define a roi use one of:

- A path to a shapefile with polygons;
- A `sfc` or `sf` object from `sf` package;
- A `SpatExtent` object from `terra` package;
- A named vector ("`lon_min`", "`lat_min`", "`lon_max`", "`lat_max`") in WGS84;
- A named vector ("`xmin`", "`xmax`", "`ymin`", "`ymax`") with XY coordinates.

Defining a region of interest using `SpatExtent` or XY values not in WGS84 requires the `crs` parameter to be specified. `sits_regularize()` function will crop the images that contain the region of interest().

Author(s)

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

Examples

```
if (sits_run_examples()) {
  # create a random forest model
  rfor_model <- sits_train(samples_modis_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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # classify a data cube
  probs_cube <- sits_classify(
    data = cube, ml_model = rfor_model, output_dir = tempdir()
  )
  # plot the resulting probability cube
  plot(probs_cube)
}
```

plot.probs_vector_cube

Plot probability vector cubes

Description

Plots a probability vector cube, which result from first running a segmentation [sits_segment](#) and then running a machine learning classification model. The result is a set of polygons, each with an assigned probability of belonging to a specific class.

Usage

```
## S3 method for class 'probs_vector_cube'
plot(
  x,
  ...,
  tile = x[["tile"]][[1L]],
  labels = NULL,
  palette = "YlGn",
  rev = FALSE,
  scale = 1,
  legend_position = "outside"
)
```

Arguments

x	Object of class "probs_vector_cube".
...	Further specifications for plot .
tile	Tile to be plotted.
labels	Labels to plot
palette	RColorBrewer palette
rev	Reverse order of colors in palette?
scale	Scale to plot map (0.4 to 1.0)
legend_position	Where to place the legend (default = "outside")

Value

A plot containing probabilities associated to each class for each pixel.

Author(s)

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

Examples

```

if (sits_run_examples()) {
  # create a random forest model
  rfor_model <- sits_train(samples_modis_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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # segment the image
  segments <- sits_segment(
    cube = cube,
    seg_fn = sits_slic(
      step = 5,
      compactness = 1,
      dist_fun = "euclidean",
      avg_fun = "median",
      iter = 20,
      minarea = 10,
      verbose = FALSE
    ),
    output_dir = tempdir()
  )
  # classify a data cube
  probs_vector_cube <- sits_classify(
    data = segments,
    ml_model = rfor_model,
    output_dir = tempdir()
  )
  # plot the resulting probability cube
  plot(probs_vector_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"]][[1L]],
    dates = NULL,
    roi = NULL,
    palette = "RdYlGn",
    rev = FALSE,
    scale = 1,
    first_quantile = 0.02,
    last_quantile = 0.98,
    max_cog_size = 1024L,
    legend_position = "inside"
)

```

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.
dates	Dates to be plotted
roi	Spatial extent to plot in WGS 84 - named vector with either (lon_min, lon_max, lat_min, lat_max) or (xmin, xmax, ymin, ymax)
palette	An RColorBrewer palette
rev	Reverse the color order in the palette?
scale	Scale to plot map (0.4 to 1.0)
first_quantile	First quantile for stretching images
last_quantile	Last quantile for stretching images
max_cog_size	Maximum size of COG overviews (lines or columns)
legend_position	Where to place the legend (default = "inside")

Value

A plot object with an RGB image or a B/W image on a color scale

Note

Use scale parameter for general output control. The dates parameter indicates the date allows plotting of different dates when a single band and three dates are provided, 'sits' will plot a multi-temporal RGB image for a single band (useful in the case of SAR data). For RGB bands with multi-dates, multiple plots will be produced.

If the user does not provide band names for b/w or RGB plots, and also does not provide dates, plot.raster_cube tries to display some reasonable color composites, using the following algorithm:

1. Each image in sits is associated to a source and a collection (e.g, "MPC" and "SENTINEL-2-L2A").
 2. For each source/collection pair, sits has a set of possible color composites stored in `./ext-data/config_colors.yml`. For example, the following composites are available for all "SENTINEL-2" images:
 - AGRICULTURE: ("B11", "B08", "B02")
 - AGRICULTURE2: ("B11", "B8A", "B02")
 - SWIR: ("B11", "B08", "B04")
 - SWIR2: ("B12", "B08", "B04")
 - SWIR3: ("B12", "B8A", "B04")
 - RGB: ("B04", "B03", "B02")
 - RGB-FALSE1 : ("B08", "B06", "B04")
 - RGB-FALSE2 : ("B08", "B11", "B04")
 3. sits tries to find if the bands required for one of the color composites are part of the cube. If they exist, that RGB composite is selected. Otherwise, the first available band is chosen.
 4. After selecting the bands, the algorithm looks for the date with the smallest percentage of cloud cover and selects that date to be displayed.
- . The following optional parameters are available to allow for detailed control over the plot output:
- graticules_labels_size: size of coord labels (default = 0.7)
 - legend_title_size: size of legend title (default = 0.7)
 - legend_text_size: size of legend text (default = 0.7)
 - legend_bg_color: color of legend background (default = "white")
 - legend_bg_alpha: legend opacity (default = 0.3)

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 = "MOD13Q1-6.1",
        data_dir = data_dir
    )
    # plot NDVI band of the least cloud cover date
    plot(cube)
}

```

plot.rfor_model	<i>Plot Random Forest model</i>
-----------------	---------------------------------

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_ndvi, ml_method = sits_rfor())
  # plot the model
  plot(rf_model)
}

```

plot.sar_cube	<i>Plot SAR data cubes</i>
---------------	----------------------------

Description

Plot SAR raster cube

Usage

```
## S3 method for class 'sar_cube'
plot(
  x,
  ...,
  band = NULL,
  red = NULL,
  green = NULL,
  blue = NULL,
  tile = x[["tile"]][[1L]],
  dates = NULL,
  roi = NULL,
  palette = "Greys",
  rev = FALSE,
  scale = 1,
  first_quantile = 0.05,
  last_quantile = 0.95,
  max_cog_size = 1024L,
  legend_position = "inside"
)
```

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.
dates	Dates to be plotted.
roi	Spatial extent to plot in WGS 84 - named vector with either (lon_min, lon_max, lat_min, lat_max) or (xmin, xmax, ymin, ymax)
palette	An RColorBrewer palette
rev	Reverse the color order in the palette?
scale	Scale to plot map (0.4 to 1.0)

first_quantile	First quantile for stretching images
last_quantile	Last quantile for stretching images
max_cog_size	Maximum size of COG overviews (lines or columns)
legend_position	Where to place the legend (default = "inside")

Value

A plot object with an RGB image or a B/W image on a color scale for SAR cubes

Note

Use scale parameter for general output control. The dates parameter indicates the date allows plotting of different dates when a single band and three dates are provided, 'sits' will plot a multi-temporal RGB image for a single band (useful in the case of SAR data). For RGB bands with multi-dates, multiple plots will be produced.

The following optional parameters are available to allow for detailed control over the plot output:

- graticules_labels_size: size of coord labels (default = 0.7)
- legend_title_size: relative size of legend title (default = 0.7)
- legend_text_size: relative size of legend text (default = 0.7)
- legend_bg_color: color of legend background (default = "white")
- legend_bg_alpha: legend opacity (default = 0.3)

Author(s)

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

Examples

```
if (sits_run_examples()) {
  # create a SAR data cube from cloud services
  cube_s1_grd <- sits_cube(
    source = "MPC",
    collection = "SENTINEL-1-GRD",
    bands = c("VV", "VH"),
    orbit = "descending",
    tiles = c("21LUJ"),
    start_date = "2021-08-01",
    end_date = "2021-09-30"
  )
  # plot VH band of the first date of the data cube
  plot(cube_s1_grd, band = "VH")
}
```

plot.sits_accuracy	<i>Plot confusion matrix</i>
--------------------	------------------------------

Description

Plot a bar graph with informations about the confusion matrix

Usage

```
## S3 method for class 'sits_accuracy'
plot(x, y, ..., title = "Confusion matrix")
```

Arguments

x	Object of class "plot.sits_accuracy".
y	Ignored.
...	Further specifications for plot .
title	Title of plot.

Value

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

Author(s)

Gilberto Camara <gilberto.camara@inpe.br>

Examples

```
if (sits_run_examples()) {
  # show accuracy for a set of samples
  train_data <- sits_sample(samples_modis_ndvi, frac = 0.5)
  test_data <- sits_sample(samples_modis_ndvi, frac = 0.5)
  # compute a random forest model
  rfor_model <- sits_train(train_data, sits_rfor())
  # classify training points
  points_class <- sits_classify(
    data = test_data, ml_model = rfor_model
  )
  # calculate accuracy
  acc <- sits_accuracy(points_class)
  # plot accuracy
  plot(acc)
}
```

plot.sits_cluster	<i>Plot a dendrogram cluster</i>
-------------------	----------------------------------

Description

Plot a dendrogram

Usage

```
## S3 method for class 'sits_cluster'  
plot(x, ..., cluster, cutree_height, palette)
```

Arguments

x	sits tibble with cluster indexes.
...	Further specifications for plot .
cluster	cluster object produced by 'sits_cluster' function.
cutree_height	dashed horizontal line to be drawn indicating the height of dendrogram cutting.
palette	HCL color palette.

Value

The dendrogram object.

Author(s)

Rolf Simoes, <rolfsimoes@gmail.com>

Examples

```
if (sits_run_examples()) {  
  samples <- sits_cluster_dendro(cerrado_2classes,  
    bands = c("NDVI", "EVI")  
  )  
}
```

plot.som_clean_samples

Plot SOM samples evaluated

Description

It is useful to visualize the output of the SOM evaluation, which classifies the samples as "clean" (good samples), "remove" (possible outliers), and "analyse" (borderline cases). This function plots the percentual distribution of the SOM evaluation per class. To use it, please run `sits_som_clean_samples` using the parameter "keep" as `c("clean", "analyze", "remove")`.

Usage

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

Arguments

x	Object of class "som_clean_samples".
...	Further specifications for plot .

Value

Called for side effects.

Author(s)

Estefania Pizarro, <epizarroa@ine.gob.cl>

Examples

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

`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

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

Value

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

Author(s)

Lorena Santos <lorena.santos@inpe.br>

Examples

```
if (sits_run_examples()) {
  # create a SOM map
  som_map <- sits_som_map(samples_modis_ndvi)
  # 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. If type is "codes", plots the vector weight for in each neuron. If type is "mapping", shows where samples are mapped.

Usage

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

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.
legend	Legend with colors to be plotted
band	What band will be plotted (character)

Value

Called for side effects.

Author(s)

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

Examples

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

plot.torch_model	<i>Plot Torch (deep learning) model</i>
------------------	---

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.

Author(s)

Felipe Carvalho, <lipecaso@gmail.com>

Rolf Simoes, <rolfsimoes@gmail.com>

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_ndvi, ml_method = sits_tempcnn)
  # plot the model
  plot(ml_model)
}
```

plot.uncertainty_cube *Plot uncertainty cubes*

Description

plots a uncertainty cube

Usage

```
## S3 method for class 'uncertainty_cube'
plot(
  x,
  ...,
  tile = x[["tile"]][[1L]],
  roi = NULL,
  palette = "RdYlGn",
  rev = TRUE,
  scale = 1,
  first_quantile = 0.02,
  last_quantile = 0.98,
  max_cog_size = 1024L,
  legend_position = "inside"
)
```

Arguments

x	Object of class "probs_image".
...	Further specifications for plot .
tile	Tiles to be plotted.
roi	Spatial extent to plot in WGS 84 - named vector with either (lon_min, lon_max, lat_min, lat_max) or (xmin, xmax, ymin, ymax)
palette	An RColorBrewer palette
rev	Reverse the color order in the palette?
scale	Scale to plot map (0.4 to 1.0)
first_quantile	First quantile for stretching images
last_quantile	Last quantile for stretching images
max_cog_size	Maximum size of COG overviews (lines or columns)
legend_position	Where to place the legend (default = "inside")

Value

A plot object produced showing the uncertainty associated to each classified pixel.

Note

The following optional parameters are available to allow for detailed control over the plot output:

- `graticules_labels_size`: size of coord labels (default = 0.7)
- `legend_title_size`: relative size of legend title (default = 1.0)
- `legend_text_size`: relative size of legend text (default = 1.0)
- `legend_bg_color`: color of legend background (default = "white")
- `legend_bg_alpha`: legend opacity (default = 0.5)

Author(s)

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

Examples

```
if (sits_run_examples()) {
  # create a random forest model
  rfor_model <- sits_train(samples_modis_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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # classify a data cube
  probs_cube <- sits_classify(
    data = cube, ml_model = rfor_model, output_dir = tempdir()
  )
  # calculate uncertainty
  uncert_cube <- sits_uncertainty(probs_cube, output_dir = tempdir())
  # plot the resulting uncertainty cube
  plot(uncert_cube)
}
```

plot.uncertainty_vector_cube

Plot uncertainty vector cubes

Description

plots a probability cube using stars

Usage

```
## S3 method for class 'uncertainty_vector_cube'
plot(
  x,
  ...,
  tile = x[["tile"]][[1L]],
  palette = "RdYlGn",
  rev = TRUE,
  scale = 1,
  legend_position = "inside"
)
```

Arguments

x	Object of class "probs_vector_cube".
...	Further specifications for plot .
tile	Tile to be plotted.
palette	RColorBrewer palette
rev	Reverse order of colors in palette?
scale	Scale to plot map (0.4 to 1.0)
legend_position	Where to place the legend (default = "inside")

Value

A plot containing probabilities associated to each class for each pixel.

Author(s)

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

Examples

```
if (sits_run_examples()) {
  # create a random forest model
  rfor_model <- sits_train(samples_modis_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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # segment the image
  segments <- sits_segment(
    cube = cube,
    seg_fn = sits_slic(
      step = 5,
```



```

        compactness = 1,
        dist_fun = "euclidean",
        avg_fun = "median",
        iter = 20,
        minarea = 10,
        verbose = FALSE
    ),
    output_dir = tempdir()
)
# classify a data cube
probs_vector_cube <- sits_classify(
  data = segments,
  ml_model = rfor_model,
  output_dir = tempdir()
)
# measure uncertainty
uncert_vector_cube <- sits_uncertainty(
  cube = probs_vector_cube,
  type = "margin",
  output_dir = tempdir()
)
# plot the resulting uncertainty cube
plot(uncert_vector_cube)
}

```

plot.variance_cube	<i>Plot variance cubes</i>
--------------------	----------------------------

Description

Plots a variance cube, useful to understand how local smoothing will work.

Usage

```

## S3 method for class 'variance_cube'
plot(
  x,
  ...,
  tile = x[["tile"]][[1L]],
  roi = NULL,
  labels = NULL,
  palette = "YlGnBu",
  rev = FALSE,
  type = "map",
  quantile = 0.75,
  scale = 1,
  max_cog_size = 1024L,
  legend_position = "inside",

```

```

    legend_title = "logvar"
  )

```

Arguments

x	Object of class "variance_cube".
...	Further specifications for plot .
tile	Tile to be plotted.
roi	Spatial extent to plot in WGS 84 - named vector with either (lon_min, lon_max, lat_min, lat_max) or (xmin, xmax, ymin, ymax)
labels	Labels to plot.
palette	RColorBrewer palette
rev	Reverse order of colors in palette?
type	Type of plot ("map" or "hist")
quantile	Minimum quantile to plot
scale	Scale to plot map (0.4 to 1.0)
max_cog_size	Maximum size of COG overviews (lines or columns)
legend_position	Where to place the legend (default = "inside")
legend_title	Title of legend (default = "probs")

Value

A plot containing local variances associated to the logit probability for each pixel and each class.

Author(s)

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

Examples

```

if (sits_run_examples()) {
  # create a random forest model
  rfor_model <- sits_train(samples_modis_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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # classify a data cube
  probs_cube <- sits_classify(
    data = cube, ml_model = rfor_model, output_dir = tempdir()
  )
  # obtain a variance cube
  var_cube <- sits_variance(probs_cube, output_dir = tempdir())
}

```

```

    # plot the variance cube
    plot(var_cube)
}

```

plot.vector_cube	<i>Plot RGB vector data cubes</i>
------------------	-----------------------------------

Description

Plot vector data cube with segments on top of raster image. Vector cubes have both a vector and a raster component. The vector part are the segments produced by [sits_segment](#). Their visual output is controlled by "seg_color" and "line_width" parameters. The raster output works in the same way as the false color and RGB plots.

Usage

```

## S3 method for class 'vector_cube'
plot(
  x,
  ...,
  band = NULL,
  red = NULL,
  green = NULL,
  blue = NULL,
  tile = x[["tile"]][[1L]],
  dates = NULL,
  seg_color = "yellow",
  line_width = 0.3,
  palette = "RdYlGn",
  rev = FALSE,
  scale = 1,
  first_quantile = 0.02,
  last_quantile = 0.98,
  max_cog_size = 1024L,
  legend_position = "inside"
)

```

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.

<code>tile</code>	Tile to be plotted.
<code>dates</code>	Dates to be plotted.
<code>seg_color</code>	Color to show the segment boundaries
<code>line_width</code>	Line width to plot the segments boundary (in pixels)
<code>palette</code>	An RColorBrewer palette
<code>rev</code>	Reverse the color order in the palette?
<code>scale</code>	Scale to plot map (0.4 to 1.5)
<code>first_quantile</code>	First quantile for stretching images
<code>last_quantile</code>	Last quantile for stretching images
<code>max_cog_size</code>	Maximum size of COG overviews (lines or columns)
<code>legend_position</code>	Where to place the legend (default = "inside")

Value

A plot object with an RGB image or a B/W image on a color scale using the palette

Note

The following optional parameters are available to allow for detailed control over the plot output:

- `graticules_labels_size`: size of coord labels (default = 0.7)
- `legend_title_size`: relative size of legend title (default = 0.7)
- `legend_text_size`: relative size of legend text (default = 0.7)
- `legend_bg_color`: color of legend background (default = "white")
- `legend_bg_alpha`: legend opacity (default = 0.3)

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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # Segment the cube
  segments <- sits_segment(
    cube = cube,
    output_dir = tempdir(),
    multicores = 2,
    memsize = 4
  )
}
```

```

    )
    # plot NDVI band of the second date of the data cube
    plot(segments, band = "NDVI", date = sits_timeline(cube)[1])
}

```

plot.xgb_model

Plot XGB model

Description

Plots trees in an extreme gradient boosting model.

Usage

```

## S3 method for class 'xgb_model'
plot(x, ..., trees = 0L:4L, width = 1500L, height = 1900L)

```

Arguments

x	Object of class "xgb_model".
...	Further specifications for plot .
trees	Vector of trees to be plotted
width	Width of the output window
height	Height of the output window

Value

A plot

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_ndvi,
    ml_method = sits_xgboost()
  )
  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_ndvi	<i>Samples of nine classes for the state of Mato Grosso</i>
--------------------	---

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_ndvi)
```

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	<i>Assess classification accuracy</i>
---------------	---------------------------------------

Description

This function calculates the accuracy of the classification result. The input is either a set of classified time series or a classified data cube. Classified time series are produced by [sits_classify](#). Classified images are generated using [sits_classify](#) followed by [sits_label_classification](#).

For a set of time series, sits_accuracy creates a confusion matrix and calculates the resulting statistics using package caret. For a classified image, the function uses an area-weighted technique proposed by Olofsson et al. according to references [1-3] to produce reliable accuracy estimates at 95% confidence level. 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 'class_vector_cube'
sits_accuracy(data, ..., prediction_attr, reference_attr)
```

```
## S3 method for class 'class_cube'
sits_accuracy(data, ..., validation, method = "olofsson")

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

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

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

## Default S3 method:
sits_accuracy(data, ...)
```

Arguments

<code>data</code>	Either a data cube with classified images or a set of time series
<code>...</code>	Specific parameters
<code>prediction_attr</code>	Name of the column of the segments object that contains the predicted values (only for vector class cubes)
<code>reference_attr</code>	Name of the column of the segments object that contains the reference values (only for vector class cubes)
<code>validation</code>	Samples for validation (see below) Only required when data is a raster class cube.
<code>method</code>	A character with 'olofsson' or 'pixel' to compute accuracy (only for raster class cubes)

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. The result is assigned to class "sits_accuracy" and can be visualized directly on the screen.

Note

The 'validation' data needs to contain the following columns: "latitude", "longitude", "start_date", "end_date", and "label". It can be either a path to a CSV file, a sits tibble, a data frame, or an sf object.

When 'validation' is an sf object, the columns "latitude" and "longitude" are not required as the locations are extracted from the geometry column. The 'centroid' is calculated before extracting the location values for any geometry type.

Author(s)

Rolf Simoes, <rolfsimoes@gmail.com>

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_ndvi, frac = 0.5)
  test_data <- sits_sample(samples_modis_ndvi, frac = 0.5)
  rfor_model <- sits_train(train_data, sits_rfor())
  points_class <- sits_classify(
    data = test_data, ml_model = rfor_model
  )
  acc <- sits_accuracy(points_class)

  # show accuracy for a data cube classification
  # create a random forest model
  rfor_model <- sits_train(samples_modis_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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # classify a data cube
  probs_cube <- sits_classify(
    data = cube, ml_model = rfor_model, output_dir = tempdir()
  )
  # label the probability cube
  label_cube <- sits_label_classification(
    probs_cube,
    output_dir = tempdir()
  )
  # 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 = ground_truth)
}
```

sits_add_base_cube	<i>Add base maps to a time series data cube</i>
--------------------	---

Description

This function add base maps to time series data cube. Base maps have information that is stable in time (e.g, DEM) which provide relevant information for modelling and classification.

To add a base cube to an existing data cube, they should share the same sensor, resolution, bounding box, timeline, and have different bands.

Usage

```
sits_add_base_cube(cube1, cube2)
```

Arguments

cube1	Data cube (tibble of class "raster_cube") .
cube2	Data cube (tibble of class "dem_cube") .

Value

a merged data cube with the inclusion of a base_info tibble

Author(s)

Felipe Carlos, <efelipecarlos@gmail.com>
 Felipe Carvalho, <felipe.carvalho@inpe.br>
 Gilberto Camara, <gilberto.camara@inpe.br>

Examples

```
if (sits_run_examples()) {
  s2_cube <- sits_cube(
    source = "MPC",
    collection = "SENTINEL-2-L2A",
    tiles = "18HYE",
    bands = c("B8A", "CLOUD"),
    start_date = "2022-01-01",
    end_date = "2022-03-31"
  )
  output_dir <- paste0(tempdir(), "/reg")
  if (!dir.exists(output_dir)) {
    dir.create(output_dir)
  }
}
```

```

dem_cube <- sits_cube(
  source = "MPC",
  collection = "COP-DEM-GLO-30",
  tiles = "18HYE",
  bands = "ELEVATION"
)
s2_reg <- sits_regularize(
  cube = s2_cube,
  period = "P1M",
  res = 240,
  output_dir = output_dir,
  multicores = 2,
  memsize = 4
)
dem_reg <- sits_regularize(
  cube = dem_cube,
  res = 240,
  tiles = "18HYE",
  output_dir = output_dir,
  multicores = 2,
  memsize = 4
)
s2_reg <- sits_add_base_cube(s2_reg, dem_reg)
}

```

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 creates a new band in output_dir.

Usage

```

sits_apply(data, ...)

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

## S3 method for class 'raster_cube'
sits_apply(
  data,
  ...,
  window_size = 3L,
  memsize = 4L,
  multicores = 2L,
  normalized = TRUE,
  output_dir,

```

```

    progress = TRUE
  )

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

  ## Default S3 method:
  sits_apply(data, ...)

```

Arguments

<code>data</code>	Valid sits tibble or cube
<code>...</code>	Named expressions to be evaluated (see details).
<code>window_size</code>	An odd 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).
<code>memsize</code>	Memory available for classification (in GB).
<code>multicores</code>	Number of cores to be used for classification.
<code>normalized</code>	Does the expression produces a normalized band?
<code>output_dir</code>	Directory where files will be saved.
<code>progress</code>	Show progress bar?

Value

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

Kernel functions available

- `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_min()`: returns the minimum of the neighborhood's values.
- `w_max()`: returns the maximum of the neighborhood's values.
- `w_var()`: returns the variance of the neighborhood's values.
- `w_modal()`: returns the modal of the neighborhood's values.

Note

The main sits classification workflow has the following steps:

1. `sits_cube`: selects a ARD image collection from a cloud provider.
2. `sits_cube_copy`: copies an ARD image collection from a cloud provider to a local directory for faster processing.
3. `sits_regularize`: create a regular data cube from an ARD image collection.

4. `sits_apply`: create new indices by combining bands of a regular data cube (optional).
5. `sits_get_data`: extract time series from a regular data cube based on user-provided labelled samples.
6. `sits_train`: train a machine learning model based on image time series.
7. `sits_classify`: classify a data cube using a machine learning model and obtain a probability cube.
8. `sits_smooth`: post-process a probability cube using a spatial smoother to remove outliers and increase spatial consistency.
9. `sits_label_classification`: produce a classified map by selecting the label with the highest probability from a smoothed cube.

`sits_apply()` allows 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()`).

Examples of valid expressions:

1. $NDVI = (B08 - B04) / (B08 + B04)$ for Sentinel-2 images.
2. $EVI = 2.5 * (B05 - B04) / (B05 + 6 * B04 - 7.5 * B02 + 1)$ for Landsat-8/9 images.
3. $VV_VH_RATIO = VH/VV$ for Sentinel-1 images. In this case, set the normalized parameter to FALSE.
4. $VV_DB = 10 * \log_{10}(VV)$ to convert Sentinel-1 RTC images available in Planetary Computer to decibels. Also, set the normalized parameter to FALSE.

`sits_apply()` also accepts a predefined set of kernel functions (see below) that can be applied to pixels considering its neighborhood. The function considers a neighborhood of a pixel as a set of pixels equidistant to it (including itself). This neighborhood forms 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. If all pixels in the window are NA the result will be NA.

By default, the indexes generated by `sits_apply()` function are normalized between -1 and 1, scaled by a factor of 0.0001. Normalized indexes are saved as INT2S (Integer with sign). If the normalized parameter is FALSE, no scaling factor will be applied and the index will be saved as FLT4S (signed float) and the values will vary between -3.4e+38 and 3.4e+38.

Author(s)

Rolf Simoes, <rolfsimoes@gmail.com>

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

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

Examples

```
if (sits_run_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)))

  # Example of generation texture band with variance
  # Create a data cube from local files
  data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
  cube <- sits_cube(
    source = "BDC",
    collection = "MOD13Q1-6.1",
    data_dir = data_dir
  )

  # Generate a texture images with variance in NDVI images
  cube_texture <- sits_apply(
    data = cube,
    NDVITEXTURE = w_median(NDVI),
    window_size = 5,
    output_dir = tempdir()
  )
}
```

sits_as_sf

Return a sits_tibble or raster_cube as an sf object.

Description

Converts a sits_tibble or raster_cube as an sf object.

Usage

```
sits_as_sf(data, ...)

## S3 method for class 'sits'
sits_as_sf(data, ..., crs = "EPSG:4326", as_crs = NULL)

## S3 method for class 'raster_cube'
sits_as_sf(data, ..., as_crs = NULL)

## S3 method for class 'vector_cube'
sits_as_sf(data, ..., as_crs = NULL)

## Default S3 method:
sits_as_sf(data, ...)
```

Arguments

<code>data</code>	A sits tibble or sits cube.
<code>...</code>	Additional parameters.
<code>crs</code>	Input coordinate reference system.
<code>as_crs</code>	Output coordinate reference system.

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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  sf_object <- sits_as_sf(cube)
}
```

sits_as_stars

Convert a data cube into a stars object

Description

Uses the information about files, bands and dates in a data cube to produce an object of class stars. User has to select a tile from the data cube. By default, all bands and dates are included in the stars object. Users can select bands and dates.

Usage

```
sits_as_stars(
  cube,
  tile = cube[1L, ]$tile,
  bands = NULL,
  dates = NULL,
  proxy = FALSE
)
```

Arguments

cube	A sits cube.
tile	Tile of the data cube.
bands	Bands of the data cube to be part of stars object.
dates	Dates of the data cube to be part of stars object.
proxy	Produce a stars proxy object.

Value

An space-time stars object.

Note

By default, the stars object will be loaded in memory. This can result in heavy memory usage. To produce a stars.proxy object, users have to select a single date, since stars does not allow proxy objects to be created with two dimensions.

Author(s)

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

Examples

```
if (sits_run_examples()) {
  # convert sits cube to an sf object (polygon)
  data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
  cube <- sits_cube(
    source = "BDC",
    collection = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  stars_object <- sits_as_stars(cube)
}
```

sits_as_terra

Convert a data cube into a Spatial Raster object from terra

Description

Uses the information about files, bands and dates in a data cube to produce an object of class terra. User has to select a tile and a date from the data cube. By default, all bands are included in the terra object. Users can select bands.

Usage

```
sits_as_terra(cube, tile = cube[1L, ]$tile, ...)

## S3 method for class 'raster_cube'
sits_as_terra(cube, tile = cube[1L, ]$tile, ..., bands = NULL, date = NULL)

## S3 method for class 'probs_cube'
sits_as_terra(cube, tile = cube[1L, ]$tile, ...)

## S3 method for class 'class_cube'
sits_as_terra(cube, tile = cube[1L, ]$tile, ...)

## S3 method for class 'variance_cube'
sits_as_terra(cube, tile = cube[1L, ]$tile, ...)

## S3 method for class 'uncertainty_cube'
sits_as_terra(cube, tile = cube[1L, ]$tile, ...)
```

Arguments

<code>cube</code>	A sits cube.
<code>tile</code>	Tile of the data cube.
<code>...</code>	Other parameters for specific types of data cubes.
<code>bands</code>	Bands of the data cube to be part of terra object.
<code>date</code>	Date of the data cube to be part of terra object.

Value

An Spatial Raster object from terra.

Author(s)

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

Examples

```
if (sits_run_examples()) {
  # convert sits cube to an sf object (polygon)
  data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
  cube <- sits_cube(
    source = "BDC",
    collection = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  spat_raster <- sits_as_terra(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 'raster_cube'
sits_bands(x)

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

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

## Default S3 method:
sits_bands(x)

sits_bands(x) <- value

## S3 replacement method for class 'sits'
sits_bands(x) <- value

## S3 replacement method for class 'raster_cube'
sits_bands(x) <- value

## Default S3 replacement method:
sits_bands(x) <- value
```

Arguments

<code>x</code>	Valid sits tibble (time series or a cube)
<code>value</code>	New value for the bands

Value

A vector with the names of the bands.

Author(s)

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

Rolf Simoes, <rolfsimoes@gmail.com>

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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # Get the bands from a data cube
  bands <- sits_bands(cube)
  # Get the bands from a sits tibble
  bands <- sits_bands(samples_modis_ndvi)
  # Get the bands from patterns
  bands <- sits_bands(sits_patterns(samples_modis_ndvi))
  # Get the bands from ML model
  rf_model <- sits_train(samples_modis_ndvi, sits_rfor())
  bands <- sits_bands(rf_model)
  # Set the bands for a SITS time series
  sits_bands(samples_modis_ndvi) <- "NDVI2"
  # Set the bands for a SITS cube
  sits_bands(cube) <- "NDVI2"
}
```

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, ..., crs = "EPSG:4326", as_crs = NULL)
```

```
## S3 method for class 'sits'
```

```
sits_bbox(data, ..., crs = "EPSG:4326", as_crs = NULL)
```

```
## S3 method for class 'raster_cube'
```

```
sits_bbox(data, ..., as_crs = NULL)
```

```
## S3 method for class 'tbl_df'
```

```
sits_bbox(data, ..., crs = "EPSG:4326", as_crs = NULL)

## Default S3 method:
sits_bbox(data, ..., crs = "EPSG:4326", as_crs = NULL)
```

Arguments

<code>data</code>	samples (class "sits") or cube.
<code>...</code>	parameters for specific types
<code>crs</code>	CRS of the time series.
<code>as_crs</code>	CRS to project the resulting bbox.

Value

A bbox.

Note

Time series in `sits` are associated with lat/long values in WGS84, while each data cubes is associated to a cartographic projection. To obtain the bounding box of a data cube in a different projection than the original, use the `as_crs` parameter.

Author(s)

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

Rolf Simoes, <rolfsimoes@gmail.com>

Examples

```
if (sits_run_examples()) {
  # get the bbox of a set of samples
  sits_bbox(samples_modis_ndvi)
  # get the bbox of a cube in WGS84
  data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
  cube <- sits_cube(
    source = "BDC",
    collection = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  sits_bbox(cube, as_crs = "EPSG:4326")
}
```

sits_classify	<i>Classify time series or data cubes</i>
---------------	---

Description

This function classifies a set of time series or data cube using a trained model prediction model created by [sits_train](#).

The `sits_classify` function takes three types of data as input and produce there types of output. Users should call [sits_classify](#) but be aware that the parameters are different for each type of input.

- [sits_classify.sits](#) is called when the input is a set of time series. The output is the same set with the additional column predicted.
- [sits_classify.raster_cube](#) is called when the input is a regular raster data cube. The output is a probability cube, which has the same tiles as the raster cube. Each tile contains a multiband image; each band contains the probability that each pixel belongs to a given class. Probability cubes are objects of class "probs_cube".
- [sits_classify.vector_cube](#) is called for vector data cubes. Vector data cubes are produced when closed regions are obtained from raster data cubes using [sits_segment](#). Classification of a vector data cube produces a vector data structure with additional columns expressing the class probabilities for each object. Probability cubes for vector data cubes are objects of class "probs_vector_cube".

Usage

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

## S3 method for class 'tbl_df'
sits_classify(data, ml_model, ...)

## S3 method for class 'derived_cube'
sits_classify(data, ml_model, ...)

## Default S3 method:
sits_classify(data, ml_model, ...)
```

Arguments

<code>data</code>	Data cube (tibble of class "raster_cube")
<code>ml_model</code>	R model trained by sits_train
<code>...</code>	Other parameters for specific functions.

Value

Time series with predicted labels for each point (tibble of class "sits") or a data cube with probabilities for each class (tibble of class "probs_cube").

Note

The main sits classification workflow has the following steps:

1. `sits_cube`: selects a ARD image collection from a cloud provider.
2. `sits_cube_copy`: copies an ARD image collection from a cloud provider to a local directory for faster processing.
3. `sits_regularize`: create a regular data cube from an ARD image collection.
4. `sits_apply`: create new indices by combining bands of a regular data cube (optional).
5. `sits_get_data`: extract time series from a regular data cube based on user-provided labelled samples.
6. `sits_train`: train a machine learning model based on image time series.
7. `sits_classify`: classify a data cube using a machine learning model and obtain a probability cube.
8. `sits_smooth`: post-process a probability cube using a spatial smoother to remove outliers and increase spatial consistency.
9. `sits_label_classification`: produce a classified map by selecting the label with the highest probability from a smoothed cube.

SITS supports the following models:

- support vector machines: `sits_svm`;
- random forests: `sits_rfor`;
- extreme gradient boosting: `sits_xgboost`;
- light gradient boosting: `sits_lightgbm`;
- multi-layer perceptrons: `sits_mlp`;
- temporal CNN: `sits_tempcnn`;
- residual network encoders: `sits_resnet`;
- LSTM with convolutional networks: `sits_lstm_fcn`;
- temporal self-attention encoders: `sits_lighttae` and `sits_tae`.

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

Author(s)

Rolf Simoes, <rolfsimoes@gmail.com>

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

Felipe Carvalho, <lipecaso@gmail.com>

Felipe Carlos, <efelipecarlos@gmail.com>

sits_classify.raster_cube

Classify a regular raster cube

Description

Called when the input is a regular raster data cube. The output is a probability cube, which has the same tiles as the raster cube. Each tile contains a multiband image; each band contains the probability that each pixel belongs to a given class. Probability cubes are objects of class "probs_cube".

Usage

```
## S3 method for class 'raster_cube'
sits_classify(
  data,
  ml_model,
  ...,
  roi = NULL,
  exclusion_mask = NULL,
  filter_fn = NULL,
  impute_fn = impute_linear(),
  start_date = NULL,
  end_date = NULL,
  memsize = 8L,
  multicores = 2L,
  gpu_memory = 4L,
  batch_size = 2L^gpu_memory,
  output_dir,
  version = "v1",
  verbose = FALSE,
  progress = TRUE
)
```

Arguments

data	Data cube (tibble of class "raster_cube")
ml_model	R model trained by sits_train
...	Other parameters for specific functions.
roi	Region of interest (either an sf object, shapefile, or a numeric vector in WGS 84 with named XY values ("xmin", "xmax", "ymin", "ymax") or named lat/long values ("lon_min", "lat_min", "lon_max", "lat_max")).
exclusion_mask	Areas to be excluded from the classification process. It can be defined by a sf object or by a shapefile.
filter_fn	Smoothing filter to be applied - optional (closure containing object of class "function").

impute_fn	Imputation function to remove NA.
start_date	Starting date for the classification (Date in YYYY-MM-DD format).
end_date	Ending date for the classification (Date in YYYY-MM-DD format).
memsize	Memory available for classification in GB (integer, min = 1, max = 16384).
multicores	Number of cores to be used for classification (integer, min = 1, max = 2048).
gpu_memory	Memory available in GPU in GB (default = 4)
batch_size	Batch size for GPU classification.
output_dir	Directory for output file.
version	Version of the output.
verbose	Logical: print information about processing time?
progress	Logical: Show progress bar?

Value

Time series with predicted labels for each point (tibble of class "sits") or a data cube with probabilities for each class (tibble of class "probs_cube").

Note

The roi parameter defines a region of interest. Either:

1. A path to a shapefile with polygons;
2. An sf object with POLYGON or MULTIPOLYGON geometry;
3. A named XY vector (xmin, xmax, ymin, ymax) in WGS84;
4. A name lat/long vector (lon_min, lon_max, lat_min, lat_max);

Parameter filter_fn parameter specifies a smoothing filter to be applied to each time series for reducing noise. Currently, options are Savitzky-Golay (see [sits_sgolay](#)) and Whittaker (see [sits_whittaker](#)) filters.

Parameter impute_fn defines a 1D function that will be used to interpolate NA values in each time series. Currently sits supports the [impute_linear](#) function, but users can define imputation functions which are defined externally.

Parameter memsize controls the amount of memory available for classification, while multicores defines the number of cores used for processing. We recommend using as much memory as possible.

Parameter exclusion_mask defines a region that will not be classify. The region can be defined by multiple polygons. Either a path to a shapefile with polygons or a sf object with POLYGON or MULTIPOLYGON geometry;

When using a GPU for deep learning, gpu_memory indicates the memory of the graphics card which is available for processing. The parameter batch_size defines the size of the matrix (measured in number of rows) which is sent to the GPU for classification. Users can test different values of batch_size to find out which one best fits their GPU architecture.

It is not possible to have an exact idea of the size of Deep Learning models in GPU memory, as the complexity of the model and factors such as CUDA Context increase the size of the model in

memory. Therefore, we recommend that you leave at least 1GB free on the video card to store the Deep Learning model that will be used.

For users of Apple M3 chips or similar with a Neural Engine, be aware that these chips share memory between the GPU and the CPU. Tests indicate that the memsize should be set to half to the total memory and the batch_size parameter should be a small number (we suggest the value of 64). Be aware that increasing these parameters may lead to memory conflicts.

Examples

```
if (sits_run_examples()) {
  # Retrieve the samples for Mato Grosso
  # train a random forest model
  rf_model <- sits_train(samples_modis_ndvi, ml_method = sits_rfor)
  # 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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # classify a data cube
  probs_cube <- sits_classify(
    data = cube,
    ml_model = rf_model,
    output_dir = tempdir(),
    version = "classify"
  )
  # label the probability cube
  label_cube <- sits_label_classification(
    probs_cube,
    output_dir = tempdir(),
    version = "ex_classify"
  )
  # plot the classified image
  plot(label_cube)
}
```

sits_classify.segs_cube

Classify a segmented data cube

Description

This function is called when the input is a vector data cube. Vector data cubes are produced when closed regions are obtained from raster data cubes using [sits_segment](#). Classification of a vector data cube produces a vector data structure with additional columns expressing the class probabilities for each segment. Probability cubes for vector data cubes are objects of class "probs_vector_cube".

Usage

```
## S3 method for class 'vector_cube'
sits_classify(
  data,
  ml_model,
  ...,
  roi = NULL,
  filter_fn = NULL,
  impute_fn = impute_linear(),
  start_date = NULL,
  end_date = NULL,
  memsize = 8L,
  multicores = 2L,
  gpu_memory = 4L,
  batch_size = 2L^gpu_memory,
  output_dir,
  version = "v1",
  n_sam_pol = 15L,
  verbose = FALSE,
  progress = TRUE
)
```

Arguments

<code>data</code>	Data cube (tibble of class "raster_cube")
<code>ml_model</code>	R model trained by sits_train (closure of class "sits_model")
<code>...</code>	Other parameters for specific functions.
<code>roi</code>	Region of interest (either an sf object, shapefile, or a numeric vector in WGS 84 with named XY values ("xmin", "xmax", "ymin", "ymax") or named lat/long values ("lon_min", "lat_min", "lon_max", "lat_max")).
<code>filter_fn</code>	Smoothing filter to be applied - optional (closure containing object of class "function").
<code>impute_fn</code>	Imputation function to remove NA.
<code>start_date</code>	Starting date for the classification (Date in YYYY-MM-DD format).
<code>end_date</code>	Ending date for the classification (Date in YYYY-MM-DD format).
<code>memsize</code>	Memory available for classification in GB (integer, min = 1, max = 16384).
<code>multicores</code>	Number of cores to be used for classification (integer, min = 1, max = 2048).
<code>gpu_memory</code>	Memory available in GPU in GB (default = 4)
<code>batch_size</code>	Batch size for GPU classification.
<code>output_dir</code>	Directory for output file.
<code>version</code>	Version of the output.
<code>n_sam_pol</code>	Number of time series per segment to be classified (integer, min = 10, max = 50).
<code>verbose</code>	Logical: print information about processing time?
<code>progress</code>	Logical: Show progress bar?

Value

Vector data cube with probabilities for each class included in new columns of the tibble. (tibble of class "probs_vector_cube").

Note

The `roi` parameter defines a region of interest. Either:

1. A path to a shapefile with polygons;
2. An `sf` object with POLYGON or MULTIPOLYGON geometry;
3. A named XY vector (`xmin`, `xmax`, `ymin`, `ymax`) in WGS84;
4. A name lat/long vector (`lon_min`, `lon_max`, `lat_min`, `lat_max`);

Parameter `filter_fn` parameter specifies a smoothing filter to be applied to each time series for reducing noise. Currently, options are Savitzky-Golay (see [sits_sgolay](#)) and Whittaker (see [sits_whittaker](#)) filters.

Parameter `impute_fn` defines a 1D function that will be used to interpolate NA values in each time series. Currently sits supports the [impute_linear](#) function, but users can define imputation functions which are defined externally.

Parameter `memsize` controls the amount of memory available for classification, while `multicores` defines the number of cores used for processing. We recommend using as much memory as possible.

For classifying vector data cubes created by [sits_segment](#), `n_sam_pol` controls is the number of time series to be classified per segment.

When using a GPU for deep learning, `gpu_memory` indicates the memory of the graphics card which is available for processing. The parameter `batch_size` defines the size of the matrix (measured in number of rows) which is sent to the GPU for classification. Users can test different values of `batch_size` to find out which one best fits their GPU architecture.

It is not possible to have an exact idea of the size of Deep Learning models in GPU memory, as the complexity of the model and factors such as CUDA Context increase the size of the model in memory. Therefore, we recommend that you leave at least 1GB free on the video card to store the Deep Learning model that will be used.

For users of Apple M3 chips or similar with a Neural Engine, be aware that these chips share memory between the GPU and the CPU. Tests indicate that the `memsize` should be set to half to the total memory and the `batch_size` parameter should be a small number (we suggest the value of 64). Be aware that increasing these parameters may lead to memory conflicts.

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

Examples

```
if (sits_run_examples()) {
  # train a random forest model
  rf_model <- sits_train(samples_modis_ndvi, ml_method = sits_rfor)
  # 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 = "MOD13Q1-6.1",
  data_dir = data_dir
)
# segment the image
segments <- sits_segment(
  cube = cube,
  seg_fn = sits_slic(
    step = 5,
    compactness = 1,
    dist_fun = "euclidean",
    avg_fun = "median",
    iter = 50,
    minarea = 10,
    verbose = FALSE
  ),
  output_dir = tempdir()
)
# Create a classified vector cube
probs_segs <- sits_classify(
  data = segments,
  ml_model = rf_model,
  output_dir = tempdir(),
  multicores = 4,
  n_sam_pol = 15,
  version = "segs"
)
# Create a labelled vector cube
class_segs <- sits_label_classification(
  cube = probs_segs,
  output_dir = tempdir(),
  multicores = 2,
  memsize = 4,
  version = "segs_classify"
)
# plot class_segs
plot(class_segs)
}

```

sits_classify.sits	<i>Classify a set of time series</i>
--------------------	--------------------------------------

Description

[sits_classify.sits](#) is called when the input is a set of time series. The output is the same set with the additional column predicted.

Usage

```
## S3 method for class 'sits'
sits_classify(
  data,
  ml_model,
  ...,
  filter_fn = NULL,
  impute_fn = impute_linear(),
  multicores = 2L,
  gpu_memory = 4L,
  batch_size = 2L^gpu_memory,
  progress = TRUE
)
```

Arguments

<code>data</code>	Set of time series ("sits tibble")
<code>ml_model</code>	R model trained by sits_train (closure of class "sits_model")
<code>...</code>	Other parameters for specific functions.
<code>filter_fn</code>	Smoothing filter to be applied - optional (closure containing object of class "function").
<code>impute_fn</code>	Imputation function to remove NA.
<code>multicores</code>	Number of cores to be used for classification (integer, min = 1, max = 2048).
<code>gpu_memory</code>	Memory available in GPU in GB (default = 4)
<code>batch_size</code>	Batch size for GPU classification.
<code>progress</code>	Logical: Show progress bar?

Value

Time series with predicted labels for each point (tibble of class "sits").

Note

Parameter `filter_fn` specifies a smoothing filter to be applied to each time series for reducing noise. Currently, options are Savitzky-Golay (see [sits_sgolay](#)) and Whittaker (see [sits_whittaker](#)) filters. Note that this parameter should also have been applied to the training set to obtain the model.

Parameter `impute_fn` defines a 1D function that will be used to interpolate NA values in each time series. Currently sits supports the [impute_linear](#) function, but users can define imputation functions which are defined externally.

Parameter `multicores` defines the number of cores used for processing. We recommend using as much memory as possible.

When using a GPU for deep learning, `gpu_memory` indicates the memory of the graphics card which is available for processing. The parameter `batch_size` defines the size of the matrix (measured in number of rows) which is sent to the GPU for classification. Users can test different values of `batch_size` to find out which one best fits their GPU architecture.

It is not possible to have an exact idea of the size of Deep Learning models in GPU memory, as the complexity of the model and factors such as CUDA Context increase the size of the model in memory. Therefore, we recommend that you leave at least 1GB free on the video card to store the Deep Learning model that will be used.

For users of Apple M3 chips or similar with a Neural Engine, be aware that these chips share memory between the GPU and the CPU. Tests indicate that the memsize should be set to half to the total memory and the batch_size parameter should be a small number (we suggest the value of 64). Be aware that increasing these parameters may lead to memory conflicts.

Examples

```
if (sits_run_examples()) {
  # Example of classification of a time series
  # Retrieve the samples for Mato Grosso
  # train a random forest model
  rf_model <- sits_train(samples_modis_ndvi, ml_method = sits_rfor)

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

sits_clean

Cleans a classified map using a local window

Description

Applies a modal function to clean up possible noisy pixels keeping the most frequently values within the neighborhood. In a tie, the first value of the vector is considered. Modal functions applied to classified cubes are useful to remove salt-and-pepper noise in the result.

Usage

```
sits_clean(cube, ...)

## S3 method for class 'class_cube'
sits_clean(
  cube,
  ...,
  window_size = 5L,
  memsize = 4L,
  multicores = 2L,
  output_dir,
  version = "v1-clean",
  progress = TRUE
```

```

)

## S3 method for class 'raster_cube'
sits_clean(cube, ...)

## S3 method for class 'derived_cube'
sits_clean(cube, ...)

## Default S3 method:
sits_clean(cube, ...)

```

Arguments

<code>cube</code>	Classified data cube (tibble of class "class_cube").
<code>...</code>	Specific parameters for specialised functions
<code>window_size</code>	An odd integer representing the size of the sliding window of the modal function (min = 1, max = 15).
<code>memszie</code>	Memory available for classification in GB (integer, min = 1, max = 16384).
<code>multicores</code>	Number of cores to be used for classification (integer, min = 1, max = 2048).
<code>output_dir</code>	Valid directory for output file. (character vector of length 1).
<code>version</code>	Version of the output file (character vector of length 1)
<code>progress</code>	Logical: Show progress bar?

Value

A tibble with an classified map (class = "class_cube").

Note

The `sits_clean` function is useful to further remove classification noise which has not been detected by [sits_smooth](#). It improves the spatial consistency of the classified maps.

Author(s)

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

Examples

```

if (sits_run_examples()) {
  rf_model <- sits_train(samples_modis_ndvi, ml_method = sits_rfor)
  # create a data cube from local files
  data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
  cube <- sits_cube(
    source = "BDC",
    collection = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # classify a data cube
}

```

```
probs_cube <- sits_classify(  
  data = cube,  
  ml_model = rf_model,  
  output_dir = tempdir()  
)  
# label the probability cube  
label_cube <- sits_label_classification(  
  probs_cube,  
  output_dir = tempdir()  
)  
# apply a mode function in the labelled cube  
clean_cube <- sits_clean(  
  cube = label_cube,  
  window_size = 5,  
  output_dir = tempdir(),  
  multicores = 1  
)  
}
```

sits_cluster_clean	<i>Removes labels that are minority in each cluster.</i>
--------------------	--

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 set of time series with additional cluster information produced by sits_cluster_dendro()
---------	--

Value

Tibble with time series (class "sits")

Author(s)

Rolf Simoes, <rolfsimoes@gmail.com>

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_dendro	<i>Find clusters in time series samples</i>
---------------------	---

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 with 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. Its input is 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,
  bands = NULL,
  dist_method = "dtw_basic",
  linkage = "ward.D2",
  k = NULL,
  palette = "RdYlGn",
  ...
)
```

Arguments

samples	Tibble with input set of time series (class "sits").
bands	Bands to be used in the clustering (character vector)
dist_method	One of the supported distances (single char vector) "dtw": DTW with a Sakoe-Chiba constraint. "dtw2": DTW with L2 norm and Sakoe-Chiba constraint. "dtw_basic": A faster DTW with less functionality. "lbk": Keogh's lower bound for DTW. "lbi": Lemire's lower bound for DTW.

linkage	Agglomeration method to be used (single char vector) One of "ward.D", "ward.D2", "single", "complete", "average", "mcquitty", "median" or "centroid".
k	Desired number of clusters (overrides default value)
palette	Color palette as per 'grDevices::hcl.pals()' function.
...	Additional parameters to be passed to dtwclust::tsclust() function.

Value

Tibble with "cluster" column (class "sits_cluster").

Note

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

Author(s)

Rolf Simoes, <rolfsimoes@gmail.com>

References

"dtwclust" R package.

Examples

```
if (sits_run_examples()) {
  # default
  clusters <- sits_cluster_dendro(cerrado_2classes)
  # with parameters
  clusters <- sits_cluster_dendro(cerrado_2classes,
    bands = "NDVI", k = 5
  )
}
```

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_cluster_dendro\(\)](#).

Value

A matrix containing frequencies of labels in clusters.

Author(s)

Rolf Simoes, <rolfsimoes@gmail.com>

Examples

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

sits_colors

Function to retrieve sits color table

Description

Returns the default color table.

Usage

```
sits_colors(legend = NULL)
```

Arguments

legend One of the accepted legends in sits

Value

A tibble with color names and values

Note

SITS has a predefined color palette with 238 class names. These colors are grouped by typical legends used by the Earth observation community, which include “IGBP”, “UMD”, “ESA_CCI_LC”, and “WORLDCOVER”. Use [sits_colors_show](#) to see a specific palette. The default color table can be extended using [sits_colors_set](#).

Author(s)

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

Examples

```
if (sits_run_examples()) {
  # return the names of all colors supported by SITS
  sits_colors()
}
```

sits_colors_qgis	<i>Function to save color table as QML style for data cube</i>
------------------	--

Description

Saves a color table associated to a classified data cube as a QGIS style file

Usage

```
sits_colors_qgis(cube, file)
```

Arguments

cube	a classified data cube
file	a QGIS style file to be written to

Value

No return, called for side effects

Author(s)

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

Examples

```
if (sits_run_examples()) {
  data_dir <- system.file("extdata/raster/classif", package = "sits")
  ro_class <- sits_cube(
    source = "MPC",
    collection = "SENTINEL-2-L2A",
    data_dir = data_dir,
    parse_info = c(
      "X1", "X2", "tile", "start_date", "end_date",
      "band", "version"
    ),
    bands = "class",
    labels = c(
      "1" = "Clear_Cut_Burned_Area",
      "2" = "Clear_Cut_Bare_Soil",
      "3" = "Clear_Cut_Vegetation",
      "4" = "Forest"
    )
  )
}
```

```

    )
    qml_file <- paste0(tempdir(), "/qgis.qml")
    sits_colors_qgis(ro_class, qml_file)
  }

```

sits_colors_reset	<i>Function to reset sits color table</i>
-------------------	---

Description

Resets the color table

Usage

```
sits_colors_reset()
```

Value

No return, called for side effects

Author(s)

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

Examples

```

if (sits_run_examples()) {
  # reset the default colors supported by SITS
  sits_colors_reset()
}

```

sits_colors_set	<i>Function to set sits color table</i>
-----------------	---

Description

Includes new colors in the SITS color sets. If the colors exist, replace them with the new HEX value. Optionally, the new colors can be associated to a legend. In this case, the new legend name should be informed. The colors parameter should be a data.frame or a tibble with name and HEX code. Colour names should be one character string only. Composite names need to be combined with underscores (e.g., use "Snow_and_Ice" and not "Snow and Ice").

This function changes the global sits color table and the global set of sits color legends. To undo these effects, please use "sits_colors_reset()".

Usage

```
sits_colors_set(colors, legend = NULL)
```

Arguments

colors	New color table (a tibble or data.frame with name and HEX code)
legend	Legend associated to the color table (optional)

Value

A modified sits color table (invisible)

Author(s)

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

Examples

```
if (sits_run_examples()) {
  # Define a color table based on the Anderson Land Classification System
  us_nlcd <- tibble::tibble(name = character(), color = character())
  us_nlcd <- us_nlcd |>
    tibble::add_row(name = "Urban_Built_Up", color = "#85929E") |>
    tibble::add_row(name = "Agricultural_Land", color = "#F0B27A") |>
    tibble::add_row(name = "Rangeland", color = "#F1C40F") |>
    tibble::add_row(name = "Forest_Land", color = "#27AE60") |>
    tibble::add_row(name = "Water", color = "#2980B9") |>
    tibble::add_row(name = "Wetland", color = "#D4E6F1") |>
    tibble::add_row(name = "Barren_Land", color = "#FDEBD0") |>
    tibble::add_row(name = "Tundra", color = "#EBDEF0") |>
    tibble::add_row(name = "Snow_and_Ice", color = "#F7F9F9")

  # Load the color table into `sits`
  sits_colors_set(colors = us_nlcd, legend = "US_NLCD")

  # Show the new color table used by sits
  sits_colors_show("US_NLCD")

  # Change colors in the sits global color table
  # First show the default colors for the UMD legend
  sits_colors_show("UMD")
  # Then change some colors associated to the UMD legend
  mycolors <- tibble::tibble(name = character(), color = character())
  mycolors <- mycolors |>
    tibble::add_row(name = "Savannas", color = "#F8C471") |>
    tibble::add_row(name = "Grasslands", color = "#ABEBC6")
  sits_colors_set(colors = mycolors)
  # Notice that the UMD colors change
  sits_colors_show("UMD")
  # Reset the color table
  sits_colors_reset()
  # Show the default colors for the UMD legend
  sits_colors_show("UMD")
}
```

sits_colors_show	<i>Function to show colors in SITS</i>
------------------	--

Description

Shows the default SITS colors

Usage

```
sits_colors_show(legend = NULL, font_family = "sans")
```

Arguments

legend	One of the accepted legends in sits
font_family	A font family loaded in SITS

Value

no return, called for side effects

Author(s)

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

Examples

```
if (sits_run_examples()) {  
  # show the colors supported by SITS  
  sits_colors_show()  
}
```

sits_combine_predictions	<i>Estimate ensemble prediction based on list of probs cubes</i>
--------------------------	--

Description

Calculate an ensemble predictor based a list of probability cubes. The function combines the output of two or more models to derive a weighted average. The supported types of ensemble predictors are 'average' and 'uncertainty'. In the latter case, the uncertainty cubes need to be provided using param uncert_cubes.

Usage

```
sits_combine_predictions(cubes, type = "average", ...)
```

```
## S3 method for class 'average'
```

```
sits_combine_predictions(
```

```
  cubes,
```

```
  type = "average",
```

```
  ...,
```

```
  weights = NULL,
```

```
  memsize = 8L,
```

```
  multicores = 2L,
```

```
  output_dir,
```

```
  version = "v1",
```

```
  progress = FALSE
```

```
)
```

```
## S3 method for class 'uncertainty'
```

```
sits_combine_predictions(
```

```
  cubes,
```

```
  type = "uncertainty",
```

```
  ...,
```

```
  uncert_cubes,
```

```
  memsize = 8L,
```

```
  multicores = 2L,
```

```
  output_dir,
```

```
  version = "v1",
```

```
  progress = FALSE
```

```
)
```

```
## Default S3 method:
```

```
sits_combine_predictions(cubes, type, ...)
```

Arguments

cubes	List of probability data cubes (class "probs_cube")
type	Method to measure uncertainty. One of "average" or "uncertainty"
...	Parameters for specific functions.
weights	Weights for averaging (numeric vector).
memsize	Memory available for classification in GB (integer, min = 1, max = 16384).
multicores	Number of cores to be used for classification (integer, min = 1, max = 2048).
output_dir	Valid directory for output file. (character vector of length 1).
version	Version of the output (character vector of length 1).
progress	Set progress bar?
uncert_cubes	Uncertainty cubes to be used as local weights when type = "uncertainty" is selected (list of tibbles with class "uncertainty_cube")

Value

A combined probability cube (tibble of class "probs_cube").

Note

The distribution of class probabilities produced by machine learning models such as random forest is quite different from that produced by deep learning models such as temporal CNN. Combining the result of two different models is recommended to remove possible bias induced by a single model.

By default, the function takes the average of the class probabilities of two or more model results. If desired, users can use the uncertainty estimates for each results to compute the weights for each pixel. In this case, the uncertainties produced by the models for each pixel are used to compute the weights for producing the combined result.

Author(s)

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

Rolf Simoes, <rolfsimoes@gmail.com>

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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # create a random forest model
  rfor_model <- sits_train(samples_modis_ndvi, sits_rfor())
  # classify a data cube using rfor model
  probs_rfor_cube <- sits_classify(
    data = cube, ml_model = rfor_model, output_dir = tempdir(),
    version = "rfor"
  )
  # create an SVM model
  svm_model <- sits_train(samples_modis_ndvi, sits_svm())
  # classify a data cube using SVM model
  probs_svm_cube <- sits_classify(
    data = cube, ml_model = svm_model, output_dir = tempdir(),
    version = "svm"
  )
  # create a list of predictions to be combined
  pred_cubes <- list(probs_rfor_cube, probs_svm_cube)
  # combine predictions
  comb_probs_cube <- sits_combine_predictions(
    pred_cubes,
    output_dir = tempdir()
  )
}
```

```

    # plot the resulting combined prediction cube
    plot(comb_probs_cube)
}

```

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. Perform confidence sampling.

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

Usage

```

sits_confidence_sampling(
  probs_cube,
  n = 20L,
  min_margin = 0.5,
  sampling_window = 10L,
  multicores = 2L,
  memsize = 4L,
  progress = TRUE
)

```

Arguments

probs_cube	A smoothed probability cube. See sits_classify and sits_smooth .
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.
multicores	Number of workers for parallel processing (integer, min = 1, max = 2048).
memsize	Maximum overall memory (in GB) to run the function.
progress	Show progress bar?

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, <rolfsimoes@gmail.com>
 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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # build a random forest model
  rfor_model <- sits_train(samples_modis_ndvi, ml_method = sits_rfor())
  # classify the cube
  probs_cube <- sits_classify(
    data = cube, ml_model = rfor_model, output_dir = tempdir()
  )
  # obtain a new set of samples for active learning
  # the samples are located in uncertain places
  new_samples <- sits_confidence_sampling(probs_cube)
}
```

sits_config

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 by the user.

Usage

```
sits_config(config_user_file = NULL)
```

Arguments

`config_user_file`
 YAML user configuration file (character vector of a file with "yaml" extension)

Details

Users can provide additional configuration files, by specifying the location of their file in the environmental variable `SITS_CONFIG_USER_FILE` or as parameter to this function.

To see the key entries and contents of the current configuration values, use `link[sits]{sits_config_show()}`.

Value

Called for side effects

Author(s)

Rolf Simoes, <rolfsimoes@gmail.com>

Examples

```
yaml_user_file <- system.file("extdata/config_user_example.yaml",
  package = "sits"
)
sits_config(config_user_file = yaml_user_file)
```

<code>sits_config_show</code>	<i>Show current sits configuration</i>
-------------------------------	--

Description

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 and collection.

Usage

```
sits_config_show()
```

Value

No return value, called for side effects.

Examples

```
sits_config_show()
```

sits_config_user_file *Create a user configuration file.*

Description

Creates a user configuration file.

Usage

```
sits_config_user_file(file_path, overwrite = FALSE)
```

Arguments

file_path	file to store the user configuration file
overwrite	replace current configuration file?

Value

Called for side effects

Examples

```
user_file <- paste0(tempdir(), "/my_config_file.yml")
sits_config_user_file(user_file)
```

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. Available options are:

- To create data cubes from providers which support the STAC protocol, use [sits_cube.stac_cube](#).
- To create raster data cubes from local image files, use [sits_cube.local_cube](#).
- To create vector data cubes from local image and vector files, use [sits_cube.vector_cube](#).
- To create raster data cubes from local image files which have been classified or labelled, use [sits_cube.results_cube](#).

Usage

```
sits_cube(source, collection, ...)
```

Arguments

source	Data source: one of "AWS", "BDC", "CDSE", "DEAFRICA", "DEAUSTRALIA", "HLS", "PLANETSCOPE", "MPC", "SDC" or "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.

Value

A tibble describing the contents of a data cube.

Note

The main `sits` classification workflow has the following steps:

1. [sits_cube](#): selects a ARD image collection from a cloud provider.
2. [sits_cube_copy](#): copies an ARD image collection from a cloud provider to a local directory for faster processing.
3. [sits_regularize](#): create a regular data cube from an ARD image collection.
4. [sits_apply](#): create new indices by combining bands of a regular data cube (optional).
5. [sits_get_data](#): extract time series from a regular data cube based on user-provided labelled samples.
6. [sits_train](#): train a machine learning model based on image time series.
7. [sits_classify](#): classify a data cube using a machine learning model and obtain a probability cube.
8. [sits_smooth](#): post-process a probability cube using a spatial smoother to remove outliers and increase spatial consistency.
9. [sits_label_classification](#): produce a classified map by selecting the label with the highest probability from a smoothed cube.

The following cloud providers are supported, based on the STAC protocol: Amazon Web Services (AWS), Brazil Data Cube (BDC), Copernicus Data Space Ecosystem (CDSE), Digital Earth Africa (DEAFRICA), Digital Earth Australia (DEAUSTRALIA), Microsoft Planetary Computer (MPC), Nasa Harmonized Landsat/Sentinel (HLS), Swiss Data Cube (SDC), TERRASCOPE and USGS Landsat (USGS). Data cubes can also be created using local files.

In `sits`, a data cube is represented as a tibble with metadata describing a set of image files obtained from cloud providers. It contains information about each individual file.

A data cube in `sits` is:

- A set of images organized in tiles of a grid system (e.g., MGRS).
- Each tile contains single-band images in a unique zone of the coordinate system (e.g, tile 20LMR in MGRS grid) covering the period between `start_date` and `end_date`.
- Each image of a tile is associated to a unique temporal interval. All intervals share the same spectral bands.
- Different tiles may cover different zones of the same grid system.

A regular data cube is a data cube where:

- All tiles share the same set of regular temporal intervals.
- All tiles share the same spectral bands and indices.
- All images have the same spatial resolution.
- Each location in a tile is associated a set of multi-band time series.
- For each tile, interval and band, the cube is reduce to a 2D image.

Author(s)

Felipe Carlos, <efelipeCarlos@gmail.com>

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

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

Rolf Simoes, <rolfsimoes@gmail.com>

Examples

```
if (sits_run_examples()) {
  # --- Access to the Brazil Data Cube
  # create a raster cube file based on the information in the BDC
  cbers_tile <- sits_cube(
    source = "BDC",
    collection = "CBERS-WFI-16D",
    bands = c("NDVI", "EVI"),
    tiles = "007004",
    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_deafrica <- sits_cube(
    source = "DEAFRICA",
    collection = "SENTINEL-2-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"
  )
  # --- Create a cube based on a local MODIS data
  # MODIS local files have names such as
  # "TERRA_MODIS_012010_NDVI_2013-09-14.jp2"
  # see the parse_info parameter as an example on how to
  # decode local files
  data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
}
```

```

modis_cube <- sits_cube(
  source = "BDC",
  collection = "MOD13Q1-6.1",
  data_dir = data_dir,
  parse_info = c("satellite", "sensor", "tile", "band", "date")
)
}

```

sits_cube.local_cube *Create sits cubes from cubes in flat files in a local*

Description

Creates data cubes based on files on local directory. Assumes users have downloaded the data from a known cloud collection or the data has been created by sits.

Usage

```

## S3 method for class 'local_cube'
sits_cube(
  source,
  collection,
  ...,
  bands = NULL,
  tiles = NULL,
  start_date = NULL,
  end_date = NULL,
  data_dir,
  parse_info = c("X1", "X2", "tile", "band", "date"),
  delim = "_",
  multicores = 2L,
  progress = TRUE
)

```

Arguments

source	Data source: one of "AWS", "BDC", "CDSE", "DEAFRICA", "DEAUSTRALIA", "HLS", "PLANETSCOPE", "MPC", "SDC" or "USGS". This is the source from which the data has been downloaded.
collection	Image collection in data source. To find out the supported collections, use sits_list_collections() .
...	Other parameters to be passed for specific types.
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).
start_date, end_date	Initial and final dates to include images from the collection in the cube (optional). (Date in YYYY-MM-DD format).

<code>data_dir</code>	Local directory where images are stored.
<code>parse_info</code>	Parsing information for local files.
<code>delim</code>	Delimiter for parsing local files (default = "_")
<code>multicores</code>	Number of workers for parallel processing (integer, min = 1, max = 2048).
<code>progress</code>	Logical: show a progress bar?

Value

A tibble describing the contents of a data cube.

Note

To create a cube from local files, please inform:

- `source`: The data provider from which the data was downloaded (e.g, "BDC", "MPC");
- `collection`: The collection from which the data comes from. (e.g., "SENTINEL-2-L2A" for the Sentinel-2 MPC collection level 2A);
- `data_dir`: The local directory where the image files are stored.
- `parse_info`: Defines how to extract metadata from file names by specifying the order and meaning of each part, separated by the "delim" character. Default value is `c("X1", "X2", "tile", "band", "date")`.
- `delim`: The delimiter character used to separate components in the file names. Default is "_".

Please ensure that local files meet the following requirements:

- All image files must have the same spatial resolution and projection;
- Each file should represent a single image band for a single date;
- File names must include information about the tile, date, and band in their names.
- The `parse_info` parameter tells `sits` how to extract metadata from file names.
- By default the `parse_info` parameter is `c(satellite, sensor, tile, band, date)`.

Example of supported file names are:

- "CBERS-4_WFI_022024_B13_2021-05-15.tif";
- "SENTINEL-1_GRD_30TXL_VV_2023-03-10.tif";
- "LANDSAT-8_OLI_198030_B04_2020-09-12.tif".

When you load a local data cube specifying the source (e.g., AWS, MPC) and collection, `sits` assumes that the data properties (e.g., scale factor, minimum, and maximum values) match those defined for the selected provider. If you are working with custom data from an unsupported source or data that does not follow the standard definitions of providers in `sits`, refer to the Technical Annex of the `sits` online book for guidance on handling such cases (e-sensing.github.io/sitsbook/technical-annex.html).

Examples

```
if (sits_run_examples()) {
  # --- Create a cube based on a local MODIS data
  # MODIS local files have names such as
  # "TERRA_MODIS_012010_NDVI_2013-09-14.jp2"
  # see the parse info parameter as an example on how to
  # decode local files
  data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
  modis_cube <- sits_cube(
    source = "BDC",
    collection = "MOD13Q1-6.1",
    data_dir = data_dir,
    parse_info = c("satellite", "sensor", "tile", "band", "date")
  )
}
```

sits_cube.results_cube

Create a results cube from local files

Description

Creates a data cube from local files produced by sits operations that produces results (such as probs_cubs and class_cubes)

Usage

```
## S3 method for class 'results_cube'
sits_cube(
  source,
  collection,
  ...,
  data_dir,
  tiles = NULL,
  bands,
  labels = NULL,
  parse_info = c("X1", "X2", "tile", "start_date", "end_date", "band", "version"),
  version = "v1",
  delim = "_",
  multicores = 2L,
  memsize = 2L,
  progress = TRUE
)
```

Arguments

source	Data source: one of "AWS", "BDC", "CDSE", "DEAFRICA", "DEAUSTRALIA", "HLS", "PLANETSCOPE", "MPC", "SDC" or "USGS". This is the source from which the original data has been downloaded.
--------	---

collection	Image collection in data source from which the original data has been downloaded. 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
tiles	Tiles from the collection to be included in the cube.
bands	Results bands to be retrieved ("probs", "bayes", "variance", "class", "uncertainty")
labels	Named vector with labels associated to the classes
parse_info	Parsing information for local files (see notes below).
version	Version of the classified and/or labelled files.
delim	Delimiter for parsing local results cubes (default = "_")
multicores	Number of workers for parallel processing (integer, min = 1, max = 2048).
memsizes	Memory available (in GB)
progress	Logical: show a progress bar?

Value

A tibble describing the contents of a data cube.

Note

This function creates result cubes from local files produced by classification or post-classification algorithms. In this case, the `parse_info` is specified differently, and additional parameters are required. The parameter `bands` should be a single character vector with the name associated to the type of result:

- "probs", for probability cubes produced by [sits_classify](#).
- "bayes", for smoothed cubes produced by [sits_smooth](#).
- "entropy" when using [sits_uncertainty](#) to measure entropy in pixel classification.
- "margin" when using [sits_uncertainty](#) to measure probability margin in pixel classification.
- "least" when using [sits_uncertainty](#) to measure difference between 100% and most probable class in pixel classification.
- "class" for cubes produced by [sits_label_classification](#).

For cubes of type "probs", "bayes", "class", the `labels` parameter should be named vector associated to the classification results. For "class" cubes, its names should be integers associated to the values of the raster files that represent the classified cube.

Parameter `parse_info` should contain parsing information to deduce the values of `tile`, `start_date`, `end_date` and `band` from the file name. Default is `c("X1", "X2", "tile", "start_date", "end_date", "band")`. Cubes processed by `sits` adhere to this format.

Examples

```

if (sits_run_examples()) {
  # create a random forest model
  rfor_model <- sits_train(samples_modis_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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # classify a data cube
  probs_cube <- sits_classify(
    data = cube, ml_model = rfor_model, output_dir = tempdir()
  )
  # plot the probability cube
  plot(probs_cube)

  # obtain and name the labels of the local probs cube
  labels <- sits_labels(rfor_model)
  names(labels) <- seq_along(labels)

  # recover the local probability cube
  probs_local_cube <- sits_cube(
    source = "BDC",
    collection = "MOD13Q1-6.1",
    data_dir = tempdir(),
    bands = "probs",
    labels = labels
  )
  # compare the two plots (they should be the same)
  plot(probs_local_cube)

  # smooth the probability cube using Bayesian statistics
  bayes_cube <- sits_smooth(probs_cube, output_dir = tempdir())
  # plot the smoothed cube
  plot(bayes_cube)

  # recover the local smoothed cube
  smooth_local_cube <- sits_cube(
    source = "BDC",
    collection = "MOD13Q1-6.1",
    data_dir = tempdir(),
    bands = "bayes",
    labels = labels
  )
  # compare the two plots (they should be the same)
  plot(smooth_local_cube)

  # label the probability cube
  label_cube <- sits_label_classification(
    bayes_cube,

```

```
        output_dir = tempdir()
    )
    # plot the labelled cube
    plot(label_cube)

    # recover the local classified cube
    class_local_cube <- sits_cube(
        source = "BDC",
        collection = "MOD13Q1-6.1",
        data_dir = tempdir(),
        bands = "class",
        labels = labels
    )
    # compare the two plots (they should be the same)
    plot(class_local_cube)

    # obtain an uncertainty cube with entropy
    entropy_cube <- sits_uncertainty(
        cube = bayes_cube,
        type = "entropy",
        output_dir = tempdir()
    )
    # plot entropy values
    plot(entropy_cube)

    # recover an uncertainty cube with entropy
    entropy_local_cube <- sits_cube(
        source = "BDC",
        collection = "MOD13Q1-6.1",
        data_dir = tempdir(),
        bands = "entropy"
    )
    # plot recovered entropy values
    plot(entropy_local_cube)

    # obtain an uncertainty cube with margin
    margin_cube <- sits_uncertainty(
        cube = bayes_cube,
        type = "margin",
        output_dir = tempdir()
    )
    # plot entropy values
    plot(margin_cube)

    # recover an uncertainty cube with entropy
    margin_local_cube <- sits_cube(
        source = "BDC",
        collection = "MOD13Q1-6.1",
        data_dir = tempdir(),
        bands = "margin"
    )
    # plot recovered entropy values
    plot(margin_local_cube)
```

```
}
```

```
sits_cube.stac_cube      Create data cubes from image collections accessible by STAC
```

Description

Creates a data cube based on spatial and temporal restrictions in collections accessible by the STAC protocol

Usage

```
## S3 method for class 'stac_cube'
sits_cube(
  source,
  collection,
  ...,
  bands = NULL,
  tiles = NULL,
  roi = NULL,
  crs = NULL,
  start_date = NULL,
  end_date = NULL,
  orbit = "descending",
  platform = NULL,
  multicores = 2L,
  progress = TRUE
)
```

Arguments

source	Data source: one of "AWS", "BDC", "CDSE", "DEAFRICA", "DEAUSTRALIA", "HLS", "PLANETSCOPE", "MPC", "SDC" or "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.
bands	Spectral bands and indices to be included in the cube (optional). Use sits_list_collections() to find out the bands available for each collection.
tiles	Tiles from the collection to be included in the cube (see details below).
roi	Region of interest (see below).
crs	The Coordinate Reference System (CRS) of the roi. (see details below).
start_date, end_date	Initial and final dates to include images from the collection in the cube (optional). (Date in YYYY-MM-DD format).
orbit	Orbit name ("ascending", "descending") for SAR cubes.

platform	Optional parameter specifying the platform in case of "LANDSAT" collection. Options: Landsat-5, Landsat-7, Landsat-8, Landsat-9.
multicores	Number of workers for parallel processing (integer, min = 1, max = 2048).
progress	Logical: show a progress bar?

Value

A tibble describing the contents of a data cube.

Note

Data cubes are identified on cloud providers using `sits_cube`. The result of `sits_cube` is a description of the location of the requested data in the cloud provider. No download is done.

To create data cube objects from cloud providers, users need to inform:

- `source`: Name of the cloud provider. One of "AWS", "BDC", "CDSE", "DEAFRICA", "DEAUSTRALIA", "HLS", "PLANETSCOPE", "MPC", "SDC", "TERRASCOPE", or "USGS";
- `collection`: Name of an image collection available in the cloud provider (e.g, "SENTINEL-1-RTC" in MPC). Use `sits_list_collections()` to see which collections are supported;
- `tiles`: A set of tiles defined according to the collection tiling grid (e.g, `c("20LMR", "20LMP")` in MGRS);
- `roi`: Region of interest (see below)

The parameters `bands`, `start_date`, and `end_date` are optional for cubes created from cloud providers.

Either `tiles` or `roi` must be informed. The `tiles` should specify a set of valid tiles for the ARD collection. For example, Landsat data has tiles in WRS2 tiling system and Sentinel-2 data uses the MGRS tiling system. The `roi` parameter is used to select all types of images. This parameter does not crop a region; it only selects images that intersect it.

To define a `roi` use one of:

- A path to a shapefile with polygons;
- A `sfc` or `sf` object from `sf` package;
- A `SpatExtent` object from `terra` package;
- A named vector ("`lon_min`", "`lat_min`", "`lon_max`", "`lat_max`") in WGS84;
- A named vector ("`xmin`", "`xmax`", "`ymin`", "`ymax`") with XY coordinates.

Defining a region of interest using `SpatExtent` or XY values not in WGS84 requires the `crs` parameter to be specified.

To get more details about each provider and collection available in `sits`, please read the online `sits` book (e-sensing.github.io/sitsbook). The chapter Earth Observation data cubes provides a detailed description of all collections you can use with `sits` (e-sensing.github.io/sitsbook/earth-observation-data-cubes.html).

Examples

```

if (sits_run_examples()) {
  # --- 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
  roi <- c(
    "lon_min" = -50.410, "lon_max" = -50.379,
    "lat_min" = -10.1910, "lat_max" = -10.1573
  )
  mpc_cube <- sits_cube(
    source = "MPC",
    collection = "LANDSAT-C2-L2",
    bands = c("BLUE", "RED", "CLOUD"),
    roi = roi,
    start_date = "2005-01-01",
    end_date = "2006-10-28"
  )

  ## Sentinel-1 SAR from MPC
  roi_sar <- c(
    "lon_min" = -50.410, "lon_max" = -50.379,
    "lat_min" = -10.1910, "lat_max" = -10.1573
  )

  s1_cube_open <- sits_cube(
    source = "MPC",
    collection = "SENTINEL-1-GRD",
    bands = c("VV", "VH"),
    orbit = "descending",
    roi = roi_sar,
    start_date = "2020-06-01",
    end_date = "2020-09-28"
  )

  # --- Access to the Brazil Data Cube
  # create a raster cube file based on the information in the BDC
  cbers_tile <- sits_cube(
    source = "BDC",
    collection = "CBERS-WFI-16D",
    bands = c("NDVI", "EVI"),
    tiles = "007004",
    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_deafrica <- sits_cube(
  source = "DEAFRICA",
  collection = "SENTINEL-2-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 Digital Earth Australia
cube_deaustralia <- sits_cube(
  source = "DEAUSTRALIA",
  collection = "GA_LS8CLS9C_GM_CYEAR_3",
  bands = c("RED", "GREEN", "BLUE"),
  roi = c(
    lon_min = 137.15991,
    lon_max = 138.18467,
    lat_min = -33.85777,
    lat_max = -32.56690
  ),
  start_date = "2018-01-01",
  end_date = "2018-12-31"
)
# --- Access to CDSE open data Sentinel 2/2A level 2 collection
# --- remember to set the appropriate environmental variables
# It is recommended that `multicores` be used to accelerate the process.
s2_cube <- sits_cube(
  source = "CDSE",
  collection = "SENTINEL-2-L2A",
  tiles = c("20LKP"),
  bands = c("B04", "B08", "B11"),
  start_date = "2018-07-18",
  end_date = "2019-01-23"
)

## --- Sentinel-1 SAR from CDSE
# --- remember to set the appropriate environmental variables
# --- Obtain a AWS_ACCESS_KEY_ID and AWS_ACCESS_SECRET_KEY_ID
# --- from CDSE
roi_sar <- c(
  "lon_min" = 33.546, "lon_max" = 34.999,
  "lat_min" = 1.427, "lat_max" = 3.726
)
s1_cube_open <- sits_cube(
  source = "CDSE",
  collection = "SENTINEL-1-RTC",
  bands = c("VV", "VH"),

```

```

        orbit = "descending",
        roi = roi_sar,
        start_date = "2020-01-01",
        end_date = "2020-06-10"
    )

    # -- Access to World Cover data (2021) via Terrascope
    cube_terrascope <- sits_cube(
        source = "TERRASCOPE",
        collection = "WORLD-COVER-2021",
        roi = c(
            lon_min = -62.7,
            lon_max = -62.5,
            lat_min = -8.83,
            lat_max = -8.70
        )
    )
}

```

sits_cube.vector_cube *Create a vector cube from local files*

Description

Creates a data cube from local files which include a vector file produced by a segmentation algorithm.

Usage

```

## S3 method for class 'vector_cube'
sits_cube(
  source,
  collection,
  ...,
  raster_cube,
  vector_dir,
  vector_band,
  parse_info = c("X1", "X2", "tile", "start_date", "end_date", "band", "version"),
  version = "v1",
  delim = "_",
  multicores = 2L,
  progress = TRUE
)

```

Arguments

source	Data source: one of "AWS", "BDC", "CDSE", "DEAFRICA", "DEAUSTRALIA", "HLS", "PLANETSCOPE", "MPC", "SDC" or "USGS". This is the source from which the data has been downloaded.
--------	--

collection	Image collection in data source. To find out the supported collections, use sits_list_collections() .
...	Other parameters to be passed for specific types.
raster_cube	Raster cube to be merged with vector data
vector_dir	Local directory where vector files are stored
vector_band	Band for vector cube ("segments", "probs", "class")
parse_info	Parsing information for local image files
version	Version of the classified and/or labelled files.
delim	Delimiter for parsing local files (default = "_")
multicores	Number of workers for parallel processing (integer, min = 1, max = 2048).
progress	Logical: show a progress bar?

Value

A tibble describing the contents of a data cube.

Note

This function creates vector cubes from local files produced by [sits_segment](#), [sits_classify](#) or [sits_label_classification](#) when the output is a vector cube. In this case, `parse_info` is specified differently as `c("X1", "X2", "tile", "start_date", "end_date", "band")`. The parameter `vector_dir` is the directory where the vector file is stored. Parameter `vector_band` is band name of the type of vector cube:

- "segments", for vector cubes produced by [sits_segment](#).
- "probs", for probability cubes produced by [sits_classify.vector_cube](#).
- "entropy" when using [sits_uncertainty.probs_vector_cube](#).
- "class" for cubes produced by [sits_label_classification](#).

Examples

```
if (sits_run_examples()) {
  # --- Create a cube based on a local MODIS data
  # MODIS local files have names such as
  # "TERRA_MODIS_012010_NDVI_2013-09-14.jp2"
  # see the parse info parameter as an example on how to
  # decode local files
  data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
  modis_cube <- sits_cube(
    source = "BDC",
    collection = "MOD13Q1-6.1",
    data_dir = data_dir,
    parse_info = c("satellite", "sensor", "tile", "band", "date")
  )
  # segment the vector cube
  segs_cube <- sits_segment(
    cube = modis_cube,
```

```

    seg_fn = sits_slic(
      step = 10,
      compactness = 1,
      dist_fun = "euclidean",
      avg_fun = "median",
      iter = 30,
      minarea = 10
    ),
    output_dir = tempdir()
  )
plot(segs_cube)

# recover the local segmented cube
local_segs_cube <- sits_cube(
  source = "BDC",
  collection = "MOD13Q1-6.1",
  raster_cube = modis_cube,
  vector_dir = tempdir(),
  vector_band = "segments"
)
# plot the recover model and compare
plot(local_segs_cube)

# classify the segments
# create a random forest model
rfor_model <- sits_train(samples_modis_ndvi, sits_rfor())
probs_vector_cube <- sits_classify(
  data = segs_cube,
  ml_model = rfor_model,
  output_dir = tempdir(),
  n_sam_pol = 10
)
plot(probs_vector_cube)

# recover vector cube
local_probs_vector_cube <- sits_cube(
  source = "BDC",
  collection = "MOD13Q1-6.1",
  raster_cube = modis_cube,
  vector_dir = tempdir(),
  vector_band = "probs"
)
plot(local_probs_vector_cube)

# label the segments
class_vector_cube <- sits_label_classification(
  cube = probs_vector_cube,
  output_dir = tempdir(),
)
plot(class_vector_cube)

# recover vector cube
local_class_vector_cube <- sits_cube(

```

```

        source = "BDC",
        collection = "MOD13Q1-6.1",
        raster_cube = modis_cube,
        vector_dir = tempdir(),
        vector_band = "class"
    )
    plot(local_class_vector_cube)
}

```

sits_cube_copy

Copy the images of a cube to a local directory

Description

This function downloads the images of a cube in parallel. A region of interest (roi) can be provided to crop the images and a resolution (res) to resample the bands. `sits_cube_copy` is useful to improve processing time in the regularization operation.

Usage

```

sits_cube_copy(
  cube,
  roi = NULL,
  res = NULL,
  crs = NULL,
  n_tries = 3L,
  multicores = 2L,
  output_dir,
  progress = TRUE
)

```

Arguments

<code>cube</code>	A data cube (class "raster_cube")
<code>roi</code>	Region of interest. Either: 1. A path to a shapefile with polygons; 2. A sf object from sf package; 3. A named vector ("lon_min", "lat_min", "lon_max", "lat_max") in WGS84; 4. A named vector ("xmin", "xmax", "ymin", "ymax") with XY coordinates in WGS84.
<code>res</code>	An integer value corresponds to the output spatial resolution of the images. Default is NULL.
<code>crs</code>	The Coordinate Reference System (CRS) of the roi. (see details below).
<code>n_tries</code>	Number of attempts to download the same image. Default is 3.
<code>multicores</code>	Number of cores for parallel downloading (integer, min = 1, max = 2048).
<code>output_dir</code>	Output directory where images will be saved. (character vector of length 1).
<code>progress</code>	Logical: show progress bar?

Value

Copy of input data cube (class "raster cube").

The main sits classification workflow has the following steps:

1. `sits_cube`: selects a ARD image collection from a cloud provider.
2. `sits_cube_copy`: copies an ARD image collection from a cloud provider to a local directory for faster processing.
3. `sits_regularize`: create a regular data cube from an ARD image collection.
4. `sits_apply`: create new indices by combining bands of a regular data cube (optional).
5. `sits_get_data`: extract time series from a regular data cube based on user-provided labelled samples.
6. `sits_train`: train a machine learning model based on image time series.
7. `sits_classify`: classify a data cube using a machine learning model and obtain a probability cube.
8. `sits_smooth`: post-process a probability cube using a spatial smoother to remove outliers and increase spatial consistency.
9. `sits_label_classification`: produce a classified map by selecting the label with the highest probability from a smoothed cube.

The `roi` parameter is used to crop cube images. To define a `roi` use one of:

- A path to a shapefile with polygons;
- A `sfc` or `sf` object from `sf` package;
- A `SpatExtent` object from `terra` package;
- A named vector ("`lon_min`", "`lat_min`", "`lon_max`", "`lat_max`") in WGS84;
- A named vector ("`xmin`", "`xmax`", "`ymin`", "`ymax`") with XY coordinates.

Defining a region of interest using `SpatExtent` or XY values not in WGS84 requires the `crs` parameter to be specified.

Author(s)

Felipe Carlos, <efelipeCarlos@gmail.com>

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

Examples

```
if (sits_run_examples()) {
  # Creating a sits cube from BDC
  bdc_cube <- sits_cube(
    source = "BDC",
    collection = "CBERS-WFI-16D",
    tiles = c("007004", "007005"),
    bands = c("B15", "CLOUD"),
    start_date = "2018-01-01",
    end_date = "2018-01-12"
```

```

    )
    # Downloading images to a temporary directory
    cube_local <- sits_cube_copy(
      cube = bdc_cube,
      output_dir = tempdir(),
      roi = c(
        lon_min = -46.5,
        lat_min = -45.5,
        lon_max = -15.5,
        lat_max = -14.6
      ),
      multicores = 2L,
      res = 250
    )
  }

```

`sits_factory_function` *Create a closure for calling functions with and without data*

Description

This function implements the factory method pattern. It creates a generic interface to closures in R so that the functions in the sits package can be called in two different ways: 1. Called directly, passing input data and parameters. 2. Called as second-order values (parameters of another function). In the second case, the call will pass no data values and only pass the parameters for execution.

The factory pattern is used in many situations in the sits package, to allow different alternatives for filtering, pattern creation, training, and cross-validation.

Please see the chapter "Technical Annex" in the sits book for details.

Usage

```
sits_factory_function(data, fun)
```

Arguments

<code>data</code>	Input data.
<code>fun</code>	Function that performs calculation on the input data.

Value

A closure that encapsulates the function applied to data.

Author(s)

Rolf Simoes, <rolfsimoes@gmail.com>

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

Examples

```
# example code
if (sits_run_examples()) {
  # Include a new machine learning function (multiple linear regression)
  # function that returns mlr model based on a sits sample tibble

  sits_mlr <- function(samples = NULL, formula = sits_formula_linear(),
                        n_weights = 20000, maxit = 2000) {
    train_fun <- function(samples) {
      # Data normalization
      ml_stats <- sits_stats(samples)
      train_samples <- sits_predictors(samples)
      train_samples <- sits_pred_normalize(
        pred = train_samples,
        stats = ml_stats
      )
      formula <- formula(train_samples[, -1])
      # call method and return the trained model
      result_mlr <- nnet::multinom(
        formula = formula,
        data = train_samples,
        maxit = maxit,
        MaxNWts = n_weights,
        trace = FALSE,
        na.action = stats::na.fail
      )

      # construct model predict closure function and returns
      predict_fun <- function(values) {
        # retrieve the prediction (values and probs)
        prediction <- tibble::as_tibble(
          stats::predict(result_mlr,
            newdata = values,
            type = "probs"
          )
        )
        return(prediction)
      }
      class(predict_fun) <- c("sits_model", class(predict_fun))
      return(predict_fun)
    }
    result <- sits_factory_function(samples, train_fun)
    return(result)
  }
  # create an mlr model using a set of samples
  mlr_model <- sits_train(samples_modis_ndvi, sits_mlr)
  # classify a point
  point_ndvi <- sits_select(point_mt_6bands, bands = "NDVI")
  point_class <- sits_classify(point_ndvi, mlr_model, multicores = 1)
  plot(point_class)
}
```

sits_filter	<i>Filter time series with smoothing filter</i>
-------------	---

Description

Applies a filter to all bands, using a filter function such as [sits_whittaker](#) or [sits_sgolay](#).

Usage

```
sits_filter(data, filter = sits_whittaker())
```

Arguments

data	Time series (tibble of class "sits") or matrix.
filter	Filter function to be applied.

Value

Filtered time series

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_formula_linear	<i>Define a linear formula for classification models</i>
---------------------	--

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 = -2L:0L)
```

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, <rolfsimoes@gmail.com>

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_ndvi,
    ml_method = sits_svm(formula = sits_formula_logref())
  )
  # classify the point
  point_ndvi <- sits_select(point_mt_6bands, bands = "NDVI")
  # classify the point
  point_class <- sits_classify(
    data = point_ndvi, ml_model = 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 = -2L:0L)
```

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, <rolfsimoes@gmail.com>

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_ndvi,
    ml_method = sits_svm(formula = sits_formula_logref())
  )
  # classify the point
  point_ndvi <- sits_select(point_mt_6bands, bands = "NDVI")
  # classify the point
  point_class <- sits_classify(
    data = point_ndvi, ml_model = ml_model
  )
  plot(point_class)
}

```

sits_geo_dist	<i>Compute the minimum distances among samples and prediction points.</i>
---------------	---

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, n = 1000L, crs = "EPSG:4326")
```

Arguments

<code>samples</code>	Time series (tibble of class "sits").
<code>roi</code>	A region of interest (ROI), either a file containing a shapefile or an "sf" object
<code>n</code>	Maximum number of samples to consider (integer)
<code>crs</code>	CRS of the samples.

Value

A tibble with sample-to-sample and sample-to-prediction distances (object of class "distances").

Note

As pointed out by Meyer and Pebesma, many classifications using machine learning assume that the reference data are independent and well-distributed in space. In practice, many training samples are strongly concentrated in some areas, and many large areas have no samples. This function compares two distributions:

1. The distribution of the spatial distances of reference data to their nearest neighbor (sample-to-sample).
2. The distribution of distances from all points of study area to the nearest reference data point (sample-to-prediction).

Author(s)

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

Rolf Simoes, <rolfsimoes@gmail.com>

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). doi:[10.1038/s41467-022-29838-9](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 = samples_modis_ndvi,
    roi = mt_sf
  )
}
```

```

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

```

sits_get_class	<i>Get values from classified maps</i>
----------------	--

Description

Given a set of lat/long locations and a classified cube, retrieve the class of each point. This function is useful to obtain values from classified cubes for accuracy estimates.

Usage

```

sits_get_class(cube, samples)

## Default S3 method:
sits_get_class(cube, samples)

## S3 method for class 'csv'
sits_get_class(cube, samples)

## S3 method for class 'shp'
sits_get_class(cube, samples)

## S3 method for class 'sf'
sits_get_class(cube, samples)

## S3 method for class 'sits'
sits_get_class(cube, samples)

## S3 method for class 'data.frame'
sits_get_class(cube, samples)

```

Arguments

cube	Classified data cube.
samples	Location of the samples to be retrieved. Either a tibble of class "sits", an "sf" object, the name of a shapefile or csv file, or a data.frame with columns "longitude" and "latitude"

Value

A tibble of with columns <longitude, latitude, start_date, end_date, label>.

Note

There are four ways of specifying data to be retrieved using the `samples` parameter: (a) CSV file: a CSV file with columns `longitude`, `latitude`; (b) SHP file: a shapefile in `POINT` geometry; (c) `sits` object: A `sits` tibble; (d) `sf` object: An `link[sf]{sf}` object with `POINT` or geometry; (e) `data.frame`: A `data.frame` with `longitude` and `latitude`.

Author(s)

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

Examples

```
if (sits_run_examples()) {
  # create a random forest model
  rfor_model <- sits_train(samples_modis_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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # classify a data cube
  probs_cube <- sits_classify(
    data = cube, ml_model = rfor_model, output_dir = tempdir()
  )
  # plot the probability cube
  plot(probs_cube)
  # smooth the probability cube using Bayesian statistics
  bayes_cube <- sits_smooth(probs_cube, output_dir = tempdir())
  # plot the smoothed cube
  plot(bayes_cube)
  # label the probability cube
  label_cube <- sits_label_classification(
    bayes_cube,
    output_dir = tempdir()
  )
  # obtain the a set of points for sampling
  ground_truth <- system.file("extdata/samples/samples_sinop_crop.csv",
    package = "sits"
  )
  # get the classification values for a selected set of locations
  labels_samples <- sits_get_class(label_cube, ground_truth)
}
```

Description

Retrieve a set of time series from a data cube and put the result in a `sits` tibble, which contains both the satellite image time series and their metadata.

There are five options for the specifying the input samples parameter:

- A CSV file: see [sits_get_data.csv](#).
- A shapefile: see [sits_get_data.shp](#).
- An `sf` object: see [sits_get_data.sf](#).
- A `sits` tibble: see [sits_get_data.sits](#).
- A `data.frame`: see [sits_get_data.data.frame](#).

Usage

```
sits_get_data(cube, samples, ...)
```

```
## Default S3 method:
```

```
sits_get_data(cube, samples, ...)
```

Arguments

<code>cube</code>	Data cube from where data is to be retrieved. (tibble of class "raster_cube").
<code>samples</code>	Location of the samples to be retrieved. Either a tibble of class "sits", an "sf" object, the name of a shapefile or csv file, or a <code>data.frame</code> with columns "longitude" and "latitude".
<code>...</code>	Specific parameters for each input.

Value

A tibble of class "sits" with set of time series <longitude, latitude, start_date, end_date, label, time_series>.

Note

The main `sits` classification workflow has the following steps:

1. [sits_cube](#): selects a ARD image collection from a cloud provider.
2. [sits_cube_copy](#): copies an ARD image collection from a cloud provider to a local directory for faster processing.
3. [sits_regularize](#): create a regular data cube from an ARD image collection.
4. [sits_apply](#): create new indices by combining bands of a regular data cube (optional).
5. [sits_get_data](#): extract time series from a regular data cube based on user-provided labelled samples.
6. [sits_train](#): train a machine learning model based on image time series.
7. [sits_classify](#): classify a data cube using a machine learning model and obtain a probability cube.

8. `sits_smooth`: post-process a probability cube using a spatial smoother to remove outliers and increase spatial consistency.
9. `sits_label_classification`: produce a classified map by selecting the label with the highest probability from a smoothed cube.

To be able to build a machine learning model to classify a data cube, one needs to use a set of labelled time series. These time series are created by taking a set of known samples, expressed as labelled points or polygons. This `sits_get_data` function uses these samples to extract time series from a data cube. It needs a cube parameter which points to a regularized data cube, and a samples parameter that describes the locations of the training set.

Author(s)

Felipe Carlos, <efelipeCarlos@gmail.com>
 Felipe Carvalho, <felipe.carvalho@inpe.br>
 Gilberto Camara, <gilberto.camara@inpe.br>
 Rolf Simoes, <rolfsimoes@gmail.com>

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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  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)
  bdc_cube <- sits_cube(
    source = "BDC",
    collection = "CBERS-WFI-16D",
    bands = c("NDVI", "EVI"),
    tiles = c("007004", "007005"),
    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_get_data.csv *Get time series using CSV files*

Description

Retrieve a set of time series from a data cube and put the result in a "sits tibble", which contains both the satellite image time series and their metadata. The `samples` parameter must point to a file with extension ".csv", with mandatory columns longitude, latitude, label, start_date and end_date.

Usage

```

## S3 method for class 'csv'
sits_get_data(
  cube,
  samples,
  ...,
  bands = NULL,
  crs = "EPSG:4326",
  impute_fn = impute_linear(),
  multicores = 2L,
  progress = FALSE
)

```

Arguments

<code>cube</code>	Data cube from where data is to be retrieved. (tibble of class "raster_cube").
<code>samples</code>	Location of a csv file.
<code>...</code>	Specific parameters for each kind of input.
<code>bands</code>	Bands to be retrieved - optional.
<code>crs</code>	A character with the samples crs. Default is "EPSG:4326".
<code>impute_fn</code>	Imputation function to remove NA.
<code>multicores</code>	Number of threads to process the time series (integer, with min = 1 and max = 2048).
<code>progress</code>	Logical: show progress bar?

Value

A tibble of class "sits" with set of time series and metadata with <longitude, latitude, start_date, end_date, label, time_series>.

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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # 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_get_data.data.frame

Get time series using sits objects

Description

Retrieve a set of time series from a data cube and and put the result in a sits tibble. The samples parameter should be a data.frame which contains mandatory columns longitude and latitude, and optional columns start_date, end_date and label for each sample.

Usage

```
## S3 method for class 'data.frame'
sits_get_data(
  cube,
  samples,
  ...,
  start_date = NULL,
  end_date = NULL,
  bands = NULL,
  impute_fn = impute_linear(),
  label = "NoClass",
  crs = "EPSG:4326",
  multicores = 2,
  progress = FALSE
)
```

Arguments

cube	Data cube from where data is to be retrieved. (tibble of class "raster_cube").
samples	A data.frame with mandatory columns longitude, and latitude, and optional columns start_date, end_date, label.
...	Specific parameters for specific cases.
start_date	Start of the interval for the time series - optional (Date in "YYYY-MM-DD" format).
end_date	End of the interval for the time series - optional (Date in "YYYY-MM-DD" format).
bands	Bands to be retrieved - optional.
impute_fn	Imputation function to remove NA.
label	Label to be assigned to all time series if column label is not provided in the data.frame.
crs	A character with the samples crs. Default is "EPSG:4326".
multicores	Number of threads to process the time series (integer, with min = 1 and max = 2048).
progress	Logical: show progress bar?

Value

A tibble of class "sits" with set of time series <longitude, latitude, start_date, end_date, label>.

Examples

```
if (sits_run_examples()) {
  # create a cube from local files
  data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
  raster_cube <- sits_cube(
    source = "BDC",
    collection = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # read a lat/long from a local cube
  samples <- data.frame(longitude = -55.66738, latitude = -11.76990)
  point_ndvi <- sits_get_data(raster_cube, samples)
}
```

sits_get_data.sf	<i>Get time series using sf objects</i>
------------------	---

Description

Retrieve a set of time series from a data cube and and put the result in a "sits tibble", which contains both the satellite image time series and their metadata. The samples parameter must be a sf object in POINT or POLYGON geometry. If start_date and end_date are not informed, the function uses these data from the cube.

Usage

```
## S3 method for class 'sf'
sits_get_data(
  cube,
  samples,
  ...,
  start_date = NULL,
  end_date = NULL,
  bands = NULL,
  impute_fn = impute_linear(),
  label = "NoClass",
  label_attr = NULL,
  n_sam_pol = 30L,
  pol_avg = FALSE,
  sampling_type = "random",
  multicores = 2L,
  progress = FALSE
)
```

Arguments

<code>cube</code>	Data cube from where data is to be retrieved. (tibble of class "raster_cube").
<code>samples</code>	The name of a shapefile.
<code>...</code>	Specific parameters for specific cases.
<code>start_date</code>	Start of the interval for the time series - optional (Date in "YYYY-MM-DD" format).
<code>end_date</code>	End of the interval for the time series - optional (Date in "YYYY-MM-DD" format).
<code>bands</code>	Bands to be retrieved - optional (character vector).
<code>impute_fn</code>	Imputation function to remove NA.
<code>label</code>	Label to be assigned to all time series - optional
<code>label_attr</code>	Attribute in the sf object to be used as a polygon label.
<code>n_sam_pol</code>	Number of samples per polygon to be read for POLYGON or MULTIPOLYGON objects.
<code>pol_avg</code>	Logical: summarize samples for each polygon?
<code>sampling_type</code>	Spatial sampling type: random, hexagonal, regular, or Fibonacci.
<code>multicores</code>	Number of threads to process the time series (integer, with min = 1 and max = 2048).
<code>progress</code>	Logical: show progress bar?

Value

A tibble of class "sits" with set of time series <longitude, latitude, start_date, end_date, label>.

Note

For sf objects, the following parameters are relevant:

- `label`: label to be assigned to the samples. Should only be used if all geometries have a single label.
- `label_attr`: defines which attribute should be used as a label, required for POINT and POLYGON geometries if `label` has not been set.
- `n_sam_pol`: indicates how many points are extracted from each polygon, required for POLYGON geometry (default = 15).
- `sampling_type`: defines how sampling is done, required for POLYGON geometry (default = "random").
- `pol_avg`: indicates if average of values for POLYGON geometry should be computed (default = "FALSE").

Examples

```
if (sits_run_examples()) {
  # reading a shapefile from BDC (Brazil Data Cube)
  bdc_cube <- sits_cube(
    source = "BDC",
    collection = "CBERS-WFI-16D",
    bands = c("NDVI", "EVI"),
    tiles = c("007004", "007005"),
    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"
  )
  # read a shapefile into an sf object
  sf_object <- sf::st_read(shp_file)
  # get samples from the BDC using an sf object
  time_series_bdc <- sits_get_data(
    cube = bdc_cube,
    samples = sf_object
  )
}
```

sits_get_data.shp

Get time series using shapefiles

Description

Retrieve a set of time series from a data cube and put the result in a `sits` tibble, which contains both the satellite image time series and their metadata. The `samples` parameter must point to a file with extension ".shp" which should be a valid shapefile in POINT or POLYGON geometry. If `start_date` and `end_date` are not informed, the function uses these data from the cube.

Usage

```
## S3 method for class 'shp'
sits_get_data(
  cube,
  samples,
  ...,
  start_date = NULL,
  end_date = NULL,
  bands = NULL,
  impute_fn = impute_linear(),
  label = "NoClass",
  label_attr = NULL,
  n_sam_pol = 30L,
  pol_avg = FALSE,
  sampling_type = "random",
  multicores = 2L,
  progress = FALSE
)
```

Arguments

cube	Data cube from where data is to be retrieved. (tibble of class "raster_cube").
samples	The name of a shapefile.
...	Specific parameters for specific cases.
start_date	Start of the interval for the time series - optional (Date in "YYYY-MM-DD" format).
end_date	End of the interval for the time series - optional (Date in "YYYY-MM-DD" format).
bands	Bands to be retrieved - optional
impute_fn	Imputation function to remove NA.
label	Label to be assigned to all time series - optional
label_attr	Attribute in the shapefile to be used as a polygon label.
n_sam_pol	Number of samples per polygon to be read for POLYGON or MULTIPOLYGON shapefiles.
pol_avg	Logical: summarize samples for each polygon?
sampling_type	Spatial sampling type: random, hexagonal, regular, or Fibonacci.
multicores	Number of threads to process the time series (integer, with min = 1 and max = 2048).
progress	Logical: show progress bar?

Value

A tibble of class "sits" with set of time series and metadata <longitude, latitude, start_date, end_date, label, time_series>.

Note

For shapefiles, the following parameters are relevant:

- `label`: label to be assigned to the samples. Should only be used if all geometries have a single label.
- `label_attr`: defines which attribute should be used as a label, required for POINT and POLYGON geometries if `label` has not been set.
- `n_sam_pol`: indicates how many points are extracted from each polygon, required for POLYGON geometry (default = 15).
- `sampling_type`: defines how sampling is done, required for POLYGON geometry (default = "random").
- `pol_avg`: indicates if average of values for POLYGON geometry should be computed (default = "FALSE").

Examples

```
if (sits_run_examples()) {
  # reading a shapefile from BDC (Brazil Data Cube)
  bdc_cube <- sits_cube(
    source = "BDC",
    collection = "CBERS-WFI-16D",
    bands = c("NDVI", "EVI"),
    tiles = c("007004", "007005"),
    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_get_data.sits *Get time series using sits objects*

Description

Retrieve a set of time series from a data cube and put the result in a sits tibble. The `samples` parameter should be a valid sits tibble which contains columns longitude, latitude, `start_date`, `end_date` and `label` for each sample.

Usage

```
## S3 method for class 'sits'
sits_get_data(
  cube,
  samples,
  ...,
  bands = NULL,
  crs = "EPSG:4326",
  impute_fn = impute_linear(),
  multicores = 2L,
  progress = FALSE
)
```

Arguments

cube	Data cube from where data is to be retrieved. (tibble of class "raster_cube").
samples	Location of the samples to be retrieved. Either a tibble of class "sits", an "sf" object, the name of a shapefile or csv file, or a data.frame with columns "longitude" and "latitude".
...	Specific parameters for specific cases.
bands	Bands to be retrieved - optional.
crs	A character with the samples crs. Default is "EPSG:4326".
impute_fn	Imputation function to remove NA.
multicores	Number of threads to process the time series (integer, with min = 1 and max = 2048).
progress	Logical: show progress bar?

Value

A tibble of class "sits" with set of time series <longitude, latitude, start_date, end_date, label>.

sits_get_probs	<i>Get values from probability maps</i>
----------------	---

Description

Given a set of lat/long locations and a probability cube, retrieve the prob values of each point. This function is useful to estimate probability distributions and to assess the differences between classifiers.

Usage

```
sits_get_probs(cube, samples, window_size = NULL)

## S3 method for class 'csv'
sits_get_probs(cube, samples, window_size = NULL)

## S3 method for class 'shp'
sits_get_probs(cube, samples, window_size = NULL)

## S3 method for class 'sf'
sits_get_probs(cube, samples, window_size = NULL)

## S3 method for class 'sits'
sits_get_probs(cube, samples, window_size = NULL)

## S3 method for class 'data.frame'
sits_get_probs(cube, samples, window_size = NULL)

## Default S3 method:
sits_get_probs(cube, samples, window_size = NULL)
```

Arguments

cube	Probability data cube.
samples	Location of the samples to be retrieved. Either a tibble of class "sits", an "sf" object with POINT geometry, the location of a POINT shapefile, the location of csv file with columns "longitude" and "latitude", or a data.frame with columns "longitude" and "latitude"
window_size	Size of window around pixel (optional)

Value

A tibble of with columns <longitude, latitude, values> in case no windows are requested and <longitude, latitude, neighbors> in case windows are requested

Note

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

- CSV: a CSV file with columns longitude, latitude.
- SHP: a shapefile in POINT geometry.
- sf object: An link[sf]{sf} object with POINT geometry.
- sits object: A valid tibble with sits timeseries.
- data.frame: A data.frame with longitude and latitude.

Author(s)

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

Examples

```

if (sits_run_examples()) {
  # create a random forest model
  rfor_model <- sits_train(samples_modis_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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # classify a data cube
  probs_cube <- sits_classify(
    data = cube, ml_model = rfor_model, output_dir = tempdir()
  )
  # obtain the a set of points for sampling
  ground_truth <- system.file("extdata/samples/samples_sinop_crop.csv",
    package = "sits"
  )
  # get the classification values for a selected set of locations
  probs_samples <- sits_get_probs(probs_cube, ground_truth)
}

```

sits_impute

*Replace NA values in time series with imputation function***Description**

Remove NA

Usage

```
sits_impute(samples, impute_fn = impute_linear())
```

Arguments

<code>samples</code>	A time series tibble
<code>impute_fn</code>	Imputation function

Value

A set of filtered time series using the imputation function.

Author(s)

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

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.

Usage

```
sits_kfold_validate(  
  samples,  
  folds = 5L,  
  ml_method = sits_rfor(),  
  filter_fn = NULL,  
  impute_fn = impute_linear(),  
  multicores = 2L,  
  gpu_memory = 4L,  
  batch_size = 2L^gpu_memory,  
  progress = TRUE  
)
```

Arguments

samples	Time series.
folds	Number of partitions to create.
ml_method	Machine learning method.
filter_fn	Smoothing filter to be applied - optional (closure containing object of class "function").
impute_fn	Imputation function to remove NA.
multicores	Number of cores to process in parallel.
gpu_memory	Memory available in GPU in GB (default = 4)
batch_size	Batch size for GPU classification.
progress	Logical: Show progress bar?

Value

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

Note

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.

Author(s)

Rolf Simoes, <rolfsimoes@gmail.com>

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_rfor <- sits_kfold_validate(
    samples_modis_ndvi,
    folds = 5,
    ml_method = sits_rfor()
  )
  # use a name
  acc_rfor$name <- "Rfor"
  # put the result in a list
  results[[length(results) + 1]] <- acc_rfor
  # save to xlsx file
  sits_to_xlsx(
    results,
    file = tempfile("accuracy_mato_grosso_dl_", fileext = ".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 'derived_cube'
sits_labels(data)

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

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

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

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

## Default S3 method:
sits_labels(data)
```

Arguments

data	Time series (tibble of class "sits"), patterns (tibble of class "patterns"), data cube (tibble of class "raster_cube"), or model (closure of class "sits_model").
------	---

Value

The labels of the input data (character vector).

Author(s)

Rolf Simoes, <rolfsimoes@gmail.com>

Examples

```
if (sits_run_examples()) {
  # get the labels for a time series set
  labels_ts <- sits_labels(samples_modis_ndvi)
  # get labels for a set of patterns
  labels_pat <- sits_labels(sits_patterns(samples_modis_ndvi))
  # create a random forest model
  rfor_model <- sits_train(samples_modis_ndvi, sits_rfor())
  # get labels for the model
  labels_mod <- sits_labels(rfor_model)
  # create a data cube from local files
  data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
}
```

```

cube <- sits_cube(
  source = "BDC",
  collection = "MOD13Q1-6.1",
  data_dir = data_dir
)
# classify a data cube
probs_cube <- sits_classify(
  data = cube, ml_model = rfor_model, output_dir = tempdir()
)
# get the labels for a probs cube
labels_probs <- sits_labels(probs_cube)
}

```

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
```

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 or data cube with modified labels.

Author(s)

Rolf Simoes, <rolfsimoes@gmail.com>

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)

```

```
sits_labels<-.class_cube
```

Change the labels of a set of time series

Description

Change the labels of a set of time series

Usage

```
## S3 replacement method for class 'class_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 .

```
sits_labels<-.default Change the labels of a set of time series
```

Description

Change the labels of a set of time series

Usage

```
## Default S3 replacement method:  
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 .

```
sits_labels<-.probs_cube
```

Change the labels of a set of time series

Description

Change the labels of a set of time series

Usage

```
## 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 .

```
sits_labels<-.sits
```

Change the labels of a set of time series

Description

Change the labels of a set of time series

Usage

```
## S3 replacement method for class 'sits'  
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 .

sits_labels_summary	<i>Inform label distribution of a set of time series</i>
---------------------	--

Description

Describes labels in a sits tibble

Usage

```
sits_labels_summary(data)

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

Arguments

data	Data.frame - Valid sits tibble
------	--------------------------------

Value

A tibble with the frequency of each label.

Author(s)

Rolf Simoes, <rolfsimoes@gmail.com>

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	<i>Build a labelled image from a probability cube</i>
---------------------------	---

Description

Takes a set of classified raster layers with probabilities, and labels them based on the maximum probability for each pixel. This function is the final step of main the land classification workflow.

Usage

```
sits_label_classification(cube, ...)

## S3 method for class 'probs_cube'
sits_label_classification(
  cube,
  ...,
  memsize = 4L,
  multicores = 2L,
  output_dir,
  version = "v1",
  progress = TRUE
)

## S3 method for class 'probs_vector_cube'
sits_label_classification(
  cube,
  ...,
  output_dir,
  version = "v1",
  progress = TRUE
)

## S3 method for class 'raster_cube'
sits_label_classification(cube, ...)

## S3 method for class 'derived_cube'
sits_label_classification(cube, ...)

## Default S3 method:
sits_label_classification(cube, ...)
```

Arguments

<code>cube</code>	Classified image data cube.
<code>...</code>	Other parameters for specific functions.
<code>memsize</code>	maximum overall memory (in GB) to label the classification.
<code>multicores</code>	Number of workers to label the classification in parallel.
<code>output_dir</code>	Output directory for classified files.
<code>version</code>	Version of resulting image (in the case of multiple runs).
<code>progress</code>	Show progress bar?

Value

A data cube with an image with the classified map.

Note

The main sits classification workflow has the following steps:

1. `sits_cube`: selects a ARD image collection from a cloud provider.
2. `sits_cube_copy`: copies an ARD image collection from a cloud provider to a local directory for faster processing.
3. `sits_regularize`: create a regular data cube from an ARD image collection.
4. `sits_apply`: create new indices by combining bands of a regular data cube (optional).
5. `sits_get_data`: extract time series from a regular data cube based on user-provided labelled samples.
6. `sits_train`: train a machine learning model based on image time series.
7. `sits_classify`: classify a data cube using a machine learning model and obtain a probability cube.
8. `sits_smooth`: post-process a probability cube using a spatial smoother to remove outliers and increase spatial consistency.
9. `sits_label_classification`: produce a classified map by selecting the label with the highest probability from a smoothed cube.

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

Author(s)

Rolf Simoes, <rolfsimoes@gmail.com>

Felipe Souza, <felipe.souza@inpe.br>

Examples

```
if (sits_run_examples()) {
  # create a random forest model
  rfor_model <- sits_train(samples_modis_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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # classify a data cube
  probs_cube <- sits_classify(
    data = cube, ml_model = rfor_model, output_dir = tempdir()
  )
  # plot the probability cube
  plot(probs_cube)
  # smooth the probability cube using Bayesian statistics
  bayes_cube <- sits_smooth(probs_cube, output_dir = tempdir())
  # plot the smoothed cube
  plot(bayes_cube)
```

```

# label the probability cube
label_cube <- sits_label_classification(
  bayes_cube,
  output_dir = tempdir()
)
# plot the labelled cube
plot(label_cube)
}

```

sits_lightgbm

Train light gradient boosting model

Description

Use LightGBM algorithm to classify samples. This function is a front-end to the `lightgbm` package. LightGBM (short for Light Gradient Boosting Machine) is a gradient boosting framework developed by Microsoft that's designed for fast, scalable, and efficient training of decision tree-based models. It is widely used in machine learning for classification, regression, ranking, and other tasks, especially with large-scale data.

Usage

```

sits_lightgbm(
  samples = NULL,
  boosting_type = "gbdt",
  objective = "multiclass",
  min_samples_leaf = 20,
  max_depth = 6,
  learning_rate = 0.1,
  num_iterations = 100,
  n_iter_no_change = 10,
  validation_split = 0.2,
  ...
)

```

Arguments

<code>samples</code>	Time series with the training samples.
<code>boosting_type</code>	Type of boosting algorithm (default = "gbdt")
<code>objective</code>	Aim of the classifier (default = "multiclass").
<code>min_samples_leaf</code>	Minimal number of data in one leaf. Can be used to deal with over-fitting.
<code>max_depth</code>	Limit the max depth for tree model.
<code>learning_rate</code>	Shrinkage rate for leaf-based algorithm.
<code>num_iterations</code>	Number of iterations to train the model.

`n_iter_no_change` Number of iterations without improvements until training stops.

`validation_split` 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.

`...` Other parameters to be passed to 'lightgbm::lightgbm' function.

Value

Model fitted to input data (to be passed to [sits_classify](#)).

Author(s)

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
  lgb_model <- sits_train(samples_modis_ndvi,
    ml_method = sits_lightgbm
  )
  # classify the point
  point_ndvi <- sits_select(point_mt_6bands, bands = "NDVI")
  # classify the point
  point_class <- sits_classify(
    data = point_ndvi, ml_model = lgb_model
  )
  plot(point_class)
}
```

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 is a lightweight version of the temporal attention encoder proposed by Garnot et al. For the TAE, please see [sits_tae](#).

TAE is a simplified version of the well-known self-attention architecture which is used in large language models. Its modified self-attention scheme that uses the input embeddings as values. TAE defines a single master query for each sequence, computed from the temporal average of the queries. This master query is compared to the sequence of keys to produce a single attention mask used to weight the temporal mean of values into a single feature vector.

The lightweight version of TAE further simplifies the TAE model. It defines master query of each head as a model parameter instead of the results of a linear layer, as is done in TAE. The authors argue that such simplification reduces the number of parameters, while the lack of flexibility is compensated by the larger number of available heads.

Usage

```
sits_lighttiae(
  samples = NULL,
  samples_validation = NULL,
  epochs = 150L,
  batch_size = 128L,
  validation_split = 0.2,
  optimizer = torch::optim_adamw,
  opt_hparams = list(lr = 5e-04, eps = 1e-08, weight_decay = 7e-04),
  lr_decay_epochs = 50L,
  lr_decay_rate = 1,
  patience = 20L,
  min_delta = 0.01,
  seed = NULL,
  verbose = FALSE
)
```

Arguments

<code>samples</code>	Time series with the training samples (tibble of class "sits").
<code>samples_validation</code>	Time series with the validation samples (tibble of class "sits"). If <code>samples_validation</code> parameter is provided, <code>validation_split</code> is ignored.
<code>epochs</code>	Number of iterations to train the model (integer, min = 1, max = 20000).
<code>batch_size</code>	Number of samples per gradient update (integer, min = 16L, max = 2048L)
<code>validation_split</code>	Fraction of training data to be used as validation data.
<code>optimizer</code>	Optimizer function to be used.
<code>opt_hparams</code>	Hyperparameters for optimizer: <code>lr</code> : Learning rate of the optimizer <code>eps</code> : Term added to the denominator to improve numerical stability. <code>weight_decay</code> : L2 regularization rate.
<code>lr_decay_epochs</code>	Number of epochs to reduce learning rate.
<code>lr_decay_rate</code>	Decay factor for reducing learning rate.
<code>patience</code>	Number of epochs without improvements until training stops.
<code>min_delta</code>	Minimum improvement in loss function to reset the patience counter.
<code>seed</code>	Seed for random values.
<code>verbose</code>	Verbosity mode (TRUE/FALSE). Default is FALSE.

Value

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

Note

sits provides a set of default values for all classification models. These settings have been chosen based on testing by the authors. Nevertheless, users can control all parameters for each model. Novice users can rely on the default values, while experienced ones can fine-tune deep learning models using [sits_tuning](#).

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>.

Author(s)

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

Rolf Simoes, <rolfsimoes@gmail.com>

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

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](https://doi.org/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](https://doi.org/10.5281/zenodo.4835356)

Examples

```
if (sits_run_examples()) {
  # create a lightTAE model
  torch_model <- sits_train(samples_modis_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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # classify a data cube
  probs_cube <- sits_classify(
    data = cube, ml_model = torch_model, output_dir = tempdir()
```

```

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

```

`sits_list_collections` *List the cloud collections supported by sits*

Description

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_list_collections(source = NULL)
```

Arguments

source	Data source to be shown in detail.
--------	------------------------------------

Value

Prints collections available in each cloud service supported by sits.

Examples

```

if (sits_run_examples()) {
  # show the names of the colors supported by SITS
  sits_list_collections()
}

```


sits_lstm_fcn

*Train a Long Short Term Memory Fully Convolutional Network***Description**

Uses a branched neural network consisting of a lstm (long short term memory) branch and a three-layer fully convolutional branch (FCN) followed by concatenation to classify time series data.

This function is based on the paper by Fazle Karim, Somshubra Majumdar, and Houshang Darabi. If you use this method, please cite the original LSTM with FCN paper.

The torch version is based on the code made available by the titu1994. The original python code is available at the website <https://github.com/titu1994/LSTM-FCN>. This code is licensed as GPL-3.

Usage

```
sits_lstm_fcn(
    samples = NULL,
    samples_validation = NULL,
    cnn_layers = c(128, 256, 128),
    cnn_kernels = c(8, 5, 3),
    lstm_width = 8,
    lstm_dropout = 0.8,
    epochs = 50,
    batch_size = 64,
    validation_split = 0.2,
    optimizer = torch::optim_adamw,
    opt_hparams = list(lr = 5e-04, eps = 1e-08, weight_decay = 1e-06),
    lr_decay_epochs = 1,
    lr_decay_rate = 0.95,
    patience = 20,
    min_delta = 0.01,
    seed = NULL,
    verbose = FALSE
)
```

Arguments

<code>samples</code>	Time series with the training samples.
<code>samples_validation</code>	Time series with the validation samples. if the <code>samples_validation</code> parameter is provided, the <code>validation_split</code> parameter is ignored.
<code>cnn_layers</code>	Number of 1D convolutional filters per layer
<code>cnn_kernels</code>	Size of the 1D convolutional kernels.
<code>lstm_width</code>	Number of neuros in the lstm's hidden layer.
<code>lstm_dropout</code>	Dropout rate of the lstm 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.
seed	Seed for random values.
verbose	Verbosity mode (TRUE/FALSE). Default is FALSE.

Value

A fitted model to be used for classification.

Author(s)

Alexandre Assuncao, <alexcarssuncao@gmail.com>

References

F. Karim, S. Majumdar, H. Darabi and S. Chen, "LSTM Fully Convolutional Networks for Time Series Classification," in IEEE Access, vol. 6, pp. 1662-1669, 2018, doi:10.1109/ACCESS.2017.2779939.

sits_merge	<i>Merge two data sets (time series or cubes)</i>
------------	---

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 for merging different bands of the same location. For example, one may want to put the raw and smoothed bands for the same set of locations in the same tibble.

In the case of data cubes, the function merges the images based on the following conditions:

1. If the two cubes have different bands but compatible timelines, the bands are combined, and the timeline is adjusted to overlap. To create the overlap, we align the timelines like a "zipper": for each interval defined by a pair of consecutive dates in the first timeline, we include matching dates from the second timeline. If the second timeline has multiple dates in the same interval, only the minimum date is kept. This ensures the final timeline avoids duplicates and is consistent. This is useful when merging data from different sensors (e.g., Sentinel-1 with Sentinel-2).

2. If the bands are the same, the cube will have the combined timeline of both cubes. This is useful for merging data from the same sensors from different satellites (e.g., Sentinel-2A with Sentinel-2B).
3. otherwise, the function will produce an error.

Usage

```
sits_merge(data1, data2, ...)

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

## S3 method for class 'raster_cube'
sits_merge(data1, data2, ...)

## Default S3 method:
sits_merge(data1, data2, ...)
```

Arguments

data1	Time series (tibble of class "sits") or data cube (tibble of class "raster_cube") .
data2	Time series (tibble of class "sits") or data cube (tibble of class "raster_cube") .
...	Additional parameters
suffix	If data1 and data2 are tibble with duplicate bands, this suffix will be added (character vector).

Value

merged data sets (tibble of class "sits" or tibble of class "raster_cube")

Author(s)

Felipe Carvalho, <felipe.carvalho@inpe.br>
 Felipe Carlos, <efelipeccarlos@gmail.com>

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_mgrs_to_roi	<i>Convert MGRS tile information to ROI in WGS84</i>
------------------	--

Description

Takes a list of MGRS tiles and produces a ROI covering them

Usage

```
sits_mgrs_to_roi(tiles)
```

Arguments

tiles	Character vector with names of MGRS tiles
-------	---

Value

roi Valid ROI to use in other SITS functions

Author(s)

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

Rolf Simoes, <rolf.simoes@gmail.com>

sits_mixture_model	<i>Multiple endmember spectral mixture analysis</i>
--------------------	---

Description

Create a multiple endmember spectral mixture analyses fractions images. We use the non-negative least squares (NNLS) solver to calculate the fractions of each endmember. The NNLS was implemented by Jakob Schwalb-Willmann in RStoolbox package (licensed as GPL>=3).

Usage

```
sits_mixture_model(data, endmembers, ...)
```

```
## S3 method for class 'sits'
```

```
sits_mixture_model(
  data,
  endmembers,
  ...,
  rmse_band = TRUE,
  multicores = 2L,
  progress = TRUE
```

```

)

## S3 method for class 'raster_cube'
sits_mixture_model(
  data,
  endmembers,
  ...,
  rmse_band = TRUE,
  memsize = 4L,
  multicores = 2L,
  output_dir,
  progress = TRUE
)

## S3 method for class 'derived_cube'
sits_mixture_model(data, endmembers, ...)

## S3 method for class 'tbl_df'
sits_mixture_model(data, endmembers, ...)

## Default S3 method:
sits_mixture_model(data, endmembers, ...)

```

Arguments

<code>data</code>	A sits data cube or a sits tibble.
<code>endmembers</code>	Reference spectral endmembers. (see details below).
<code>...</code>	Parameters for specific functions.
<code>rmse_band</code>	A boolean indicating whether the error associated with the linear model should be generated. If true, a new band with errors for each pixel is generated using the root mean square measure (RMSE). Default is TRUE.
<code>multicores</code>	Number of cores to be used for generate the mixture model.
<code>progress</code>	Show progress bar? Default is TRUE.
<code>memsize</code>	Memory available for the mixture model (in GB).
<code>output_dir</code>	Directory for output images.

Value

In case of a cube, a sits cube with the fractions of each endmember will be returned. The sum of all fractions is restricted to 1 (scaled from 0 to 10000), corresponding to the abundance of the endmembers in the pixels. In case of a sits tibble, the time series will be returned with the values corresponding to each fraction.

Note

Many pixels in images of medium-resolution satellites such as Landsat or Sentinel-2 contain a mixture of spectral responses of different land cover types. In many applications, it is desirable to

obtain the proportion of a given class inside a mixed pixel. For this purpose, the literature proposes mixture models; these models represent pixel values as a combination of multiple pure land cover types. Assuming that the spectral response of pure land cover classes (called endmembers) is known, spectral mixture analysis derives new bands containing the proportion of each endmember inside a pixel.

The endmembers parameter should be a tibble, csv or a shapefile. endmembers parameter 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. The band values must follow the product scale. For example, in the case of sentinel-2 images the bands should be in the range 0 to 1. See the example in this documentation for more details.

Author(s)

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

Felipe Carlos, <efelipecarlos@gmail.com>

Rolf Simoes, <rolfsimoes@gmail.com>

References

RStoolbox R package.

Examples

```
if (sits_run_examples()) {
  # Create a sentinel-2 cube
  s2_cube <- sits_cube(
    source = "AWS",
    collection = "SENTINEL-2-L2A",
    tiles = "20LKP",
    bands = c("B02", "B03", "B04", "B8A", "B11", "B12", "CLOUD"),
    start_date = "2019-06-13",
    end_date = "2019-06-30"
  )
  # create a directory to store the regularized file
  reg_dir <- paste0(tempdir(), "/mix_model")
  dir.create(reg_dir)
  # Cube regularization for 16 days and 160 meters
  reg_cube <- sits_regularize(
    cube = s2_cube,
    period = "P16D",
    res = 160,
    roi = c(
      lon_min = -65.54870165,
      lat_min = -10.63479162,
      lon_max = -65.07629670,
      lat_max = -10.36046639
    ),
    multicores = 2,
    output_dir = reg_dir
  )
}
```

```

# Create the endmembers tibble
em <- tibble::tribble(
  ~class, ~B02, ~B03, ~B04, ~B8A, ~B11, ~B12,
  "forest", 0.02, 0.0352, 0.0189, 0.28, 0.134, 0.0546,
  "land", 0.04, 0.065, 0.07, 0.36, 0.35, 0.18,
  "water", 0.07, 0.11, 0.14, 0.085, 0.004, 0.0026
)

# Generate the mixture model
mm <- sits_mixture_model(
  data = reg_cube,
  endmembers = em,
  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(512L, 512L, 512L),
  dropout_rates = c(0.2, 0.3, 0.4),
  optimizer = torch::optim_adamw,
  opt_hparams = list(lr = 0.001, eps = 1e-08, weight_decay = 1e-06),
  epochs = 100L,
  batch_size = 64L,
  validation_split = 0.2,
  patience = 20L,
  min_delta = 0.01,
  seed = NULL,
  verbose = FALSE
)

```

Arguments

<code>samples</code>	Time series with the training samples.
<code>samples_validation</code>	Time series with the validation samples. if the <code>samples_validation</code> parameter is provided, the <code>validation_split</code> parameter is ignored.
<code>layers</code>	Vector with number of hidden nodes in each layer.
<code>dropout_rates</code>	Vector with the dropout rates (0,1) for each layer.
<code>optimizer</code>	Optimizer function to be used.
<code>opt_hparams</code>	Hyperparameters for optimizer: <code>lr</code> : Learning rate of the optimizer <code>eps</code> : Term added to the denominator to improve numerical stability.. <code>weight_decay</code> : L2 regularization
<code>epochs</code>	Number of iterations to train the model.
<code>batch_size</code>	Number of samples per gradient update.
<code>validation_split</code>	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.
<code>patience</code>	Number of epochs without improvements until training stops.
<code>min_delta</code>	Minimum improvement in loss function to reset the patience counter.
<code>seed</code>	Seed for random values.
<code>verbose</code>	Verbosity mode (TRUE/FALSE). Default is FALSE.

Value

A torch mlp model to be used for classification.

Note

`sits` provides a set of default values for all classification models. These settings have been chosen based on testing by the authors. Nevertheless, users can control all parameters for each model. Novice users can rely on the default values, while experienced ones can fine-tune deep learning models using [sits_tuning](#).

The default 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.

Author(s)

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

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).

Examples

```
if (sits_run_examples()) {
  # create an MLP model
  torch_model <- sits_train(
    samples_modis_ndvi,
    sits_mlp(epochs = 20, verbose = TRUE)
  )
  # 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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # classify a data cube
  probs_cube <- sits_classify(
    data = cube, ml_model = torch_model, output_dir = tempdir()
  )
  # plot the probability cube
  plot(probs_cube)
  # smooth the probability cube using Bayesian statistics
  bayes_cube <- sits_smooth(probs_cube, output_dir = tempdir())
  # plot the smoothed cube
  plot(bayes_cube)
  # label the probability cube
  label_cube <- sits_label_classification(
    bayes_cube,
    output_dir = tempdir()
  )
  # plot the labelled cube
  plot(label_cube)
}
```

sits_model_export

Export classification models

Description

Given a trained machine learning or deep learning model, exports the model as an object for further exploration outside the sits package.

Usage

```
sits_model_export(ml_model)

## S3 method for class 'sits_model'
sits_model_export(ml_model)
```

Arguments

`ml_model` A trained machine learning model

Value

An R object containing the model in the original format of machine learning or deep learning package.

Author(s)

Rolf Simoes, <rolfsimoes@gmail.com>

Examples

```
if (sits_run_examples()) {
  # create a classification model
  rfor_model <- sits_train(samples_modis_ndvi, sits_rfor())
  # export the model
  rfor_object <- sits_model_export(rfor_model)
}
```

sits_mosaic	<i>Mosaic classified cubes</i>
-------------	--------------------------------

Description

Creates a mosaic of all tiles of a sits cube. Mosaics can be created from both regularized ARD images or from classified maps. In the case of ARD images, a mosaic will be produce for each band/date combination. It is better to first regularize the data cubes and then use sits_mosaic.

Usage

```
sits_mosaic(
  cube,
  crs = "EPSG:3857",
  roi = NULL,
  multicores = 2L,
  output_dir,
  res = NULL,
  version = "v1",
```

```
    progress = TRUE
  )
```

Arguments

<code>cube</code>	A sits data cube.
<code>crs</code>	A target coordinate reference system of raster mosaic. The provided crs could be a string (e.g. "EPSG:4326" or a proj4string), or an EPSG code number (e.g. 4326). Default is "EPSG:3857" - WGS 84 / Pseudo-Mercator.
<code>roi</code>	Region of interest (see below).
<code>multicores</code>	Number of cores that will be used to crop the images in parallel.
<code>output_dir</code>	Directory for output images.
<code>res</code>	Spatial resolution of the mosaic. Default is NULL.
<code>version</code>	Version of resulting image (in the case of multiple tests)
<code>progress</code>	Show progress bar? Default is TRUE.

Value

a sits cube with only one tile.

Note

To define a `roi` use one of:

- A path to a shapefile with polygons;
- A `sfc` or `sf` object from `sf` package;
- A `SpatExtent` object from `terra` package;
- A named vector ("`lon_min`", "`lat_min`", "`lon_max`", "`lat_max`") in WGS84;
- A named vector ("`xmin`", "`xmax`", "`ymin`", "`ymax`") with XY coordinates.

The user should specify the CRS of the mosaic. We use "EPSG:3857" (Pseudo-Mercator) as the default.

Author(s)

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

Rolf Simoes, <rolfsimoes@gmail.com>

Felipe Carlos, <efelipecarlos@gmail.com>

Examples

```
if (sits_run_examples()) {
  # create a random forest model
  rfor_model <- sits_train(samples_modis_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 = "MOD13Q1-6.1",
        data_dir = data_dir
    )
    # classify a data cube
    probs_cube <- sits_classify(
        data = cube, ml_model = rfor_model, output_dir = tempdir()
    )
    # smooth the probability cube using Bayesian statistics
    bayes_cube <- sits_smooth(probs_cube, output_dir = tempdir())
    # label the probability cube
    label_cube <- sits_label_classification(
        bayes_cube,
        output_dir = tempdir()
    )
    # create roi
    roi <- sf::st_sfc(
        sf::st_polygon(
            list(rbind(
                c(-55.64768, -11.68649),
                c(-55.69654, -11.66455),
                c(-55.62973, -11.61519),
                c(-55.64768, -11.68649)
            ))
        ),
        crs = "EPSG:4326"
    )
    # crop and mosaic classified image
    mosaic_cube <- sits_mosaic(
        cube = label_cube,
        roi = roi,
        crs = "EPSG:4326",
        output_dir = tempdir()
    )
}

```

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 $y \sim s(x)$, 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 R dtwSat package, which is also described in the reference paper.

Usage

```
sits_patterns(data = NULL, freq = 8L, 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.

Author(s)

Victor Maus, <vwmaus1@gmail.com>
Gilberto Camara, <gilberto.camara@inpe.br>
Rolf Simoes, <rolfsimoes@gmail.com>

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](https://doi.org/10.1109/JSTARS.2016.2517118).

Maus, V., Câmara, G., Appel, M., & Pebesma, E. (2019). dtwSat: Time-Weighted Dynamic Time Warping for Satellite Image Time Series Analysis in R. Journal of Statistical Software, 88(5), 1–31. doi:[10.18637/jss.v088.i05](https://doi.org/10.18637/jss.v088.i05).

Examples

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

sits_predictors	<i>Obtain predictors for time series samples</i>
-----------------	--

Description

Predictors are X-Y values required for machine learning algorithms, organized as a data table where each row corresponds to a training sample. The first two columns of the predictors table are categorical (label_id and label). The other columns are the values of each band and time, organized first by band and then by time.

Usage

```
sits_predictors(samples)
```

Arguments

samples	Time series in sits format (tibble of class "sits")
---------	---

Value

The predictors for the sample: a data.frame with one row per sample.

Author(s)

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

Examples

```
if (sits_run_examples()) {
  # Include a new machine learning function (multiple linear regression)
  # function that returns mlr model based on a sits sample tibble

  sits_mlr <- function(samples = NULL, formula = sits_formula_linear(),
                        n_weights = 20000, maxit = 2000) {
    # create a training function
    train_fun <- function(samples) {
      # Data normalization
      ml_stats <- sits_stats(samples)
      train_samples <- sits_predictors(samples)
      train_samples <- sits_pred_normalize(
        pred = train_samples,
        stats = ml_stats
      )
      formula <- formula(train_samples[, -1])
      # call method and return the trained model
      result_mlr <- nnet::multinom(
        formula = formula,
        data = train_samples,
        maxit = maxit,
```

```

        MaxNWts = n_weights,
        trace = FALSE,
        na.action = stats::na.fail
    )

    # construct model predict closure function and returns
    predict_fun <- function(values) {
        # retrieve the prediction (values and probs)
        prediction <- tibble::as_tibble(
            stats::predict(result_mlr,
                newdata = values,
                type = "probs"
            )
        )
        return(prediction)
    }
    class(predict_fun) <- c("sits_model", class(predict_fun))
    return(predict_fun)
}
result <- sits_factory_function(samples, train_fun)
return(result)
}
# create an mlr model using a set of samples
mlr_model <- sits_train(samples_modis_ndvi, sits_mlr)
# classify a point
point_ndvi <- sits_select(point_mt_6bands, bands = "NDVI")
point_class <- sits_classify(point_ndvi, mlr_model, multicores = 1)
plot(point_class)
}

```

sits_pred_features	<i>Obtain numerical values of predictors for time series samples</i>
--------------------	--

Description

Predictors are X-Y values required for machine learning algorithms, organized as a data table where each row corresponds to a training sample. The first two columns of the predictors table are categorical ("label_id" and "label"). The other columns are the values of each band and time, organized first by band and then by time. This function returns the numeric values associated to each sample.

Usage

```
sits_pred_features(pred)
```

Arguments

pred	X-Y predictors: a data.frame with one row per sample.
------	---

Value

The Y predictors for the sample: data.frame with one row per sample.

Author(s)

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

Examples

```
if (sits_run_examples()) {
  pred <- sits_predictors(samples_modis_ndvi)
  features <- sits_pred_features(pred)
}
```

sits_pred_normalize *Normalize predictor values*

Description

Most machine learning algorithms require data to be normalized. This applies to the "SVM" method and to all deep learning ones. To normalize the predictors, it is required that the statistics per band for each sample have been obtained by the "sits_stats" function.

Usage

```
sits_pred_normalize(pred, stats)
```

Arguments

pred	X-Y predictors: a data.frame with one row per sample.
stats	Values of time series for Q02 and Q98 of the data (list of numeric values with two elements)

Value

A data.frame with normalized predictor values

Author(s)

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

Examples

```
if (sits_run_examples()) {
  stats <- sits_stats(samples_modis_ndvi)
  pred <- sits_predictors(samples_modis_ndvi)
  pred_norm <- sits_pred_normalize(pred, stats)
}
```

sits_pred_references	<i>Obtain categorical id and predictor labels for time series samples</i>
----------------------	---

Description

Predictors are X-Y values required for machine learning algorithms, organized as a data table where each row corresponds to a training sample. The first two columns of the predictors table are categorical ("label_id" and "label"). The other columns are the values of each band and time, organized first by band and then by time. This function returns the numeric values associated to each sample.

Usage

```
sits_pred_references(pred)
```

Arguments

pred	X-Y predictors: a data.frame with one row per sample.
------	---

Value

A character vector with labels associated to training samples.

Author(s)

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

Examples

```
if (sits_run_examples()) {  
  pred <- sits_predictors(samples_modis_ndvi)  
  ref <- sits_pred_references(pred)  
}
```

sits_pred_sample	<i>Obtain a fraction of the predictors data frame</i>
------------------	---

Description

Many machine learning algorithms (especially deep learning) use part of the original samples as test data to adjust its hyperparameters and to find an optimal point of convergence using gradient descent. This function extracts a fraction of the predictors to serve as test values for the deep learning algorithm.

Usage

```
sits_pred_sample(pred, frac)
```

Arguments

pred X-Y predictors: a data.frame with one row per sample.
 frac Fraction of the X-Y predictors to be extracted

Value

A data.frame with the chosen fraction of the X-Y predictors.

Author(s)

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

Examples

```
if (sits_run_examples()) {
  pred <- sits_predictors(samples_modis_ndvi)
  pred_frac <- sits_pred_sample(pred, frac = 0.5)
}
```

sits_reclassify	<i>Reclassify a classified cube</i>
-----------------	-------------------------------------

Description

Apply a set of named expressions to reclassify a classified image. The expressions should use character values to refer to labels in logical expressions.

Usage

```
sits_reclassify(cube, ...)

## S3 method for class 'class_cube'
sits_reclassify(
  cube,
  ...,
  mask,
  rules,
  memsize = 4L,
  multicores = 2L,
  output_dir,
  version = "v1",
  progress = TRUE
)

## Default S3 method:
sits_reclassify(cube, ...)
```

Arguments

<code>cube</code>	Image cube to be reclassified (class = "class_cube")
<code>...</code>	Other parameters for specific functions.
<code>mask</code>	Image cube with additional information to be used in expressions (class = "class_cube").
<code>rules</code>	Expressions to be evaluated (named list).
<code>memsize</code>	Memory available for classification in GB (integer, min = 1, max = 16384).
<code>multicores</code>	Number of cores to be used for classification (integer, min = 1, max = 2048).
<code>output_dir</code>	Directory where files will be saved (character vector of length 1 with valid location).
<code>version</code>	Version of resulting image (character).
<code>progress</code>	Set progress bar??

Value

An object of class "class_cube" (reclassified cube).

Note

Reclassification of a remote sensing map refers to changing the classes assigned to different pixels in the image. Reclassification involves assigning new classes to pixels based on additional information from a reference map. Users define rules according to the desired outcome. These rules are then applied to the classified map to produce a new map with updated classes.

`sits_reclassify()` allow any valid R expression to compute reclassification. User should refer to `cube` and `mask` to construct logical expressions. Users can use any R expression that evaluates to logical. TRUE values will be relabeled to expression name. Updates are done in asynchronous manner, that is, all expressions are evaluated using original classified values. Expressions are evaluated sequentially and resulting values are assigned to output cube. Last expressions has precedence over first ones.

Author(s)

Rolf Simoes, <rolfsimoes@gmail.com>

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

Examples

```
if (sits_run_examples()) {
  # Open mask map
  data_dir <- system.file("extdata/raster/prodes", package = "sits")
  prodes2021 <- sits_cube(
    source = "USGS",
    collection = "LANDSAT-C2L2-SR",
    data_dir = data_dir,
    parse_info = c(
      "X1", "X2", "tile", "start_date", "end_date",
      "band", "version"
    ),
  ),
```

```

bands = "class",
version = "v20220606",
labels = c(
  "1" = "Forest", "2" = "Water", "3" = "NonForest",
  "4" = "NonForest2", "6" = "d2007", "7" = "d2008",
  "8" = "d2009", "9" = "d2010", "10" = "d2011",
  "11" = "d2012", "12" = "d2013", "13" = "d2014",
  "14" = "d2015", "15" = "d2016", "16" = "d2017",
  "17" = "d2018", "18" = "r2010", "19" = "r2011",
  "20" = "r2012", "21" = "r2013", "22" = "r2014",
  "23" = "r2015", "24" = "r2016", "25" = "r2017",
  "26" = "r2018", "27" = "d2019", "28" = "r2019",
  "29" = "d2020", "31" = "r2020", "32" = "Clouds2021",
  "33" = "d2021", "34" = "r2021"
),
progress = FALSE
)
#' Open classification map
data_dir <- system.file("extdata/raster/classif", package = "sits")
ro_class <- sits_cube(
  source = "MPC",
  collection = "SENTINEL-2-L2A",
  data_dir = data_dir,
  parse_info = c(
    "X1", "X2", "tile", "start_date", "end_date",
    "band", "version"
  ),
  bands = "class",
  labels = c(
    "1" = "ClearCut_Fire", "2" = "ClearCut_Soil",
    "3" = "ClearCut_Veg", "4" = "Forest"
  ),
  progress = FALSE
)
# Reclassify cube
ro_mask <- sits_reclassify(
  cube = ro_class,
  mask = prodes2021,
  rules = list(
    "Old_Deforestation" = mask %in% c(
      "d2007", "d2008", "d2009",
      "d2010", "d2011", "d2012",
      "d2013", "d2014", "d2015",
      "d2016", "d2017", "d2018",
      "r2010", "r2011", "r2012",
      "r2013", "r2014", "r2015",
      "r2016", "r2017", "r2018",
      "d2019", "r2019", "d2020",
      "r2020", "r2021"
    ),
    "Water_Mask" = mask == "Water",
    "NonForest_Mask" = mask %in% c("NonForest", "NonForest2")
  ),

```

```

    memsize = 4,
    multicores = 2,
    output_dir = tempdir(),
    version = "ex_reclassify"
  )
}

```

sits_reduce

Reduces a cube or samples from a summarization function

Description

Apply a temporal reduction from a named expression in cube or sits tibble. In the case of cubes, it materializes a new band in output_dir. The result will be a cube with only one date with the raster reduced from the function.

Usage

```

sits_reduce(data, ...)

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

## S3 method for class 'raster_cube'
sits_reduce(
  data,
  ...,
  impute_fn = impute_linear(),
  memsize = 4L,
  multicores = 2L,
  output_dir,
  progress = TRUE
)

```

Arguments

data	Valid sits tibble or cube
...	Named expressions to be evaluated (see details).
impute_fn	Imputation function to remove NA values.
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_reduce()` allows 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. The provided functions must operate at line level in order to perform temporal reduction on a pixel.

`sits_reduce()` Applies a function to each row of a matrix. In this matrix, each row represents a pixel and each column represents a single date. We provide some operations already implemented in the package to perform the reduce operation. See the list of available functions below:

Value

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

Summarizing temporal functions

- `t_max()`: Returns the maximum value of the series.
- `t_min()`: Returns the minimum value of the series
- `t_mean()`: Returns the mean of the series.
- `t_median()`: Returns the median of the series.
- `t_std()`: Returns the standard deviation of the series.
- `t_skewness()`: Returns the skewness of the series.
- `t_kurtosis()`: Returns the kurtosis of the series.
- `t_amplitude()`: Returns the difference between the maximum and minimum values of the cycle. A small amplitude means a stable cycle.
- `t_fslope()`: Returns the maximum value of the first slope of the cycle. Indicates when the cycle presents an abrupt change in the curve. The slope between two values relates the speed of the growth or senescence phases
- `t_mse()`: Returns the average spectral energy density. The energy of the time series is distributed by frequency.
- `t_fqr()`: Returns the value of the first quartile of the series (0.25).
- `t_tqr()`: Returns the value of the third quartile of the series (0.75).
- `t_iqr()`: Returns the interquartile range (difference between the third and first quartiles).

Note

The `t_sum()`, `t_std()`, `t_skewness()`, `t_kurtosis`, `t_mse` indexes generate values greater than the limit of a two-byte integer. Therefore, we save the images generated by these as Float-32 with no scale.

Author(s)

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

Rolf Simoes, <rolfsimoes@gmail.com>

Examples

```
if (sits_run_examples()) {
  # Reduce summarization function

  point2 <-
    sits_select(point_mt_6bands, "NDVI") |>
    sits_reduce(NDVI_MEDIAN = t_median(NDVI))

  # Example of generation mean summarization from a cube
  # Create a data cube from local files
  data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
  cube <- sits_cube(
    source = "BDC",
    collection = "MOD13Q1-6.1",
    data_dir = data_dir
  )

  # Reduce NDVI band with mean function
  cube_mean <- sits_reduce(
    data = cube,
    NDVIMEAN = t_mean(NDVI),
    output_dir = tempdir()
  )
}
```

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 = 200L,
  n_samples_under = 400L,
  method = "smote",
  multicores = 2L
)
```

Arguments

samples Sample set to rebalance

n_samples_over	Number of samples to oversample for classes with samples less than this number.
n_samples_under	Number of samples to undersample for classes with samples more than this number.
method	Method for oversampling (default = "smote")
multicores	Number of cores to process the data (default 2).

Value

A sits tibble with reduced sample imbalance.

Note

Many training samples for Earth observation data analysis are imbalanced. This situation arises when the distribution of samples associated with each label is uneven. Sample imbalance is an undesirable property of a training set. Reducing sample imbalance improves classification accuracy.

The function `sits_reduce_imbalance` increases the number of samples of least frequent labels, and reduces the number of samples of most frequent labels. To generate new samples, `sits` uses the SMOTE method that estimates new samples by considering the cluster formed by the nearest neighbors of each minority label.

To perform undersampling, `sits_reduce_imbalance` builds a SOM map for each majority label based on the required number of samples. Each dimension of the SOM is set to $\text{ceiling}(\sqrt{\text{new_number_samples}/4})$ to allow a reasonable number of neurons to group similar samples. After calculating the SOM map, the algorithm extracts four samples per neuron to generate a reduced set of samples that approximates the variation of the original one. See also [sits_som_map](#).

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, [doi:10.1613/jair.953](#).

The SOM map technique for time series 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. [doi:10.1016/j.isprsjprs.2021.04.014](#).

Examples

```
if (sits_run_examples()) {
  # print the labels summary for a sample set
  summary(samples_modis_ndvi)
  # reduce the sample imbalance
  new_samples <- sits_reduce_imbalance(samples_modis_ndvi,
```



```

        n_samples_over = 200,
        n_samples_under = 200,
        multicores = 1
    )
    # print the labels summary for the rebalanced set
    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. This function uses the `gdalcubes` package.

Usage

```

sits_regularize(cube, ...)

## S3 method for class 'raster_cube'
sits_regularize(
  cube,
  ...,
  period,
  res,
  output_dir,
  timeline = NULL,
  roi = NULL,
  crs = NULL,
  tiles = NULL,
  grid_system = NULL,
  multicores = 2L,
  progress = TRUE
)

## S3 method for class 'sar_cube'
sits_regularize(
  cube,
  ...,
  period,
  res,
  output_dir,

```

```
    timeline = NULL,
    grid_system = "MGRS",
    roi = NULL,
    crs = NULL,
    tiles = NULL,
    multicores = 2L,
    progress = TRUE
)

## S3 method for class 'combined_cube'
sits_regularize(
  cube,
  ...,
  period,
  res,
  output_dir,
  grid_system = NULL,
  roi = NULL,
  crs = NULL,
  tiles = NULL,
  multicores = 2L,
  progress = TRUE
)

## S3 method for class 'rainfall_cube'
sits_regularize(
  cube,
  ...,
  period,
  res,
  output_dir,
  timeline = NULL,
  grid_system = "MGRS",
  roi = NULL,
  crs = NULL,
  tiles = NULL,
  multicores = 2L,
  progress = TRUE
)

## S3 method for class 'dem_cube'
sits_regularize(
  cube,
  ...,
  res,
  output_dir,
  grid_system = "MGRS",
  roi = NULL,
```

```

        crs = NULL,
        tiles = NULL,
        multicores = 2L,
        progress = TRUE
    )

## S3 method for class 'ogh_cube'
sits_regularize(
    cube,
    ...,
    period,
    res,
    output_dir,
    timeline = NULL,
    grid_system = "MGRS",
    roi = NULL,
    crs = NULL,
    tiles = NULL,
    multicores = 2L,
    progress = TRUE
)

## S3 method for class 'derived_cube'
sits_regularize(cube, ...)

## Default S3 method:
sits_regularize(cube, ...)

```

Arguments

<code>cube</code>	raster_cube object whose observation period and/or spatial resolution is not constant.
<code>...</code>	Additional parameters.
<code>period</code>	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.
<code>res</code>	Spatial resolution of regularized images (in meters).
<code>output_dir</code>	Valid directory for storing regularized images.
<code>timeline</code>	User-defined timeline for regularized cube.
<code>roi</code>	Region of interest (see notes below).
<code>crs</code>	Coordinate Reference System (CRS) of the roi. (see details below).
<code>tiles</code>	Tiles to be produced.
<code>grid_system</code>	Grid system to be used for the output images.
<code>multicores</code>	Number of cores used for regularization; used for parallel processing of input (integer)
<code>progress</code>	show progress bar?

Value

A raster_cube object with aggregated images.

Note

The main sits classification workflow has the following steps:

1. `sits_cube`: selects a ARD image collection from a cloud provider.
2. `sits_cube_copy`: copies an ARD image collection from a cloud provider to a local directory for faster processing.
3. `sits_regularize`: create a regular data cube from an ARD image collection.
4. `sits_apply`: create new indices by combining bands of a regular data cube (optional).
5. `sits_get_data`: extract time series from a regular data cube based on user-provided labelled samples.
6. `sits_train`: train a machine learning model based on image time series.
7. `sits_classify`: classify a data cube using a machine learning model and obtain a probability cube.
8. `sits_smooth`: post-process a probability cube using a spatial smoother to remove outliers and increase spatial consistency.
9. `sits_label_classification`: produce a classified map by selecting the label with the highest probability from a smoothed cube.

The regularization operation converts subsets of image collections available in cloud providers into regular data cubes. It is an essential part of the sits workflow. The input to `sits_regularize` should be an ARD cube which includes the cloud band. The aggregation method used in `sits_regularize` sorts the images based on cloud cover, putting images with the least 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 "period" parameter is mandatory, and defines the time interval between two images of the regularized cube. When combining Sentinel-1A and Sentinel-1B images, experiments show that a 16-day period ("P16D") are a good default. Landsat images require a longer period of one to three months.

By default, the date of the first image of the input cube is taken as the starting date for the regular cube. In many situations, users may want to pre-define the required times using the "timeline" parameter. The "timeline" parameter, if used, must contain a set of dates which are compatible with the input cube.

To define a roi use one of:

- A path to a shapefile with polygons;
- A sfc or sf object from sf package;
- A SpatExtent object from terra package;
- A named vector ("lon_min", "lat_min", "lon_max", "lat_max") in WGS84;
- A named vector ("xmin", "xmax", "ymin", "ymax") with XY coordinates.

Defining a region of interest using `SpatExtent` or `XY` values not in WGS84 requires the `crs` parameter to be specified. `sits_regularize()` function will crop the images that contain the region of interest().

The optional `tiles` parameter indicates which tiles of the input cube will be used for regularization.

The `grid_system` parameter allows the user to reproject the files to a grid system which is different from that used in the ARD image collection of the cloud provider. Currently, the package supports the use of MGRS grid system and those used by the Brazil Data Cube ("BDC_LG_V2" "BDC_MD_V2" "BDC_SM_V2").

Author(s)

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

Rolf Simoes, <rolfsimoes@gmail.com>

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](https://doi.org/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-2-L2A",
    tiles = c("20LKP", "20LLP"),
    bands = c("B8A", "CLOUD"),
    start_date = "2018-10-01",
    end_date = "2018-11-01"
  )
  # regularize the cube
  rg_cube <- sits_regularize(
    cube = s2_cube_open,
    period = "P16D",
    res = 60,
    multicores = 2,
    output_dir = tempdir()
  )

  ## Sentinel-1 SAR
  roi <- c(
    "lon_min" = -50.410, "lon_max" = -50.379,
    "lat_min" = -10.1910, "lat_max" = -10.1573
  )
  s1_cube_open <- sits_cube(
    source = "MPC",
    collection = "SENTINEL-1-GRD",
    bands = c("VV", "VH"),
    orbit = "descending",
    roi = roi,
```

```

        start_date = "2020-06-01",
        end_date = "2020-09-28"
    )
    # regularize the cube
    rg_cube <- sits_regularize(
      cube = sl_cube_open,
      period = "P12D",
      res = 60,
      roi = roi,
      multicores = 2,
      output_dir = tempdir()
    )
  }

```

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 = torch::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,
    seed = NULL,
    verbose = FALSE
)

```

Arguments

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

Value

A fitted model to be used for classification.

Author(s)

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

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

Felipe Souza, <lipecaso@gmail.com>

Felipe Carlos, <efelipecarlos@gmail.com>

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()) {
  # create a ResNet model
  torch_model <- sits_train(samples_modis_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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # classify a data cube
  probs_cube <- sits_classify(
    data = cube, ml_model = torch_model, output_dir = tempdir()
  )
  # plot the probability cube
  plot(probs_cube)
  # smooth the probability cube using Bayesian statistics
  bayes_cube <- sits_smooth(probs_cube, output_dir = tempdir())
  # plot the smoothed cube
  plot(bayes_cube)
  # label the probability cube
  label_cube <- sits_label_classification(
    bayes_cube, output_dir = tempdir()
  )
  # 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 = 100L, mtry = NULL, ...)
```

Arguments

<code>samples</code>	Time series with the training samples (tibble of class "sits").
<code>num_trees</code>	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: 100) (integer, min = 50, max = 150).
<code>mtry</code>	Number of variables randomly sampled as candidates at each split (default: NULL - use default value of <code>randomForest::randomForest()</code> function, i.e. <code>floor(sqrt(features))</code>).
<code>...</code>	Other parameters to be passed to 'randomForest::randomForest' function.

Value

Model fitted to input data (to be passed to [sits_classify](#)).

Author(s)

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

Rolf Simoes, <rolfsimoes@gmail.com>

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_ndvi,
    ml_method = sits_rfor
  )
  # classify the point
  point_ndvi <- sits_select(point_mt_6bands, bands = "NDVI")
  # classify the point
  point_class <- sits_classify(
    data = point_ndvi, ml_model = rf_model
  )
  plot(point_class)
}
```

sits_roi_to_mgrs

Given a ROI, find MGRS tiles intersecting it.

Description

Takes a ROI and produces a list of MGRS tiles intersecting it

Usage

```
sits_roi_to_mgrs(roi)
```

Arguments

roi Valid ROI to use in other SITS functions

Value

tiles Character vector with names of MGRS tiles

Note

To define a roi use one of:

- A path to a shapefile with polygons;
- A sfc or sf object from sf package;
- A SpatExtent object from terra package;
- A named vector ("lon_min", "lat_min", "lon_max", "lat_max") in WGS84;
- A named vector ("xmin", "xmax", "ymin", "ymax") with XY coordinates.

Defining a region of interest using SpatExtent or XY values not in WGS84 requires the crs parameter to be specified.

Author(s)

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

Felipe Carlos, <efelipeccarlos@gmail.com>

Examples

```
if (sits_run_examples()) {
# Defining a ROI
roi <- c(
  lon_min = -64.037,
  lat_min = -9.644,
  lon_max = -63.886,
  lat_max = -9.389
)
# Finding tiles
```

```

tiles <- sits_roi_to_mgrs(roi)
}

```

sits_roi_to_tiles	<i>Find tiles of a given ROI and Grid System</i>
-------------------	--

Description

Given an ROI and grid system, this function finds the intersected tiles and returns them as an SF object.

Usage

```
sits_roi_to_tiles(roi, crs = NULL, grid_system = "MGRS")
```

Arguments

roi	Region of interest (see notes below).
crs	Coordinate Reference System (CRS) of the roi. (see details below).
grid_system	Grid system to be used for the output images. (Default is "MGRS")

Value

A sf object with the intersect tiles with three columns tile_id, epsg, and the percentage of coverage area.

Note

To define a roi use one of:

- A path to a shapefile with polygons;
- A sfc or sf object from sf package;
- A SpatExtent object from terra package;
- A named vector ("lon_min", "lat_min", "lon_max", "lat_max") in WGS84;
- A named vector ("xmin", "xmax", "ymin", "ymax") with XY coordinates.

Defining a region of interest using SpatExtent or XY values not in WGS84 requires the crs parameter to be specified.

The grid_system parameter allows the user to reproject the files to a grid system which is different from that used in the ARD image collection of the could provider. Currently, the package supports the use of MGRS grid system and those used by the Brazil Data Cube ("BDC_LG_V2" "BDC_MD_V2" "BDC_SM_V2").

Author(s)

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

Felipe Carlos, <efelipeccarlos@gmail.com>

Examples

```

if (sits_run_examples()) {
# Defining a ROI
roi <- c(
  lon_min = -64.037,
  lat_min = -9.644,
  lon_max = -63.886,
  lat_max = -9.389
)
# Finding tiles
tiles <- sits_roi_to_tiles(roi, grid_system = "MGRS")
}

```

sits_run_examples	<i>Informs if sits examples should run</i>
-------------------	--

Description

This function informs if sits examples should run. To run the examples, set "SITS_RUN_EXAMPLES" to "YES" using Sys.setenv("SITS_RUN_EXAMPLES" = "YES") To come back to the default behaviour, please set Sys.setenv("SITS_RUN_EXAMPLES" = "NO")

Usage

```
sits_run_examples()
```

Value

A logical value

sits_run_tests	<i>Informs if sits tests should run</i>
----------------	---

Description

To run the tests, set "SITS_RUN_TESTS" environment to "YES" using Sys.setenv("SITS_RUN_TESTS" = "YES") To come back to the default behaviour, please set Sys.setenv("SITS_RUN_TESTS" = "NO")

Usage

```
sits_run_tests()
```

Value

TRUE/FALSE

sits_sample	<i>Sample a percentage of a time series</i>
-------------	---

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 percentage of the total number of samples per group. If `frac > 1`, all sampling will be done with replacement.

Usage

```
sits_sample(data, frac = 0.2, oversample = TRUE)
```

Arguments

<code>data</code>	Sits time series tibble
<code>frac</code>	Percentage of samples to extract (range: 0.0 to 2.0, default = 0.2)
<code>oversample</code>	Logical: oversample classes with small number of samples? (TRUE/FALSE)

Value

A sits tibble with a fixed quantity of samples.

Author(s)

Rolf Simoes, <rolfsimoes@gmail.com>

Examples

```
# Retrieve a set of time series with 2 classes
data(cerrado_2classes)
# Print the labels of the resulting tibble
summary(cerrado_2classes)
# Sample by fraction
data_02 <- sits_sample(cerrado_2classes, frac = 0.2)
# Print the labels
summary(data_02)
```

sits_sampling_design *Allocation of sample size to strata*

Description

Takes a class cube with different labels and allocates a number of sample sizes per strata to obtain suitable values of error-adjusted area, providing five allocation strategies.

Usage

```
sits_sampling_design(
  cube,
  expected_ua = 0.75,
  alloc_options = c(100L, 75L, 50L),
  std_err = 0.01,
  rare_class_prop = 0.1
)
```

Arguments

cube	Classified cube
expected_ua	Expected values of user's accuracy
alloc_options	Fixed sample allocation for rare classes
std_err	Standard error we would like to achieve
rare_class_prop	Proportional area limit for rare classes

Value

A matrix with options to decide allocation of sample size to each class. This matrix uses the same format as Table 5 of Olofsson et al.(2014).

Author(s)

Gilberto Camara, <gilberto.camara@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.

Examples

```
if (sits_run_examples()) {
  # create a random forest model
  rfor_model <- sits_train(samples_modis_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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # classify a data cube
  probs_cube <- sits_classify(
    data = cube, ml_model = rfor_model, output_dir = tempdir()
  )
  # label the probability cube
  label_cube <- sits_label_classification(
    probs_cube,
    output_dir = tempdir()
  )
  # estimated UA for classes
  expected_ua <- c(
    Cerrado = 0.75, Forest = 0.9,
    Pasture = 0.8, Soy_Corn = 0.8
  )
  sampling_design <- sits_sampling_design(label_cube, expected_ua)
}
```

sits_segment

Segment an image

Description

Apply a spatial-temporal segmentation on a data cube based on a user defined segmentation function. The function applies the segmentation algorithm "seg_fn" to each tile. The output is a vector data cube, which is a data cube with an additional vector file in "geopackage" format.

Usage

```
sits_segment(
  cube,
  seg_fn = sits_slic(),
  roi = NULL,
  impute_fn = impute_linear(),
  start_date = NULL,
  end_date = NULL,
  memsize = 4L,
  multicores = 2L,
  output_dir,
```

```

    version = "v1",
    progress = TRUE
  )

```

Arguments

<code>cube</code>	Regular data cube
<code>seg_fn</code>	Function to apply the segmentation
<code>roi</code>	Region of interest (see below)
<code>impute_fn</code>	Imputation function to remove NA values.
<code>start_date</code>	Start date for the segmentation
<code>end_date</code>	End date for the segmentation.
<code>memsizes</code>	Memory available for classification (in GB).
<code>multicores</code>	Number of cores to be used for classification.
<code>output_dir</code>	Directory for output file.
<code>version</code>	Version of the output (for multiple segmentations).
<code>progress</code>	Show progress bar?

Value

A tibble of class 'segs_cube' representing the segmentation.

Note

Segmentation requires the following steps:

1. Create a regular data cube with [sits_cube](#) and [sits_regularize](#);
2. Run [sits_segment](#) to obtain a vector data cube with polygons that define the boundary of the segments;
3. Classify the time series associated to the segments with [sits_classify](#), to get obtain a vector probability cube;
4. Use [sits_label_classification](#) to label the vector probability cube;
5. Display the results with [plot](#) or [sits_view](#).

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").

As of version 1.5.3, the only `seg_fn` function available is [sits_slic](#), which uses the Simple Linear Iterative Clustering (SLIC) algorithm that clusters pixels to generate compact, nearly uniform superpixels. This algorithm has been adapted by Nowosad and Stepinski to work with multispectral and multitemporal images. SLIC uses spectral similarity and proximity in the spectral and temporal space to segment the image into superpixels. Superpixels are clusters of pixels with similar spectral and temporal responses that are spatially close.

The result of `sits_segment` is a data cube tibble with an additional vector file in the geopackage format. The location of the vector file is included in the data cube tibble in a new column, called `vector_info`.

Author(s)

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

Rolf Simoes, <rolfsimoes@gmail.com>

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

Felipe Carlos, <efelipecarlos@gmail.com>

References

Achanta, Radhakrishna, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. 2012. "SLIC Superpixels Compared to State-of-the-Art Superpixel Methods." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34 (11): 2274–82.

Nowosad, Jakub, and Tomasz F. Stepinski. 2022. "Extended SLIC Superpixels Algorithm for Applications to Non-Imagery Geospatial Rasters." *International Journal of Applied Earth Observation and Geoinformation* 112 (August): 102935.

Examples

```
if (sits_run_examples()) {
  data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
  # create a data cube
  cube <- sits_cube(
    source = "BDC",
    collection = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # segment the vector cube
  segments <- sits_segment(
    cube = cube,
    seg_fn = sits_slic(
      step = 10,
      compactness = 1,
      dist_fun = "euclidean",
      avg_fun = "median",
      iter = 30,
      minarea = 10
    ),
    output_dir = tempdir()
  )
  # create a classification model
  rfor_model <- sits_train(samples_modis_ndvi, sits_rfor())
  # classify the segments
  seg_probs <- sits_classify(
    data = segments,
    ml_model = rfor_model,
    output_dir = tempdir()
  )
  # label the probability segments
  seg_label <- sits_label_classification(
    cube = seg_probs,
    output_dir = tempdir()
  )
}
```

```
    )
  }
```

sits_select

Filter a data set (tibble or cube) for bands, tiles, and dates

Description

Filter the bands, tiles, dates and labels from a set of time series or from a data cube.

Usage

```
sits_select(data, ...)

## S3 method for class 'sits'
sits_select(
  data,
  ...,
  bands = NULL,
  start_date = NULL,
  end_date = NULL,
  dates = NULL,
  labels = NULL
)

## S3 method for class 'raster_cube'
sits_select(
  data,
  ...,
  bands = NULL,
  start_date = NULL,
  end_date = NULL,
  dates = NULL,
  tiles = NULL
)

## Default S3 method:
sits_select(data, ...)
```

Arguments

data	Tibble with time series or data cube.
...	Additional parameters to be provided
bands	Character vector with the names of the bands.
start_date	Date in YYYY-MM-DD format: start date to be filtered.
end_date	Date in YYYY-MM-DD format: end date to be filtered.

dates	Character vector with sparse dates to be selected.
labels	Character vector with sparse labels to be selected (Only applied for sits tibble data).
tiles	Character vector with the names of the tiles.

Value

Tibble with time series or data cube.

Author(s)

Rolf Simoes, <rolfsimoes@gmail.com>

Felipe Carlos, <efelipecarlos@gmail.com>

Felipe Carvalho, <felipe.carvalho@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)
# select start and end date
point_2010 <- sits_select(point_mt_6bands,
  start_date = "2000-09-13",
  end_date = "2017-08-29"
)
```

sits_sgolay

Filter time series with Savitzky-Golay filter

Description

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_sgolay(data = NULL, order = 3L, length = 5L)
```

Arguments

data	Time series or matrix.
order	Filter order (integer).
length	Filter length (must be odd).

Value

Filtered time series

Author(s)

Rolf Simoes, <rolfsimoes@gmail.com>
 Gilberto Camara, <gilberto.camara@inpe.br>
 Felipe Carvalho, <felipe.carvalho@inpe.br>

References

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

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 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_slic

Segment an image using SLIC

Description

Apply a segmentation on a data cube based on the supercells package. This is an adaptation and extension to remote sensing data of the SLIC superpixels algorithm proposed by Achanta et al. (2012). See references for more details.

Usage

```
sits_slic(
  data = NULL,
  step = 30L,
  compactness = 1,
  dist_fun = "euclidean",
  avg_fun = "median",
  iter = 30L,
  minarea = 10L,
  verbose = FALSE
)
```

Arguments

<code>data</code>	A matrix with time series.
<code>step</code>	Distance (in number of cells) between initial supercells' centers.
<code>compactness</code>	A compactness value. Larger values cause clusters to be more compact/even (square).
<code>dist_fun</code>	Distance function. Currently implemented: euclidean, jsd, dtw, and any distance function from the philentropy package. See <code>philentropy::getDistMethods()</code> .
<code>avg_fun</code>	Averaging function to calculate the values of the supercells' centers. Accepts any fitting R function (e.g., <code>base::mean()</code> or <code>stats::median()</code>) or one of internally implemented "mean" and "median". Default: "median"
<code>iter</code>	Number of iterations to create the output.
<code>minarea</code>	Specifies the minimal size of a supercell (in cells).
<code>verbose</code>	Show the progress bar?

Value

Set of segments for a single tile

Author(s)

Rolf Simoes, <rolfsimoes@gmail.com>
 Felipe Carvalho, <felipe.carvalho@inpe.br>
 Felipe Carlos, <efelipecarlos@gmail.com>

References

Achanta, Radhakrishna, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. 2012. "SLIC Superpixels Compared to State-of-the-Art Superpixel Methods." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34 (11): 2274–82.

Nowosad, Jakub, and Tomasz F. Stepinski. 2022. "Extended SLIC Superpixels Algorithm for Applications to Non-Imagery Geospatial Rasters." *International Journal of Applied Earth Observation and Geoinformation* 112 (August): 102935.

Examples

```
if (sits_run_examples()) {
  data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
  # create a data cube
  cube <- sits_cube(
    source = "BDC",
    collection = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # segment the vector cube
  segments <- sits_segment(
    cube = cube,
    seg_fn = sits_slic(
      step = 10,
      compactness = 1,
      dist_fun = "euclidean",
      avg_fun = "median",
      iter = 30,
      minarea = 10
    ),
    output_dir = tempdir(),
    version = "slic-demo"
  )
  # create a classification model
  rfor_model <- sits_train(samples_modis_ndvi, sits_rfor())
  # classify the segments
  seg_probs <- sits_classify(
    data = segments,
    ml_model = rfor_model,
    output_dir = tempdir(),
    version = "slic-demo"
  )
  # label the probability segments
  seg_label <- sits_label_classification(
    cube = seg_probs,
    output_dir = tempdir(),
    version = "slic-demo"
  )
}
```

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 Bayesian smoothing function.

Usage

```
sits_smooth(cube, ...)

## S3 method for class 'probs_cube'
sits_smooth(
  cube,
  ...,
  window_size = 9L,
  neigh_fraction = 0.5,
  smoothness = 20,
  exclusion_mask = NULL,
  memsize = 4L,
  multicores = 2L,
  output_dir,
  version = "v1",
  progress = TRUE
)

## S3 method for class 'probs_vector_cube'
sits_smooth(cube, ...)

## S3 method for class 'raster_cube'
sits_smooth(cube, ...)

## S3 method for class 'derived_cube'
sits_smooth(cube, ...)

## Default S3 method:
sits_smooth(cube, ...)
```

Arguments

<code>cube</code>	Probability data cube.
<code>...</code>	Other parameters for specific functions.
<code>window_size</code>	Size of the neighborhood (integer, min = 3, max = 21)
<code>neigh_fraction</code>	Fraction of neighbors with high probabilities to be used in Bayesian inference. (numeric, min = 0.1, max = 1.0)
<code>smoothness</code>	Estimated variance of logit of class probabilities (Bayesian smoothing parameter) (integer vector or scalar, min = 1, max = 200).
<code>exclusion_mask</code>	Areas to be excluded from the classification process. It can be defined as a sf object or a shapefile.
<code>memsize</code>	Memory available for classification in GB (integer, min = 1, max = 16384).
<code>multicores</code>	Number of cores to be used for classification (integer, min = 1, max = 2048).
<code>output_dir</code>	Valid directory for output file. (character vector of length 1).
<code>version</code>	Version of the output (character vector of length 1).
<code>progress</code>	Check progress bar?

Value

A data cube.

Note

The main `sits` classification workflow has the following steps:

1. `sits_cube`: selects a ARD image collection from a cloud provider.
2. `sits_cube_copy`: copies an ARD image collection from a cloud provider to a local directory for faster processing.
3. `sits_regularize`: create a regular data cube from an ARD image collection.
4. `sits_apply`: create new indices by combining bands of a regular data cube (optional).
5. `sits_get_data`: extract time series from a regular data cube based on user-provided labelled samples.
6. `sits_train`: train a machine learning model based on image time series.
7. `sits_classify`: classify a data cube using a machine learning model and obtain a probability cube.
8. `sits_smooth`: post-process a probability cube using a spatial smoother to remove outliers and increase spatial consistency.
9. `sits_label_classification`: produce a classified map by selecting the label with the highest probability from a smoothed cube.

Machine learning algorithms rely on training samples that are derived from “pure” pixels, hand-picked by users to represent the desired output classes. Given the presence of mixed pixels in images regardless of resolution, and the considerable data variability within each class, these classifiers often produce results with misclassified pixels.

Post-processing the results of `sits_classify` using `sits_smooth` reduces salt-and-pepper and border effects. By minimizing noise, `sits_smooth` brings a significant gain in the overall accuracy and interpretability of the final output.

Author(s)

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

Rolf Simoes, <rolfsimoes@gmail.com>

References

Gilberto Camara, Renato Assunção, Alexandre Carvalho, Rolf Simões, Felipe Souza, Felipe Carlos, Anielli Souza, Ana Rorato, Ana Paula Dal’Asta, “Bayesian inference for post-processing of remote sensing image classification”. *Remote Sensing*, 16(23), 4572, 2024. doi:[10.3390/rs16234572](https://doi.org/10.3390/rs16234572).

Examples

```
if (sits_run_examples()) {
  # create an xgboost model
  xgb_model <- sits_train(samples_modis_ndvi, sits_xgboost())
  # create a data cube from local files
```



```

data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
cube <- sits_cube(
  source = "BDC",
  collection = "MOD13Q1-6.1",
  data_dir = data_dir
)
# classify a data cube
probs_cube <- sits_classify(
  data = cube, ml_model = xgb_model, output_dir = tempdir()
)
# plot the probability cube
plot(probs_cube)
# smooth the probability cube using Bayesian statistics
bayes_cube <- sits_smooth(probs_cube, output_dir = tempdir())
# plot the smoothed cube
plot(bayes_cube)
# label the probability cube
label_cube <- sits_label_classification(
  bayes_cube,
  output_dir = tempdir()
)
# plot the labelled cube
plot(label_cube)
}

```

sits_som_clean_samples

Cleans the samples based on SOM map information

Description

sits_som_clean_samples() evaluates the quality of the samples based on the results of the SOM map.

Usage

```

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

```

Arguments

som_map	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

tibble with an two additional columns. The first indicates if each sample is clean, should be analyzed or should be removed. The second is the posterior probability of the sample. The "keep" parameter indicates which

Note

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".

Author(s)

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

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

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

Examples

```
if (sits_run_examples()) {
  # create a som map
  som_map <- sits_som_map(samples_modis_ndvi)
  # 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_som_evaluate_cluster`*Evaluate cluster*

Description

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

Usage

```
sits_som_evaluate_cluster(som_map)
```

Arguments

`som_map` A SOM map produced by the `som_map()` function

Value

A tibble stating the purity for each cluster

Author(s)

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

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

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

Examples

```
if (sits_run_examples()) {  
  # create a som map  
  som_map <- sits_som_map(samples_modis_ndvi)  
  # 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_som_map

*Build a SOM for quality analysis of time series samples***Description**

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

Usage

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

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). The following similarity measurements are supported: "euclidean" and "dtw". The default similarity measure is "dtw".
som_radius	Radius of SOM neighborhood.
mode	Type of learning algorithm. The following learning algorithm are available: "online", "batch", and "pbatch". The default learning algorithm is "online".

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.

Note

`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 sample quality 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".

To learn more about the learning algorithms, check the `kohonen::supersom` function.

The `sits` package implements the "dtw" (Dynamic Time Warping) similarity measure. The "euclidean" similarity measurement come from the `kohonen::supersom(dist.fcts)` function.

Author(s)

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

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

Gilberto Camara, <gilberto.camara@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. doi:10.1016/j.isprsjprs.2021.04.014.

Examples

```
if (sits_run_examples()) {
  # create a som map
  som_map <- sits_som_map(samples_modis_ndvi)
  # 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_som_remove_samples`*Evaluate cluster*

Description

Remove samples from a given class inside a neuron of another class

Usage

```
sits_som_remove_samples(som_map, som_eval, class_cluster, class_remove)
```

Arguments

<code>som_map</code>	A SOM map produced by the <code>som_map()</code> function
<code>som_eval</code>	An evaluation produced by the <code>som_eval()</code> function
<code>class_cluster</code>	Dominant class of a set of neurons
<code>class_remove</code>	Class to be removed from the neurons of "class_cluster"

Value

A new set of samples with the desired class neurons remove

Author(s)

Lorena Alves, <lorena.santos@inpe.br>
Karine Ferreira, <karine.ferreira@inpe.br>
Gilberto Camara, <gilberto.camara@inpe.br>

Examples

```
if (sits_run_examples()) {  
  # create a som map  
  som_map <- sits_som_map(samples_modis_ndvi)  
  # evaluate the som map and create clusters  
  som_eval <- sits_som_evaluate_cluster(som_map)  
  # clean the samples  
  new_samples <- sits_som_remove_samples(  
    som_map, som_eval,  
    "Pasture", "Cerrado"  
  )  
}
```

sits_stats

Obtain statistics for all sample bands

Description

Most machine learning algorithms require data to be normalized. This applies to the "SVM" method and to all deep learning ones. To normalize the predictors, it is necessary to extract the statistics of each band of the samples. This function computes the 2 of the distribution of each band of the samples. This values are used as minimum and maximum values in the normalization operation performed by the sits_pred_normalize() function.

Usage

```
sits_stats(samples)
```

Arguments

samples Time series samples uses as training data.

Value

A list with the 2 training data.

Author(s)

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

Examples

```
if (sits_run_examples()) {
  stats <- sits_stats(samples_modis_ndvi)
}
```

sits_stratified_sampling

Allocation of sample size to strata

Description

Takes a class cube with different labels and a sampling design with a number of samples per class and allocates a set of locations for each class

Usage

```
sits_stratified_sampling(
  cube,
  sampling_design,
  alloc = "alloc_prop",
  overhead = 1.2,
  multicores = 2L,
  memsize = 2L,
  shp_file = NULL,
  progress = TRUE
)
```

Arguments

cube	Classified cube
sampling_design	Result of sits_sampling_design
alloc	Allocation method chosen
overhead	Additional percentage to account for border points
multicores	Number of cores that will be used to sample the images in parallel.
memsize	Memory available for sampling.
shp_file	Name of shapefile to be saved (optional)
progress	Show progress bar? Default is TRUE.

Value

samples Point sf object with required samples

Author(s)

Gilberto Camara, <gilberto.camara@inpe.br>
 Felipe Carlos, <efelipecarlos@gmail.com>
 Felipe Carvalho, <felipe.carvalho@inpe.br>

Examples

```
if (sits_run_examples()) {
  # create a random forest model
  rfor_model <- sits_train(samples_modis_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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # classify a data cube
}
```



```

    probs_cube <- sits_classify(
      data = cube, ml_model = rfor_model, output_dir = tempdir()
    )
    # label the probability cube
    label_cube <- sits_label_classification(
      probs_cube,
      output_dir = tempdir()
    )
    # estimated UA for classes
    expected_ua <- c(
      Cerrado = 0.95, Forest = 0.95,
      Pasture = 0.95, Soy_Corn = 0.95
    )
    # design sampling
    sampling_design <- sits_sampling_design(label_cube, expected_ua)
    # select samples
    samples <- sits_stratified_sampling(
      label_cube,
      sampling_design, "alloc_prop"
    )
  }

```

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 = 1000L,
  kernel = "radial",
  degree = 3L,
  coef0 = 0L,
  cost = 10,
  tolerance = 0.001,
  epsilon = 0.1,
  cross = 10L,
  ...
)

```

Arguments

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

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_ndvi, ml_method = sits_svm)
  # classify the point
  point_ndvi <- sits_select(point_mt_6bands, bands = "NDVI")
  # classify the point
  point_class <- sits_classify(
    data = point_ndvi, ml_model = 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.

TAE is a simplified version of the well-known self-attention architecture used in large language models. Its modified self-attention scheme that uses the input embeddings as values. TAE defines a single master query for each sequence, computed from the temporal average of the queries. This master query is compared to the sequence of keys to produce a single attention mask used to weight the temporal mean of values into a single feature vector.

Usage

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

Arguments

<code>samples</code>	Time series with the training samples.
<code>samples_validation</code>	Time series with the validation samples. if the <code>samples_validation</code> parameter is provided, the <code>validation_split</code> parameter is ignored.
<code>epochs</code>	Number of iterations to train the model.
<code>batch_size</code>	Number of samples per gradient update.
<code>validation_split</code>	Number between 0 and 1. Fraction of training data to be used as validation data.
<code>optimizer</code>	Optimizer function to be used.
<code>opt_hparams</code>	Hyperparameters for optimizer: <code>lr</code> : Learning rate of the optimizer <code>eps</code> : Term added to the denominator to improve numerical stability. <code>weight_decay</code> : 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.
seed	Seed for random values.
verbose	Verbosity mode (TRUE/FALSE). Default is FALSE.

Value

A fitted model to be used for classification.

Note

sits provides a set of default values for all classification models. These settings have been chosen based on testing by the authors. Nevertheless, users can control all parameters for each model. Novice users can rely on the default values, while experienced ones can fine-tune deep learning models using [sits_tuning](#).

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.

Author(s)

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

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

Rolf Simoes, <rolfsimoes@gmail.com>

Felipe Souza, <lipecaso@gmail.com>

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()) {
  # create a TAE model
  torch_model <- sits_train(samples_modis_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 = "MOD13Q1-6.1",
  data_dir = data_dir
)
# classify a data cube
probs_cube <- sits_classify(
  data = cube, ml_model = torch_model, output_dir = tempdir()
)
# plot the probability cube
plot(probs_cube)
# smooth the probability cube using Bayesian statistics
bayes_cube <- sits_smooth(probs_cube, output_dir = tempdir())
# plot the smoothed cube
plot(bayes_cube)
# label the probability cube
label_cube <- sits_label_classification(
  bayes_cube,
  output_dir = tempdir()
)
# 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.

Usage

```

sits_tempcnn(
  samples = NULL,
  samples_validation = NULL,
  cnn_layers = c(64L, 64L, 64L),
  cnn_kernels = c(3L, 3L, 3L),
  cnn_dropout_rates = c(0.2, 0.2, 0.2),
  dense_layer_nodes = 256L,
  dense_layer_dropout_rate = 0.5,
  epochs = 150L,
  batch_size = 64L,
  validation_split = 0.2,
  optimizer = torch::optim_adamw,

```

```

    opt_hparams = list(lr = 5e-04, eps = 1e-08, weight_decay = 1e-06),
    lr_decay_epochs = 1L,
    lr_decay_rate = 0.95,
    patience = 20L,
    min_delta = 0.01,
    seed = NULL,
    verbose = FALSE
)

```

Arguments

<code>samples</code>	Time series with the training samples.
<code>samples_validation</code>	Time series with the validation samples. if the <code>samples_validation</code> parameter is provided, the <code>validation_split</code> parameter is ignored.
<code>cnn_layers</code>	Number of 1D convolutional filters per layer
<code>cnn_kernels</code>	Size of the 1D convolutional kernels.
<code>cnn_dropout_rates</code>	Dropout rates for 1D convolutional filters.
<code>dense_layer_nodes</code>	Number of nodes in the dense layer.
<code>dense_layer_dropout_rate</code>	Dropout rate (0,1) for the dense layer.
<code>epochs</code>	Number of iterations to train the model.
<code>batch_size</code>	Number of samples per gradient update.
<code>validation_split</code>	Fraction of training data to be used for validation.
<code>optimizer</code>	Optimizer function to be used.
<code>opt_hparams</code>	Hyperparameters for optimizer: <code>lr</code> : Learning rate of the optimizer <code>eps</code> : Term added to the denominator to improve numerical stability. <code>weight_decay</code> : L2 regularization
<code>lr_decay_epochs</code>	Number of epochs to reduce learning rate.
<code>lr_decay_rate</code>	Decay factor for reducing learning rate.
<code>patience</code>	Number of epochs without improvements until training stops.
<code>min_delta</code>	Minimum improvement in loss function to reset the patience counter.
<code>seed</code>	Seed for random values.
<code>verbose</code>	Verbosity mode (TRUE/FALSE). Default is FALSE.

Value

A fitted model to be used for classification.

Note

sits provides a set of default values for all classification models. These settings have been chosen based on testing by the authors. Nevertheless, users can control all parameters for each model. Novice users can rely on the default values, while experienced ones can fine-tune deep learning models using [sits_tuning](#).

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.

Author(s)

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

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

Rolf Simoes, <rolfsimoes@gmail.com>

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()) {
  # create a TempCNN model
  torch_model <- sits_train(
    samples_modis_ndvi,
    sits_tempcnn(epochs = 20, verbose = TRUE)
  )
  # 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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # classify a data cube
  probs_cube <- sits_classify(
    data = cube, ml_model = torch_model, output_dir = tempdir()
  )
  # plot the probability cube
  plot(probs_cube)
  # smooth the probability cube using Bayesian statistics
  bayes_cube <- sits_smooth(probs_cube, output_dir = tempdir())
}
```

```

    # plot the smoothed cube
    plot(bayes_cube)
    # label the probability cube
    label_cube <- sits_label_classification(
      bayes_cube,
      output_dir = tempdir()
    )
    # plot the labelled cube
    plot(label_cube)
  }

```

sits_texture

Apply a set of texture measures on a data cube.

Description

A set of texture measures based on the Grey Level Co-occurrence Matrix (GLCM) described by Haralick. Our implementation follows the guidelines and equations described by Hall-Beyer (both are referenced below).

Usage

```

sits_texture(cube, ...)

## S3 method for class 'raster_cube'
sits_texture(
  cube,
  ...,
  window_size = 3L,
  angles = 0,
  memsize = 4L,
  multicores = 2L,
  output_dir,
  progress = TRUE
)

## S3 method for class 'derived_cube'
sits_texture(cube, ...)

## Default S3 method:
sits_texture(cube, ...)

```

Arguments

cube	Valid sits cube
...	GLCM function (see details).
window_size	An odd number representing the size of the sliding window.

angles	The direction angles in radians related to the central pixel and its neighbor (See details). Default is 0.
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

The spatial relation between the central pixel and its neighbor is expressed in radians values, where: #'

- 0: corresponds to the neighbor on right-side
- $\pi/4$: corresponds to the neighbor on the top-right diagonals
- $\pi/2$: corresponds to the neighbor on above
- $3\pi/4$: corresponds to the neighbor on the top-left diagonals

Our implementation relies on a symmetric co-occurrence matrix, which considers the opposite directions of an angle. For example, the neighbor pixels based on 0 angle rely on the left and right direction; the neighbor pixels of $\pi/2$ are above and below the central pixel, and so on. If more than one angle is provided, we compute their average.

Value

A sits cube with new bands, produced according to the requested measure.

Available texture functions

- `glcm_contrast()`: measures the contrast or the amount of local variations present in an image. Low contrast values indicate regions with low spatial frequency.
- `glcm_homogeneity()`: also known as the Inverse Difference Moment, it measures image homogeneity by assuming larger values for smaller gray tone differences in pair elements.
- `glcm_asm()`: the Angular Second Moment (ASM) measures textural uniformity. High ASM values indicate a constant or a periodic form in the window values.
- `glcm_energy()`: measures textural uniformity. Energy is defined as the square root of the ASM.
- `glcm_mean()`: measures the mean of the probability of co-occurrence of specific pixel values within the neighborhood.
- `glcm_variance()`: measures the heterogeneity and is strongly correlated to first order statistical variables such as standard deviation. Variance values increase as the gray-level values deviate from their mean.
- `glcm_std()`: measures the heterogeneity and is strongly correlated to first order statistical variables such as standard deviation. STD is defined as the square root of the variance.
- `glcm_correlation()`: measures the gray-tone linear dependencies of the image. Low correlation values indicate homogeneous region edges.

Author(s)

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

Felipe Carlos, <efelipecarlos@gmail.com>

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

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

References

Robert M. Haralick, K. Shanmugam, Its'Hak Dinstein, "Textural Features for Image Classification", IEEE Transactions on Systems, Man, and Cybernetics, SMC-3, 6, 610-621, 1973, doi:10.1109/TSMC.1973.4309314.

Hall-Beyer, M., "GLCM Texture Tutorial", 2007, doi:10.13140/RG.2.2.12424.21767.

Hall-Beyer, M., "Practical guidelines for choosing GLCM textures to use in landscape classification tasks over a range of moderate spatial scales", International Journal of Remote Sensing, 38, 1312–1338, 2017, doi:10.1080/01431161.2016.1278314.

A. Baraldi and F. Pannigiani, "An investigation of the textural characteristics associated with gray level co-occurrence matrix statistical parameters," IEEE Transactions on Geoscience and Remote Sensing, 33, 2, 293-304, 1995, doi:10.1109/TGRS.1995.8746010.

Shokr, M. E., "Evaluation of second-order texture parameters for sea ice classification from radar images", J. Geophys. Res., 96, 10625–10640, 1991, doi:10.1029/91JC00693.

Peng Gong, Danielle J. Marceau, Philip J. Howarth, "A comparison of spatial feature extraction algorithms for land-use classification with SPOT HRV data", Remote Sensing of Environment, 40, 2, 1992, 137-151, doi:10.1016/00344257(92)900118.

Examples

```
if (sits_run_examples()) {
  data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
  cube <- sits_cube(
    source = "BDC",
    collection = "MOD13Q1-6.1",
    data_dir = data_dir
  )

  # Compute the NDVI variance
  cube_texture <- sits_texture(
    cube = cube,
    NDVIVAR = glcm_variance(NDVI),
    window_size = 5,
    output_dir = tempdir()
  )
}
```

sits_tiles_to_roi	<i>Convert MGRS tile information to ROI in WGS84</i>
-------------------	--

Description

Takes a list of MGRS tiles and produces a ROI covering them

Usage

```
sits_tiles_to_roi(tiles, grid_system = "MGRS")
```

Arguments

tiles	Character vector with names of MGRS tiles
grid_system	Grid system to be used

Value

roi Valid ROI to use in other SITS functions

Author(s)

Gilberto Camara, <gilberto.camara@inpe.br>
Rolf Simoes, <rolf.simoes@gmail.com>

sits_timeline	<i>Get timeline of a cube or a set of time series</i>
---------------	---

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)

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

## S3 method for class 'sits_model'
sits_timeline(data)

## S3 method for class 'raster_cube'
sits_timeline(data)
```

```
## S3 method for class 'derived_cube'
sits_timeline(data)

## S3 method for class 'tbl_df'
sits_timeline(data)

## Default S3 method:
sits_timeline(data)
```

Arguments

`data` Tibble of class "sits" or class "raster_cube"

Value

Vector of class Date with timeline of samples or data cube.

Author(s)

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

Examples

```
sits_timeline(samples_modis_ndvi)
```

```
sits_timeseries_to_csv
```

Export a a full sits tibble to the CSV format

Description

Converts metadata and data 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"), plus the all the time series for each data

Usage

```
sits_timeseries_to_csv(data, file = NULL)
```

Arguments

`data` Time series (tibble of class "sits").

`file` Full path of the exported CSV file (valid file name with extension ".csv").

Value

Return data.frame with CSV columns (optional)

Author(s)

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

Examples

```
csv_ts <- sits_timeseries_to_csv(cerrado_2classes)
csv_file <- paste0(tempdir(), "/cerrado_2classes_ts.csv")
sits_timeseries_to_csv(cerrado_2classes, file = csv_file)
```

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"). If the file is NULL, returns a data.frame as an object

Usage

```
sits_to_csv(data, file = NULL)

## S3 method for class 'sits'
sits_to_csv(data, file = NULL)

## S3 method for class 'tbl_df'
sits_to_csv(data, file)

## Default S3 method:
sits_to_csv(data, file)
```

Arguments

data	Time series (tibble of class "sits").
file	Full path of the exported CSV file (valid file name with extension ".csv").

Value

Return data.frame with CSV columns (optional)

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	<i>Save accuracy assessments as Excel files</i>
--------------	---

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, file)

## S3 method for class 'sits_accuracy'
sits_to_xlsx(acc, file)

## S3 method for class 'list'
sits_to_xlsx(acc, file)
```

Arguments

acc	Accuracy statistics (either an output of <code>sits_accuracy</code> or a list of those)
file	The file where the XLSX data is to be saved.

Value

No return value, called for side effects.

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_ndvi,
    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 = tempfile("accuracy_mato_grosso_dl_", fileext = ".xlsx")
    )
  }

```

sits_train

Train classification models

Description

Given a tibble with a set of time series, returns trained models. Currently, sits supports the following models:

- support vector machines: [sits_svm](#);
- random forests: [sits_rfor](#);
- extreme gradient boosting: [sits_xgboost](#);
- light gradient boosting: [sits_lightgbm](#);
- multi-layer perceptrons: [sits_mlp](#);
- temporal CNN: [sits_tempcnn](#);
- residual network encoders: [sits_resnet](#);
- LSTM with convolutional networks: [sits_lstm_fcn](#);
- temporal self-attention encoders: [sits_lighttae](#) and [sits_tae](#).

Usage

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

Arguments

<code>samples</code>	Time series with the training samples.
<code>ml_method</code>	Machine learning method.

Value

Model fitted to input data to be passed to [sits_classify](#)

Note

The main sits classification workflow has the following steps:

1. `sits_cube`: selects a ARD image collection from a cloud provider.
2. `sits_cube_copy`: copies an ARD image collection from a cloud provider to a local directory for faster processing.
3. `sits_regularize`: create a regular data cube from an ARD image collection.
4. `sits_apply`: create new indices by combining bands of a regular data cube (optional).
5. `sits_get_data`: extract time series from a regular data cube based on user-provided labelled samples.
6. `sits_train`: train a machine learning model based on image time series.
7. `sits_classify`: classify a data cube using a machine learning model and obtain a probability cube.
8. `sits_smooth`: post-process a probability cube using a spatial smoother to remove outliers and increase spatial consistency.
9. `sits_label_classification`: produce a classified map by selecting the label with the highest probability from a smoothed cube.

`sits_train` provides a standard interface to machine learning models. It takes two mandatory parameters: the training data (`samples`) and the ML algorithm (`ml_method`). The output is a model that can be used to classify individual time series or data cubes with `sits_classify`.

`sits` provides a set of default values for all classification models. These settings have been chosen based on testing by the authors. Nevertheless, users can control all parameters for each model. Novice users can rely on the default values, while experienced ones can fine-tune deep learning models using `sits_tuning`.

Author(s)

Rolf Simoes, <rolfsimoes@gmail.com>

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

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

Examples

```
if (sits_run_examples()) {
  # Retrieve the set of samples for Mato Grosso
  # fit a training model (rfor model)
  ml_model <- sits_train(samples_modis_ndvi, 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(
    data = point_ndvi, ml_model = ml_model
  )
}
```


sits_tuning

*Tuning machine learning models hyper-parameters***Description**

This function performs a random search on values of selected hyperparameters, and produces a data frame with the accuracy and kappa values produced by a validation procedure. The result allows users to select appropriate hyperparameters for deep learning models.

Usage

```
sits_tuning(
  samples,
  samples_validation = NULL,
  validation_split = 0.2,
  ml_method = sits_tempcnn(),
  params = sits_tuning_hparams(optimizer = torch::optim_adamw, opt_hparams = list(lr =
    loguniform(0.01, 1e-04))),
  trials = 30L,
  multicores = 2L,
  gpu_memory = 4L,
  batch_size = 2L^gpu_memory,
  progress = FALSE
)
```

Arguments

<code>samples</code>	Time series set to be validated.
<code>samples_validation</code>	Time series set used for validation.
<code>validation_split</code>	Percent of original time series set to be used for validation (if <code>samples_validation</code> is NULL)
<code>ml_method</code>	Machine learning method.
<code>params</code>	List with hyper parameters to be passed to <code>ml_method</code> . User can use <code>uniform</code> , <code>choice</code> , <code>randint</code> , <code>normal</code> , <code>lognormal</code> , <code>loguniform</code> , and <code>beta</code> distribution functions to randomize parameters.
<code>trials</code>	Number of random trials to perform the search.
<code>multicores</code>	Number of cores to process in parallel.
<code>gpu_memory</code>	Memory available in GPU in GB (default = 4)
<code>batch_size</code>	Batch size for GPU classification.
<code>progress</code>	Show progress bar?

Value

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

Note

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. Instead of performing an exhaustive test of all parameter combinations, `sits_tuning` selects them randomly. Validation is done using an independent set of samples or by a validation split. The function returns the best hyperparameters in a list. Hyper-parameters passed to `params` parameter should be passed by calling `sits_tuning_hparams`.

When using a GPU for deep learning, `gpu_memory` indicates the memory of the graphics card which is available for processing. The parameter `batch_size` defines the size of the matrix (measured in number of rows) which is sent to the GPU for classification. Users can test different values of `batch_size` to find out which one best fits their GPU architecture.

It is not possible to have an exact idea of the size of Deep Learning models in GPU memory, as the complexity of the model and factors such as CUDA Context increase the size of the model in memory. Therefore, we recommend that you leave at least 1GB free on the video card to store the Deep Learning model that will be used.

For users of Apple M3 chips or similar with a Neural Engine, be aware that these chips share memory between the GPU and the CPU. Tests indicate that the `memsize` should be set to half to the total memory and the `batch_size` parameter should be a small number (we suggest the value of 64). Be aware that increasing these parameters may lead to memory conflicts.

Author(s)

Rolf Simoes, <rolfsimoes@gmail.com>

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_ndvi,
    ml_method = sits_tempcnn(),
    params = sits_tuning_hparams(
      optimizer = choice(
        torch::optim_adamw
      ),
      opt_hparams = list(
        lr = loguniform(10^-2, 10^-4)
      )
    ),
    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.

Users should pass the possible values for hyper-parameters as constants 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_ndvi,
    ml_method = sits_tempcnn(),
    params = sits_tuning_hparams(
      optimizer = choice(
        torch::optim_adamw,
        torch::optim_adagrad
      ),
      opt_hparams = list(
        lr = loguniform(10^-2, 10^-4),
        weight_decay = loguniform(10^-2, 10^-8)
      )
    ),
    trials = 20,
    multicores = 2,
    progress = FALSE
  )
}
```

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 and produces a uncertainty cube.

Usage

```
sits_uncertainty(cube, ...)

## S3 method for class 'probs_cube'
sits_uncertainty(
  cube,
  ...,
  type = "entropy",
  multicores = 2L,
  memsize = 4L,
  output_dir,
  version = "v1",
  progress = TRUE
)

## S3 method for class 'probs_vector_cube'
```

```

sits_uncertainty(
    cube,
    ...,
    type = "entropy",
    multicores = 2L,
    memsize = 4L,
    output_dir,
    version = "v1"
)

## S3 method for class 'raster_cube'
sits_uncertainty(cube, ...)

## Default S3 method:
sits_uncertainty(cube, ...)

```

Arguments

<code>cube</code>	Probability data cube.
<code>...</code>	Other parameters for specific functions.
<code>type</code>	Method to measure uncertainty. See details.
<code>multicores</code>	Number of cores to run the function.
<code>memsize</code>	Maximum overall memory (in GB) to run the function.
<code>output_dir</code>	Output directory for image files.
<code>version</code>	Version of resulting image (in the case of multiple tests).
<code>progress</code>	Check progress bar?

Value

An uncertainty data cube

Note

The output of `sits_classify` and `sits_smooth` is a probability cube containing the class probability for all pixels, which are generated by the machine learning model. The `sits_uncertainty` function takes a probability cube and produces a uncertainty code which contains a measure of uncertainty for each pixel, based on the class probabilities.

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:

1. `entropy`: the difference between all predictions expressed a Shannon measure of entropy.
2. `least`: the difference between 1.0 and most confident prediction.
3. `margin`: the difference between the two most confident predictions.

Author(s)

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

Rolf Simoes, <rolfsimoes@gmail.com>

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()) {
  # create a random forest model
  rfor_model <- sits_train(samples_modis_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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # classify a data cube
  probs_cube <- sits_classify(
    data = cube, ml_model = rfor_model, output_dir = tempdir()
  )
  # calculate uncertainty
  uncert_cube <- sits_uncertainty(probs_cube, output_dir = tempdir())
  # 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 = 100L,
  min_uncert = 0.4,
  sampling_window = 10L,
  multicores = 2L,
  memsize = 4L
)
```

Arguments

<code>uncert_cube</code>	An uncertainty cube. See sits_uncertainty .
<code>n</code>	Number of suggested points to be sampled per tile.
<code>min_uncert</code>	Minimum uncertainty value to select a sample.
<code>sampling_window</code>	Window size for collecting points (in pixels). The minimum window size is 10.
<code>multicores</code>	Number of workers for parallel processing (integer, min = 1, max = 2048).
<code>memsize</code>	Maximum overall memory (in GB) to run the function.

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, <rolfsimoes@gmail.com>
 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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
}
```

```

# build a random forest model
rfor_model <- sits_train(samples_modis_ndvi, ml_method = sits_rfor())
# classify the cube
probs_cube <- sits_classify(
  data = cube, ml_model = rfor_model, output_dir = tempdir()
)
# create an uncertainty cube
uncert_cube <- sits_uncertainty(probs_cube,
  type = "entropy",
  output_dir = tempdir()
)
# obtain a new set of samples for active learning
# the samples are located in uncertain places
new_samples <- sits_uncertainty_sampling(
  uncert_cube,
  n = 10, min_uncert = 0.4
)
}

```

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(),
  gpu_memory = 4L,
  batch_size = 2L^gpu_memory
)

```


Arguments

<code>samples</code>	Time series to be validated (class "sits").
<code>samples_validation</code>	Optional: Time series used for validation (class "sits")
<code>validation_split</code>	Percent of original time series set to be used for validation if <code>samples_validation</code> is NULL (numeric value).
<code>ml_method</code>	Machine learning method (function)
<code>gpu_memory</code>	Memory available in GPU in GB (default = 4)
<code>batch_size</code>	Batch size for GPU classification.

Value

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

Note

When using a GPU for deep learning, `gpu_memory` indicates the memory of the graphics card which is available for processing. The parameter `batch_size` defines the size of the matrix (measured in number of rows) which is sent to the GPU for classification. Users can test different values of `batch_size` to find out which one best fits their GPU architecture.

It is not possible to have an exact idea of the size of Deep Learning models in GPU memory, as the complexity of the model and factors such as CUDA Context increase the size of the model in memory. Therefore, we recommend that you leave at least 1GB free on the video card to store the Deep Learning model that will be used.

For users of Apple M3 chips or similar with a Neural Engine, be aware that these chips share memory between the GPU and the CPU. Tests indicate that the `memsizes` should be set to half to the total memory and the `batch_size` parameter should be a small number (we suggest the value of 64). Be aware that increasing these parameters may lead to memory conflicts.

Author(s)

Rolf Simoes, <rolfsimoes@gmail.com>

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

Examples

```
if (sits_run_examples()) {
  samples <- sits_sample(cerrado_2classes, frac = 0.5)
  samples_validation <- sits_sample(cerrado_2classes, frac = 0.5)
  conf_matrix_1 <- sits_validate(
    samples = samples,
    samples_validation = samples_validation,
    ml_method = sits_rfor()
  )
  conf_matrix_2 <- sits_validate(
    samples = cerrado_2classes,
```

```

        validation_split = 0.2,
        ml_method = sits_rfor()
    )
}

```

sits_variance

Calculate the variance of a probability cube

Description

Takes a probability cube and estimate the local variance of the logit of the probability, to support the choice of parameters for Bayesian smoothing.

Usage

```

sits_variance(cube, ...)

## S3 method for class 'probs_cube'
sits_variance(
  cube,
  ...,
  window_size = 9L,
  neigh_fraction = 0.5,
  memsize = 4L,
  multicores = 2L,
  output_dir,
  version = "v1",
  progress = TRUE
)

## S3 method for class 'raster_cube'
sits_variance(cube, ...)

## S3 method for class 'derived_cube'
sits_variance(cube, ...)

## Default S3 method:
sits_variance(cube, ...)

```

Arguments

cube	Probability data cube (class "probs_cube")
...	Parameters for specific functions
window_size	Size of the neighborhood (odd integer)
neigh_fraction	Fraction of neighbors with highest probability for Bayesian inference (numeric from 0.0 to 1.0)

memsize	Maximum overall memory (in GB) to run the smoothing (integer, min = 1, max = 16384)
multicores	Number of cores to run the smoothing function (integer, min = 1, max = 2048)
output_dir	Output directory for image files (character vector of length 1)
version	Version of resulting image (character vector of length 1)
progress	Check progress bar?

Value

A variance data cube.

Author(s)

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

Rolf Simoes, <rolfsimoes@gmail.com>

Examples

```
if (sits_run_examples()) {
  # create a random forest model
  rfor_model <- sits_train(samples_modis_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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # classify a data cube
  probs_cube <- sits_classify(
    data = cube, ml_model = rfor_model, output_dir = tempdir()
  )
  # plot the probability cube
  plot(probs_cube)
  # smooth the probability cube using Bayesian statistics
  var_cube <- sits_variance(probs_cube, output_dir = tempdir())
  # plot the variance cube
  plot(var_cube)
}
```

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 = "Set3", radius = 10L, add = FALSE)

## S3 method for class 'data.frame'
sits_view(x, ..., legend = NULL, palette = "Harmonic", add = FALSE)

## S3 method for class 'som_map'
sits_view(
  x,
  ...,
  id_neurons,
  legend = NULL,
  palette = "Harmonic",
  radius = 10L,
  add = FALSE
)

## S3 method for class 'raster_cube'
sits_view(
  x,
  ...,
  band = NULL,
  red = NULL,
  green = NULL,
  blue = NULL,
  tiles = x[["tile"]][[1L]],
  dates = NULL,
  palette = "RdYlGn",
  rev = FALSE,
  opacity = 0.85,
  max_cog_size = 2048L,
  first_quantile = 0.02,
  last_quantile = 0.98,
  leaflet_megabytes = 64L,
  add = FALSE
)

## S3 method for class 'uncertainty_cube'
sits_view(
  x,
  ...,
  tiles = x[["tile"]][[1L]],
  legend = NULL,
  palette = "RdYlGn",
  rev = FALSE,

```

```

        opacity = 0.85,
        max_cog_size = 2048L,
        first_quantile = 0.02,
        last_quantile = 0.98,
        leaflet_megabytes = 64L,
        add = FALSE
    )

## S3 method for class 'class_cube'
sits_view(
    x,
    ...,
    tiles = x[["tile"]],
    legend = NULL,
    palette = "Set3",
    version = NULL,
    opacity = 0.85,
    max_cog_size = 2048L,
    leaflet_megabytes = 32L,
    add = FALSE
)

## S3 method for class 'probs_cube'
sits_view(
    x,
    ...,
    tiles = x[["tile"]][[1L]],
    label = x[["labels"]][[1L]][[1L]],
    legend = NULL,
    palette = "YlGn",
    rev = FALSE,
    opacity = 0.85,
    max_cog_size = 2048L,
    first_quantile = 0.02,
    last_quantile = 0.98,
    leaflet_megabytes = 64L,
    add = FALSE
)

## S3 method for class 'vector_cube'
sits_view(
    x,
    ...,
    tiles = x[["tile"]][[1L]],
    seg_color = "yellow",
    line_width = 0.5,
    add = FALSE
)

```

```
## S3 method for class 'class_vector_cube'
sits_view(
  x,
  ...,
  tiles = x[["tile"]][[1L]],
  seg_color = "yellow",
  line_width = 0.2,
  version = NULL,
  legend = NULL,
  palette = "Set3",
  opacity = 0.85,
  add = FALSE
)

## Default S3 method:
sits_view(x, ...)
```

Arguments

x	Object of class "sits", "data.frame", "som_map", "raster_cube", "probs_cube", "vector_cube", or "class cube".
...	Further specifications for sits_view .
legend	Named vector that associates labels to colors.
palette	Color palette from RColorBrewer
radius	Radius of circle markers
add	Add image to current leaflet
id_neurons	Neurons from the SOM map to be shown.
band	Single band for viewing false color 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.
rev	Revert color palette?
opacity	Opacity of segment fill or class cube
max_cog_size	Maximum size of COG overviews (lines or columns)
first_quantile	First quantile for stretching images
last_quantile	Last quantile for stretching images
leaflet_megabytes	Maximum size for leaflet (in MB)
version	Version name (to compare different classifications)
label	Label to be plotted (in case of probs cube)
seg_color	Color for segment boundaries
line_width	Line width for segments (in pixels)

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

To show a false color image, use "band" to chose one of the bands, "tiles" to select tiles, "first_quantile" and "last_quantile" to set the cutoff points. Choose only one date in the "dates" parameter. The color scheme is defined by either "palette" (use an available color scheme) or legend (user-defined color scheme). To see which palettes are pre-defined, use `cols4all : g4a_gui` or select any ColorBrewer name. The "rev" parameter reverts the order of colors in the palette.

To show an RGB composite, select "red", "green" and "blue" bands, "tiles", "dates", "opacity", "first_quantile" and "last_quantile". One can also get an RGB composite, by selecting one band and three dates. In this case, the first date will be shown in red, the second in green and third in blue.

Probability cubes are shown in false color. The parameter "labels" controls which labels are shown. If left blank, only the first map is shown. For color control, use "palette", "legend", and "rev" (as described above).

Vector cubes have both a vector and a raster component. The vector part are the segments produced by [sits_segment](#). Their visual output is controlled by "seg_color" and "line_width" parameters. The raster output works in the same way as the false color and RGB views described above.

Classified cubes need information on how to render each class. There are three options: (a) the classes are part of an existing color scheme; (b) the user provides a legend which associates each class to a color; (c) use a generic palette (such as "Spectral") and allocate colors based on this palette. To find out how to create a customized color scheme, read the chapter "Data Visualisation in sits" in the sits book.

To compare different classifications, use the "version" parameter to distinguish between the different maps that are shown.

Vector classified cubes are displayed as classified cubes, with the segments overlaid on top of the class map, controlled by "seg_color" and "line_width".

Samples are shown on the map based on their geographical locations and on the color of their classes assigned in their color scheme. Users can also assign a legend or a palette to choose colors. See information above on the display of classified cubes.

For all types of data cubes, the following parameters apply:

- `opacity`: controls the transparency of the map.
- `max_cog_size`: For COG data, controls the level of aggregation to be used for display, measured in pixels, e.g., a value of 512 will select a 512 x 512 aggregated image. Small values are faster to show, at a loss of visual quality.
- `leaflet_megabytes`: maximum size of leaflet to be shown associated to the map (in megabytes). Bigger values use more memory.
- `add`: controls whether a new visualisation will be overlaid on top of an existing one. Default is FALSE.

Author(s)

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

Examples

```

if (sits_run_examples()) {
  # view samples
  sits_view(cerrado_2classes)
  # create a local data cube
  data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
  modis_cube <- sits_cube(
    source = "BDC",
    collection = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # view the data cube
  sits_view(modis_cube,
    band = "NDVI"
  )
  # train a model
  rf_model <- sits_train(samples_modis_ndvi, sits_rfor())
  # classify the cube
  modis_probs <- sits_classify(
    data = modis_cube,
    ml_model = rf_model,
    output_dir = tempdir()
  )
  # generate a map
  modis_label <- sits_label_classification(
    modis_probs,
    output_dir = tempdir()
  )
  # view the classified map
  sits_view(modis_label)
  # view the classified map with the B/W image
  sits_view(modis_cube,
    band = "NDVI",
    class_cube = modis_label,
    dates = sits_timeline(modis_cube)[[1]]
  )
  # view the classified map with the RGB image
  sits_view(modis_cube,
    red = "NDVI", green = "NDVI", blue = "NDVI",
    class_cube = modis_label,
    dates = sits_timeline(modis_cube)[[1]]
  )
  # create an uncertainty cube
  modis_uncert <- sits_uncertainty(
    cube = modis_probs,
    output_dir = tempdir()
  )
  # view the uncertainty cube
  sits_view(modis_uncert, rev = TRUE)
}

```

sits_whittaker	<i>Filter time series with whittaker filter</i>
----------------	---

Description

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.

Usage

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

Arguments

data	Time series or matrix.
lambda	Smoothing factor to be applied (default 0.5).

Value

Filtered time series

Author(s)

Rolf Simoes, <rolfsimoes@gmail.com>
 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.

See Also

[sits_apply](#)

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_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 = 5L,
  min_child_weight = 1,
  max_delta_step = 1,
  subsample = 0.85,
  nfold = 5L,
  nrounds = 100L,
  nthread = 6L,
  early_stopping_rounds = 20L,
  verbose = FALSE
)

```

Arguments

<code>samples</code>	Time series with the training samples.
<code>learning_rate</code>	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
<code>min_split_loss</code>	Minimum loss reduction to make a further partition of a leaf. Default: 1.
<code>max_depth</code>	Maximum depth of a tree. Increasing this value makes the model more complex and more likely to overfit. Default: 5.
<code>min_child_weight</code>	If the leaf node has a minimum sum of instance weights lower than <code>min_child_weight</code> , tree splitting stops. The larger <code>min_child_weight</code> 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)
nthread	Number of threads (default = 6)
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)

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_ndvi, ml_method = sits_xgboost)
  # classify the point
  point_ndvi <- sits_select(point_mt_6bands, bands = "NDVI")
  # classify the point
  point_class <- sits_classify(
    data = point_ndvi, ml_model = ml_model
  )
  plot(point_class)
}
```

summary.class_cube	<i>Summarize data cubes</i>
--------------------	-----------------------------

Description

This is a generic function. Parameters depend on the specific type of input.

Usage

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

Arguments

object	Object of class "class_cube"
...	Further specifications for summary .

Value

A summary of a classified cube

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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # create a random forest model
  rfor_model <- sits_train(samples_modis_ndvi, sits_rfor())
  # classify a data cube
  probs_cube <- sits_classify(
    data = cube, ml_model = rfor_model, output_dir = tempdir()
  )
  # label the probability cube
  label_cube <- sits_label_classification(
    probs_cube,
    output_dir = tempdir()
  )
  summary(label_cube)
}
```

summary.raster_cube	<i>Summarize data cubes</i>
---------------------	-----------------------------

Description

This is a generic function. Parameters depend on the specific type of input.

Usage

```
## S3 method for class 'raster_cube'
summary(object, ..., tile = NULL, date = NULL)
```

Arguments

object	Object of classes "raster_cube".
...	Further specifications for summary .
tile	Tile to be summarized
date	Date to be summarized

Value

A summary of the data cube.

Author(s)

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

Felipe Souza, <felipe.souza@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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  summary(cube)
}
```

summary.sits	<i>Summarize sits</i>
--------------	-----------------------

Description

This is a generic function. Parameters depend on the specific type of input.

Usage

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

Arguments

object	Object of class "sits".
...	Further specifications for summary .

Value

A summary of the sits tibble.

Author(s)

Gilberto Camara, <gilberto.camara@inpe.br>
 Felipe Carvalho, <felipe.carvalho@inpe.br>

Examples

```
if (sits_run_examples()) {
  summary(samples_modis_ndvi)
}
```

summary.sits_accuracy	<i>Summarize accuracy matrix for training data</i>
-----------------------	--

Description

This is a generic function. Parameters depend on the specific type of input.

Usage

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

Arguments

object Object of class "sits_accuracy".
 ... Further specifications for [summary](#).

Value

A summary of the sample accuracy

Author(s)

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

Examples

```
if (sits_run_examples()) {
  data(cerrado_2classes)
  # split training and test data
  train_data <- sits_sample(cerrado_2classes, frac = 0.5)
  test_data <- sits_sample(cerrado_2classes, frac = 0.5)
  # train a random forest model
  rfor_model <- sits_train(train_data, sits_rfor())
  # classify test data
  points_class <- sits_classify(
    data = test_data,
    ml_model = rfor_model
  )
  # measure accuracy
  acc <- sits_accuracy(points_class)
  summary(acc)
}
```

summary.sits_area_accuracy

Summarize accuracy matrix for area data

Description

This is a generic function. Parameters depend on the specific type of input.

Usage

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

Arguments

object Object of classe "sits_accuracy".
 ... Further specifications for [summary](#).

Value

A summary of the sample accuracy

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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # create a random forest model
  rfor_model <- sits_train(samples_modis_ndvi, sits_rfor())
  # classify a data cube
  probs_cube <- sits_classify(
    data = cube, ml_model = rfor_model, output_dir = tempdir()
  )
  # label the probability cube
  label_cube <- sits_label_classification(
    probs_cube,
    output_dir = tempdir()
  )
  # 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 = ground_truth)
  summary(as)
}
```

summary.variance_cube *Summarize variance cubes*

Description

This is a generic function. Parameters depend on the specific type of input.

Usage

```
## S3 method for class 'variance_cube'
summary(
  object,
  ...,
  intervals = 0.05,
  sample_size = 10000L,
  multicores = 2L,
  memsize = 2L,
  quantiles = c("75%", "80%", "85%", "90%", "95%", "100%")
)
```

Arguments

object	Object of class "class_cube"
...	Further specifications for summary .
intervals	Intervals to calculate the quantiles
sample_size	The approximate size of samples will be extracted from the variance cube (by tile).
multicores	Number of cores to summarize data (integer, min = 1, max = 2048).
memsize	Memory in GB available to summarize data (integer, min = 1, max = 16384).
quantiles	Quantiles to be shown

Value

A summary of a variance cube

Author(s)

Gilberto Camara, <gilberto.camara@inpe.br>
 Felipe Carlos, <efelipecarlos@gmail.com>
 Felipe Souza, <lipecaso@gmail.com>

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 = "MOD13Q1-6.1",
    data_dir = data_dir
  )
  # create a random forest model
  rfor_model <- sits_train(samples_modis_ndvi, sits_rfor())
  # classify a data cube
  probs_cube <- sits_classify(
```

```
      data = cube, ml_model = rfor_model, output_dir = tempdir()
    )
    variance_cube <- sits_variance(
      data = probs_cube,
      output_dir = tempdir()
    )
    summary(variance_cube)
  }
```

Index

* datasets

- cerrado_2classes, 9
- point_mt_6bands, 46
- samples_l8_rondonia_2bands, 46
- samples_modis_ndvi, 47

cerrado_2classes, 9

hist, 11, 12

hist.probs_cube, 9

hist.raster_cube, 10

hist.sits, 11

hist.uncertainty_cube, 12

impute_linear, 13, 64, 67, 69

kohonen::supersom, 189

plot, 14, 14, 15, 17, 19–23, 25, 27, 29, 30, 32–38, 40, 42, 43, 45, 176

plot.class_cube, 14, 15

plot.class_vector_cube, 14, 17

plot.dem_cube, 14, 18

plot.geo_distances, 14, 20

plot.patterns, 14, 21

plot.predicted, 14, 22

plot.probs_cube, 14, 23

plot.probs_vector_cube, 25

plot.raster_cube, 14, 26

plot.rfor_model, 14, 29

plot.sar_cube, 14, 30

plot.sits, 14

plot.sits_accuracy, 32

plot.sits_cluster, 14, 33

plot.som_clean_samples, 34

plot.som_evaluate_cluster, 14, 35

plot.som_map, 14, 36

plot.torch_model, 14, 37

plot.uncertainty_cube, 14, 38

plot.uncertainty_vector_cube, 14, 39

plot.variance_cube, 41

plot.vector_cube, 14, 43

plot.xgb_model, 14, 45

point_mt_6bands, 46

samples_l8_rondonia_2bands, 46

samples_modis_ndvi, 47

sits (sits-package), 7

sits-package, 7

sits_accuracy, 47, 206

sits_add_base_cube, 50

sits_apply, 7, 51, 53, 62, 86, 102, 111, 131, 164, 184, 208, 225

sits_as_sf, 54

sits_as_stars, 55

sits_as_terra, 56

sits_bands, 58

sits_bands<- (sits_bands), 58

sits_bbox, 59

sits_classify, 7, 47, 53, 61, 61, 62, 82, 86, 91, 99, 102, 111, 131, 133, 164, 169, 176, 184, 194, 207, 208, 213, 227

sits_classify.raster_cube, 61, 63

sits_classify.segs_cube, 65

sits_classify.sits, 61, 68, 68

sits_classify.vector_cube, 61, 99

sits_classify.vector_cube (sits_classify.segs_cube), 65

sits_clean, 70

sits_cluster_clean, 72

sits_cluster_clean(), 73

sits_cluster_dendro, 73

sits_cluster_dendro(), 72, 73, 75

sits_cluster_frequency, 74

sits_cluster_frequency(), 73

sits_colors, 75

sits_colors_qgis, 76

sits_colors_reset, 77

sits_colors_set, 75, 77

sits_colors_show, 75, 79

sits_combine_predictions, 79

- sits_confidence_sampling, 82
- sits_config, 83
- sits_config_show, 84
- sits_config_user_file, 85
- sits_cube, 7, 52, 62, 85, 86, 102, 111, 131, 164, 176, 182, 184, 208
- sits_cube.local_cube, 85, 88
- sits_cube.results_cube, 85, 90
- sits_cube.stac_cube, 85, 94
- sits_cube.vector_cube, 85, 98
- sits_cube_copy, 7, 52, 62, 86, 101, 102, 111, 131, 164, 184, 208
- sits_factory_function, 103
- sits_filter, 105
- sits_formula_linear, 105
- sits_formula_logref, 106
- sits_geo_dist, 107
- sits_get_class, 109
- sits_get_data, 7, 53, 62, 86, 102, 110, 111, 131, 164, 184, 208
- sits_get_data.csv, 111, 113
- sits_get_data.data.frame, 111, 114
- sits_get_data.sf, 111, 115
- sits_get_data.shp, 111, 117
- sits_get_data.sits, 111, 119
- sits_get_probs, 120
- sits_impute, 122
- sits_kfold_validate, 123
- sits_label_classification, 7, 47, 53, 62, 86, 91, 99, 102, 112, 129, 131, 164, 176, 184, 208
- sits_labels, 124, 126–128
- sits_labels<-, 126
- sits_labels<->.class_cube, 127
- sits_labels<->.default, 127
- sits_labels<->.probs_cube, 128
- sits_labels<->.sits, 128
- sits_labels_summary, 129
- sits_lightgbm, 62, 132, 207
- sits_lighttae, 62, 133, 207
- sits_list_collections, 86, 88, 91, 94, 95, 99, 136
- sits_lstm_fcn, 62, 137, 207
- sits_merge, 138
- sits_mgrs_to_roi, 140
- sits_mixture_model, 140
- sits_mlp, 62, 143, 207
- sits_model_export, 145
- sits_mosaic, 146
- sits_patterns, 148
- sits_pred_features, 151
- sits_pred_normalize, 152
- sits_pred_references, 153
- sits_pred_sample, 153
- sits_predictors, 150
- sits_reclassify, 154
- sits_reduce, 157
- sits_reduce_imbalance, 159
- sits_regularize, 7, 52, 62, 86, 102, 111, 131, 161, 164, 176, 184, 208
- sits_resnet, 62, 166, 207
- sits_rfor, 62, 168, 207
- sits_roi_to_mgrs, 170
- sits_roi_to_tiles, 171
- sits_run_examples, 172
- sits_run_tests, 172
- sits_sample, 173
- sits_sampling_design, 174
- sits_segment, 25, 43, 61, 65, 67, 99, 175, 176, 223
- sits_select, 178
- sits_sgolay, 64, 67, 69, 105, 179
- sits_slic, 176, 180
- sits_smooth, 7, 53, 62, 71, 82, 86, 91, 102, 112, 131, 164, 182, 184, 208, 213
- sits_som_clean_samples, 185, 189
- sits_som_evaluate_cluster, 187, 189
- sits_som_map, 160, 185, 188, 189
- sits_som_remove_samples, 190
- sits_stats, 191
- sits_stratified_sampling, 191
- sits_svm, 62, 193, 207
- sits_tae, 62, 133, 195, 207
- sits_tempcnn, 62, 197, 207
- sits_texture, 200
- sits_tiles_to_roi, 203
- sits_timeline, 203
- sits_timeseries_to_csv, 204
- sits_to_csv, 205
- sits_to_xlsx, 206
- sits_train, 7, 53, 61–63, 66, 69, 86, 102, 111, 131, 164, 184, 207, 208
- sits_tuning, 135, 144, 196, 199, 208, 209
- sits_tuning_hparams, 210, 211
- sits_uncertainty, 91, 212, 215
- sits_uncertainty.probs_vector_cube, 99

sits_uncertainty_sampling, [214](#)
sits_validate, [216](#)
sits_variance, [218](#)
sits_view, [176](#), [219](#), [222](#)
sits_whittaker, [64](#), [67](#), [69](#), [105](#), [225](#)
sits_xgboost, [62](#), [207](#), [226](#)
summary, [9](#), [10](#), [228–231](#), [233](#)
summary.class_cube, [228](#)
summary.raster_cube, [229](#)
summary.sits, [230](#)
summary.sits_accuracy, [230](#)
summary.sits_area_accuracy, [231](#)
summary.variance_cube, [232](#)