SITS: Data Analysis and Machine Learning for Data Cubes using Satellite Image Time Series

Rolf Simoes *National Institute for Space Research (INPE), Brazil*
Gilberto Camara *National Institute for Space Research (INPE), Brazil*
Pedro R. Andrade *National Institute for Space Research (INPE), Brazil*
Alexandre Carvalho *Institute for Applied Economics Research (IPEA), Brazil*
Lorena Santos *National Institute for Space Research (INPE), Brazil*
Karine Ferreira *National Institute for Space Research (INPE), Brazil*
Victor Maus *University of Vienna, Austria*
Gilberto Queiroz *National Institute for Space Research (INPE), Brazil*

Using time series derived from big Earth Observation data sets is one of the leading research trends in Land Use Science and Remote Sensing. One of the more promising uses of satellite time series is its application for classification of land use and land cover, since our growing demand for natural resources has caused major environmental impacts. Here, we present an open source *R* package for satellite image time series analysis called `sits`. Package `sits` provides support on how to use statistical learning techniques with image time series obtained from data cubes. These methods include linear and quadratic discrimination analysis, support vector machines, random forests, boosting, deep learning and convolution neural networks.

Introduction

Earth observation satellites provide a regular and consistent set of information about the land and oceans of the planet. Recently, most space agencies have adopted open data policies, making unprecedented amounts of satellite data available for research and operational use. This data deluge has brought about a major challenge: *How to design and build technologies that allow the Earth observation community to analyse big data sets?*

The approach taken in the current work is to develop data analysis methods that work with satellite image time series, obtained by taking calibrated and comparable measures of the same location in Earth at different times. These measures can be obtained by a single sensor (e.g., MODIS) or by combining different sensors (e.g., Landsat 8 and Sentinel-2). If obtained by frequent revisits, the temporal resolution of these data sets can capture important land use changes.

Time series of remote sensing data show that land cover can occur not only in a progressive and gradual way, but they may also show discontinuities with abrupt changes [Lambin et al., 2003]. Analyses of multiyear time series of land surface attributes, their fine-scale spatial pattern, and their seasonal evolution leads to a broader view of land-cover change. Satellite image time series have already been used in applications such as mapping for detecting forest disturbance [Kennedy et al., 2010], ecology dynamics [Pasquarella et al., 2016], agricultural intensification [Galford et al., 2008], and its impacts on deforestation [Arvor et al., 2012]. Algorithms for processing image time series include BFAST for detecting breaks [Verbesselt et al., 2010], TIMESAT for modelling and measuring phenological attributes [Jönsson and Eklundh, 2004] and methods based on Dynamic Time Warping (DTW) for land use and land cover classification [Petitjean et al., 2012][Maus et al., 2016].

In this work, we present SITS, an open source R package for satellite image time series analysis. It provides support on how to use machine learning techniques with image time series. These methods include linear and quadratic discrimination analysis, support vector machines, random forests, and neural networks. One important contribution of the SITS package is to support the complete cycle of data analysis for time series classification, including data acquisition, visualisation, filtering, clustering, classification, validation and post-classification adjustments.

Most studies using satellite image time series for land cover classification use a *space-first, time-later* approach. For multiyear studies, researchers first derive best-fit yearly composites and then classify each composite image. For a review of these methods for land use and land cover classification using time series, see [Gomez et al., 2016]. As an alternative to *Space-first, time-later* methods, the SITS package provides support for classification of time series, preserving the full temporal resolution of the input data, using a *time-first, space-later* approach. SITS uses all data in the image time series to create larger dimensional spaces for machine learning. The idea is to have as many temporal attributes as possible, increasing the dimension of the classification space. Each temporal instance of a time series is taken as an independent dimension in the feature space of the classifier. To the authors' best knowledge, the classification techniques for image time series included in the package are not previoulsy available in other R or python packages. Furthermore, the package includes methods for filtering, clustering and post-processing that also have not been published in the literature.

Image data cubes as the basis for big Earth observation data analysis

In broad terms, the cloud computing model is one where large satellite-generated data sets are archived on cloud services, which also provide computing facilities to process them. By using cloud services, users can share big Earth observation databases and minimize the amount of data download. Investment in infrastructure is minimised and sharing of data and software increases. However, data available in the cloud is best organised for analysis by creating data cubes.

Generalising Appel and Pebesma [2019], we consider that a data cube is a four-dimensional structure with dimensions x (longitude or easting), y (latitude or northing), time, and bands. Its spatial dimensions refer to a single spatial reference system (SRS). Cells of a data cube have a constant spatial size (with regard to the cube's SRS). The temporal dimension is specified by a set of intervals. For every combination of dimensions, a cell has a single value. Data cubes are particularly amenable for machine learning techniques; their data cane be transformed into arrays in memory, which can be fed to training and classification algorithms. Given the widespread availability of large data sets of Earth observation data, there is a growing interest in organising large sets of data into "data cubes".

As explained below, a data cube is the data type used in `sits` to handle dense raster data. Many of the operations involve creating, transforming and analysing data cubes.

Using Web Data Services to Access Image Data Cubes

One of the distinguishing features of SITS is that it has been designed to work with big satellite image data sets which reside on the cloud and with data cubes. Many *R* packages that work with remote sensing images require data to be accessible in a local computer. However, with the coming of age of big Earth observation data, it is not always practical to transfer large data sets. Users have to rely on web services to provide access to these data sets. In this context, SITS is based on access to data cubes using web services.

A web service is software system designed to support remote access to image collections through APIs. These are machine-to-machine protocols that allow access to image collections and to generate *data cubes*. SITS uses two kinds of services: those that provide time series data and those that provide access to data cubes. Currently, there are no established standards for these two services. For this reason, `sits` uses two new protocols: WTSS ("Web Time Series Serice") and EOCubes ("Earth observation Data Cubes Services") that we have develop. WTSS is a light-weight service for retrieval of time series data for selected locations and periods [Vinhas et al., 2016]. The WTSS R client is available in the CRAN archive using the `wtss` package. The EOCUBES service provides access to big data collections, which are organised by the `EOCubes` package. the package also allows users to work on local files, if desired.

To find out the services and respective data cubes available, one should use the `sits_services()` function.

```r
sits_services()
```

```
## Service: "WTSS"
##    Cube: "MOD13Q1"
##   Bands: "mir", "blue", "nir", "red", "evi", "ndvi"
##    Cube: "MOD13Q1_M"
##   Bands: "quality", "reliability"
## Service: "SATVEG"
## Service: "EOCUBES"
##    Cube: "MOD13Q1/006"
##       Bands: "blue", "evi", "mir", "ndvi", "nir", "red"
```

These services are set on the SITS configuration file, which is described later in this document. For each services, the above function lists the names of the data cubes available and additional information.

Defining a data cube using the WTSS service

To define a data cube for the WTSS, in principle the following parameters should be provided: (a) the name of the service; (b) the URL; (c) the satellite and sensor associated to the cube, and (d) name of the data cube in the remote service. If the URL, satellite and sensor are not provided, the package will search for default information in the SITS configuration file.

```
# define the data cube "MOD13Q1" using the WTSS service
# In this case, the WTSS service is run by a server in INPE Brazil
wtss_cube <- sits_cube(service = "WTSS", name = "MOD13Q1")

# get information on the data cube
wtss_cube %>% dplyr::select(service, URL, satellite, sensor)
```

```
## # A tibble: 1 x 4
##   service URL                                   satellite sensor
##   <chr>   <chr>                                 <chr>     <chr>
## 1 WTSS    http://www.esensing.dpi.inpe.br/wtss/ TERRA     MODIS
```

```
# spatial dimensions of the data cube
wtss_cube %>% dplyr::select(xmin, xmax, ymin, ymax)
```

```
## # A tibble: 1 x 4
##    xmin  xmax  ymin  ymax
##   <dbl> <dbl> <dbl> <dbl>
## 1 -81.2 -30.0 -40.0 10.00
```

```
# temporal dimension of the data cube
timeline <- sits_timeline(wtss_cube)
message(paste0("start date = ", timeline[1], " end date = ", timeline[length(timeline)], "
```

```
## start date = 2000-02-18 end date = 2018-06-26 steps = 423
```

```
# bands of the data cube
sits_bands(wtss_cube)
```

```
## [1] "mir"  "blue" "nir"  "red"  "evi"  "ndvi"
```

Defining a data cube using the EOCUBES service

In the case of EOCUBES service, additional obtional parameters are: (a) a string with tile names; (b) a geometry to restrict the spatial extent; (c) start and end dates to restrict the temporal extent. If desired, the cube based on the EOCUBES service can be specified only by the service and cube names, provided that the correct additional information is provided in the configuration file.

```r
# create a coverage from EOCUBES service from the collection "MOD13Q1/006"
modis_cube <- sits_cube(service = "EOCUBES",
                        name    = "MOD13Q1/006")

# get information on the data cube
modis_cube %>% dplyr::select(service, URL, satellite, sensor)
```

```
## # A tibble: 1 x 4
##   service URL     satellite sensor
##   <chr>   <chr>   <chr>     <chr>
## 1 EOCUBES eocubes TERRA     MODIS
```

```r
# get information on the cube
modis_cube %>% dplyr::select(xmin, xmax, ymin, ymax, timeline)
```

```
## # A tibble: 1 x 5
##        xmin      xmax      ymin     ymax timeline
##       <dbl>     <dbl>     <dbl>    <dbl> <list>
## 1 -6782898. -5448558. -2112706. -778365. <list [1]>
```

Defining a data cube using files organised as raster bricks

The SITS package enables uses to create data cube based on files. In this case, these files should be organized as `raster bricks`. A RasterBrick is a multi-layer raster object used by the *R* `raster` package. Each brick is a multi-layer file, containing different time instances of one spectral band. To allow users to create data cubes based on files, SITS needs to know what is the timeline of the data sets and what are the names of the files that contain the RasterBricks. The example below shows two bricks which are available at the AWS service, each containing 392 time instances of the "ndvi" and "evi" bands for the years 2000 to 2016. The timeline is available as part of the SITS package.

In the example, neither the satellite nor the sensor are provided; this information is deduced by SITS considering the cartographical projection associated to the image. If the projection is "sinusoidal", SITS will deduce these are MODIS files; data on "utm" projection is associated to the LANDSAT-8 satellite.

```r
# Obtain a raster brick with 23 instances for one year
# Select the bands "ndvi", "evi" from bricks  available in the "inSitu" package
evi_file <- system.file("extdata/Sinop", "Sinop_evi_2014.tif", package = "inSitu")
ndvi_file <- system.file("extdata/Sinop", "Sinop_ndvi_2014.tif", package = "inSitu")

# Obtain the associated timeline
time_file <- system.file("extdata/Sinop", "timeline_2014.txt", package = "inSitu")
timeline_2013_2014 <- scan(time_file, character())
```

```r
# create a raster metadata file based on the information about the files
raster_cube <- sits_cube(name = "Sinop", timeline = timeline_2013_2014, bands = c("ndvi", "
```

```
## satellite information not provided - assuming TERRA
```

```
## sensor information not provided - assuming MODIS
```

```r
# get information on the data cube
raster_cube %>% dplyr::select(service, URL, satellite, sensor)
```

```
## # A tibble: 1 x 4
##   service URL              satellite sensor
##   <chr>   <chr>            <chr>     <chr>
## 1 BRICK   http://127.0.0.1 TERRA     MODIS
```

```r
# get information on the coverage
raster_cube %>% dplyr::select(xmin, xmax, ymin, ymax)
```

```
## # A tibble: 1 x 4
##        xmin      xmax      ymin      ymax
##       <dbl>     <dbl>     <dbl>     <dbl>
## 1 -6087719. -5984864. -1355172. -1256255.
```

To create the raster cube, we a set of consistent raster bricks (one for each satellite band) and a `timeline` that matches the input images of the raster brick. Once created, the coverage can be used either to retrieve time series data from the raster bricks using `sits_get_data()` or to do the raster classification by calling the function `sits_classify`.

Data structures for satellite image time series

The `sits` package requires a set of time series data, describing properties in spatio-temporal locations of interest. For land use classification, this set consists of samples provided by experts that take *in-situ* field observations or recognize land classes using high resolution images. The package can also be used for any type of classification, provided that the timeline and bands of the time series (used for training) match that of the data cubes.

For handling time series, the package uses a `sits tibble` to organize time series data with associated spatial information. A `tibble` is a generalization of a `data.frame`, the usual way in *R* to organise data in tables. Tibbles are part of the `tidyverse`, a collection of R packages designed to work together in data manipulation [Wickham and Grolemund, 2017]. As a example of how the `sits` tibble works, the following code shows the first three lines of a tibble containing 2,115 labelled samples of land cover in Mato Grosso state of Brazil. It is the most important agricultural frontier of Brazil and it is the largest producer of soybeans, corn, and cotton. The samples contain time series extracted from the MODIS MOD13Q1 product from 2000 to 2016, provided every 16 days at 250-meter spatial resolution in the Sinusoidal projection. Based on ground surveys and high resolution imagery, it includes 425 samples of nine classes: "Forest", "Cerrado", "Pasture", "Soybean-fallow", "Fallow-Cotton", "Soybean-Cotton", "Soybean-Corn", "Soybean-Millet", and "Soybean-Sunflower".

```
# data set of samples
data(samples_mt_6bands)
samples_mt_6bands[1:3,]
```

```
## # A tibble: 3 x 7
##   longitude latitude start_date end_date   label   cube    time_series
##       <dbl>    <dbl> <date>     <date>     <chr>   <chr>   <list>
## 1     -58.6    -13.9 2007-09-14 2008-08-28 Cerrado MOD13Q1 <tibble [23 x 7~
## 2     -59.7    -13.6 2006-09-14 2007-08-29 Cerrado MOD13Q1 <tibble [23 x 7~
## 3     -52.6    -15.0 2008-09-13 2009-08-29 Cerrado MOD13Q1 <tibble [23 x 7~
```

A `sits` `tibble` contains data and metadata. The first six columns contain the metadata: spatial and temporal information, label assigned to the sample, and the data cube from where the data has been extracted. The spatial location is given in longitude and latitude coordinates for the "WGS84" ellipsoid. For example, the first sample has been labelled "Cerrado, at location (−58.5631, −13.8844), and is considered valid for the period (2007-09-14, 2008-08-28). Informing the dates where the label is valid is crucial for correct classification. In this case, the researchers involved in labeling the samples chose to use the agricultural calendar in Brazil, where the spring crop is planted in the months of September and October, and the autumn crop is planted in the months of February and March. For other applications and other countries, the relevant dates will most likely be different from those used in the example. The `time_series` column contains the time series data for each spatiotemporal location. This data is also organized as a tibble, with a column with the dates and the other columns with the values for each spectral band.

```
# print the first time series records of the first sample
sits_time_series(samples_mt_6bands[1,])[1:3,]
```

```
## # A tibble: 3 x 7
##   Index        mir   blue   nir    red    evi  ndvi
##   <date>     <dbl>  <dbl> <dbl>  <dbl>  <dbl> <dbl>
## 1 2007-09-14 0.163 0.0417 0.173 0.0827 0.166  0.352
## 2 2007-09-30 0.134 0.0468 0.165 0.112  0.0887 0.191
## 3 2007-10-16 0.102 0.0208 0.144 0.0533 0.173  0.459
```

The `sits` package provides functions for data manipulation and displaying information for `sits` tibbles. For example, `sits_labels()` shows the labels of the sample set and their frequencies.

```r
sits_labels(samples_mt_6bands)
```

```
## # A tibble: 9 x 3
##   label          count   prop
##   <chr>          <int>  <dbl>
## 1 Cerrado           80 0.188
## 2 Fallow_Cotton      7 0.0165
## 3 Forest            28 0.0659
## 4 Pasture           74 0.174
## 5 Soy_Corn          80 0.188
## 6 Soy_Cotton        80 0.188
## 7 Soy_Fallow        18 0.0424
## 8 Soy_Millet        47 0.111
## 9 Soy_Sunflower     11 0.0259
```

In many cases, it is useful to relabel the data set. For example, there may be situations when one wants to use a smaller set of labels, since samples in one label on the original set may not be distinguishable from samples with other labels. We then could use `sits_relabel()`, which requires a conversion list (for details, see `?sits_relabel`).

Given that we have used the tibble data format for the metadata and and the embedded time series, one can use the functions from `dplyr`, `tidyr` and `purrr` packages of the `tidyverse` [Wickham and Grolemund, 2017] to process the data. For example, the following code uses `sits_select_bands()` to get a subset of the sample data set with two bands (NDVI and EVI) and then uses the `dplyr::filter()` to select the samples labelled either as "Cerrado" or "Pasture". We can then use the `sits_plot()` to display the time series. Given a small number of samples to display, `sits_plot()` tries to group as many spatial locations together. In the following example, the first 15 samples of "Cerrado" class refer to the same spatial location in consecutive time periods. For this reason, these samples are plotted together.

```r
# select NDVI band
samples_ndvi.tb <- sits_select_bands(samples_mt_6bands, ndvi)
# select only samples with Cerrado label
samples_cerrado.tb <-
    dplyr::filter(samples_ndvi.tb, label == "Cerrado")
# plot the first sample
sits_plot(samples_cerrado.tb[1,])
```

For a large number of samples, where the amount of individual plots would be substantial, the default visualization combines all samples together in a single temporal interval (even if they belong to different years). All samples with the same band and label are aligned to a common time interval. This plot is useful to show the spread of values for the time series of each band. The strong red line in the plot shows the median of the values, while the two orange lines are the first and third interquartile ranges. The documentation of `sits_plot()` has more details about the different ways it can display data.

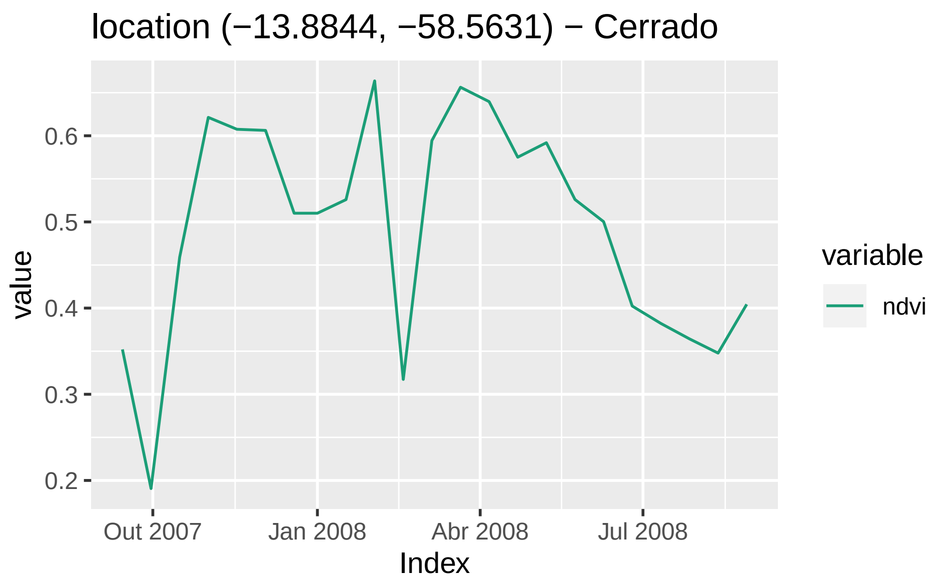


Figure 1: Plot of the first 'Cerrado' sample from data set

```
# plot all cerrado samples together
sits_plot(samples_cerrado.tb)
```

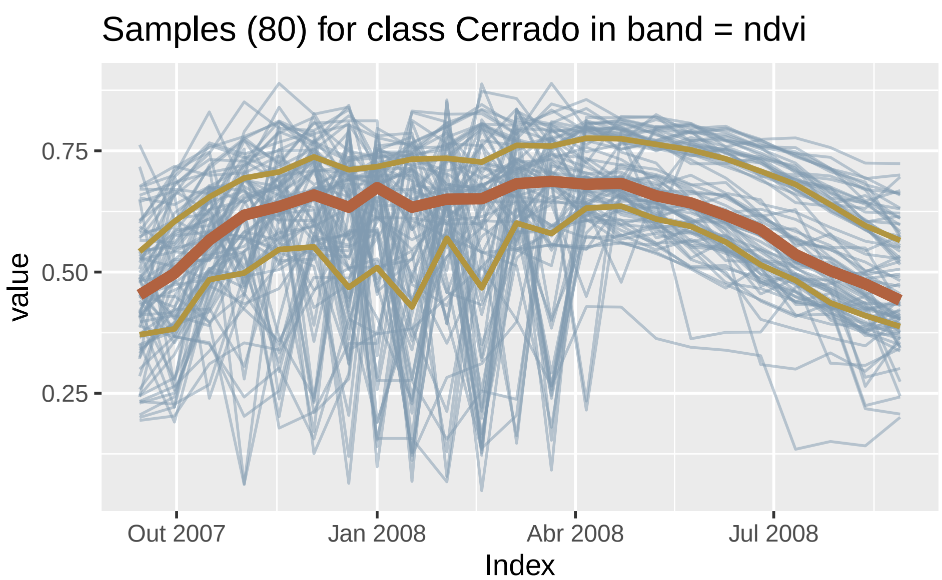


Figure 2: Plot of all Cerrado samples from data set

Obtaining time series data

To get a time series in SITS, one has to create a data cube first, as described above. Alternatively, the time series can also be converted from data stored in the ZOO format [Zeileis and Grothendieck, 2005]. Users can request one or more time series points from a data cube by using `sits_get_data()`. This function provides a general means of access to image time series. Given data cue, the user provides the latitude and longitude of the desired location, the bands, and the start date and end date of the time series. If the start and end dates are not provided, it retrieves all the available period. The result is a tibble that can be visualized using `sits_plot()`.

```
# a point in the transition forest to pasture in Northern MT
# obtain a time series from the WTSS server for this point
# define the data cube "MOD13Q1" using the WTSS service
# In this case, the WTSS service is run by a server in INPE Brazil
wtss_cube <- sits_cube(service = "WTSS",
                       name    = "MOD13Q1")
series.tb <- sits_get_data(cube      = wtss_cube,
                           longitude = -55.57320,
                           latitude  = -11.50566,
                           bands     = c("ndvi", "evi"))
sits_plot(series.tb)
```

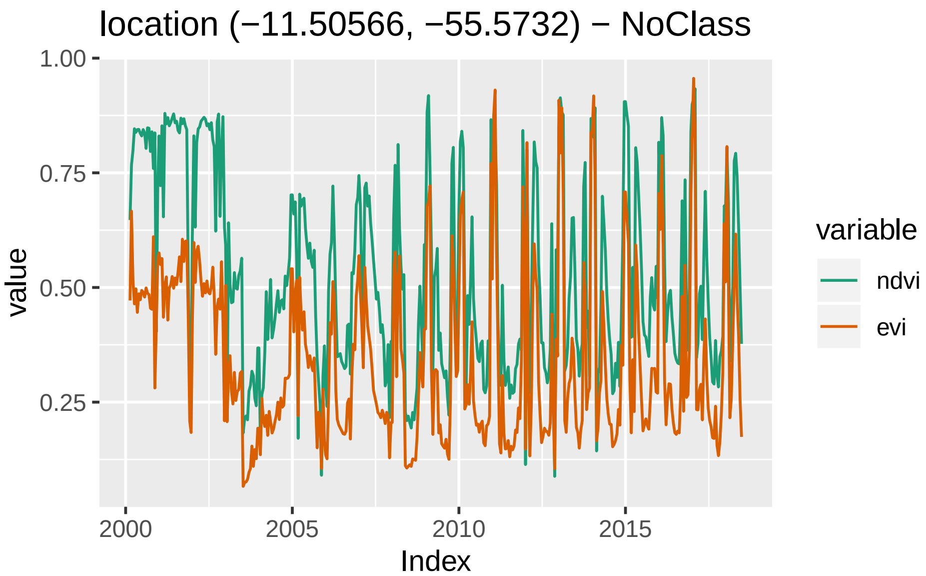


Figure 3: NDVI and EVI time series fetched from WTSS service.

A useful case is when a set of labelled samples are available to be used as a training data set. In this case, one usually has trusted observations which are labelled and commonly stored in plain text CSV files. Function `sits_get_data()` can get a CSV file path as an argument. The CSV file must provide, for each time series, its latitude and longitude, the start and end dates, and a label associated to a ground sample. An example of a CSV file used is shown below:

```
# print the first line of a CSV file used to retrieve data
csv_sinop <- system.file("extdata/samples/samples_matogrosso.csv", package = "sits")
head(read.csv(csv_sinop))
```

```
## # A tibble: 6 x 6
##      id longitude latitude start_date end_date   label
##   <int>     <dbl>    <dbl> <fct>      <fct>      <fct>
## 1     1     -55.0    -15.2 2015-09-14 2016-08-28 Pasture
## 2     2     -55.0    -15.2 2015-09-14 2016-08-28 Pasture
## 3     3     -55.0    -15.2 2015-09-14 2016-08-28 Pasture
## 4     4     -46.6    -10.4 2004-09-13 2005-08-29 Cerrado
## 5     5     -46.4    -10.9 2007-09-13 2008-08-29 Cerrado
## 6     6     -46.4    -10.9 2006-09-13 2007-08-29 Cerrado
```

```
# read the first three samples from the CSV file
csv_data <- sits_get_data(cube = wtss_cube, file = csv_sinop, .n_max = 3)
csv_data
```

```
## # A tibble: 3 x 7
##   longitude latitude start_date end_date   label   cube    time_series
##       <dbl>    <dbl> <date>     <date>     <chr>   <chr>   <list>
## 1     -55.0    -15.2 2015-09-14 2016-08-28 Pasture MOD13Q1 <tibble [23 x 7~
## 2     -55.0    -15.2 2015-09-14 2016-08-28 Pasture MOD13Q1 <tibble [23 x 7~
## 3     -55.0    -15.2 2015-09-14 2016-08-28 Pasture MOD13Q1 <tibble [23 x 7~
```

A common situation is when users have samples available as shapefiles in point format. Since shapefiles contain only geometries, we need to provide information about the start and end times for which each label is valid. in this case, one should use the function `sits_shp_to_csv` to produce a CSV which describes the data to be read from a data cube. For best results, one needs to pre-select the data cube from which data will be read, and use the timeline of this data cube.

```
# select the cube
wtss_cube <- sits_cube(service = "WTSS", name    = "MOD13Q1")
#  get the timeline from the cube
cube_timeline <- sits_timeline(wtss_cube)
# define the input shapefile (consisting of POINTS)
shpfile <- system.file("extdata/shapefiles/cerrado_forested.shp", package = "sits")
# set the start and end dates for the validity of the labels of the points in the shapefile
start_date <- lubridate::ymd("2002-08-29")
end_date   <- lubridate::ymd("2013-08-13")
# define the output csv file
csvfile <- paste0("cerrado_forested.csv")
#' # define the label
label <- "Cerrado_Forested"
# read the points in the shapefile and produce a CSV file
sits_shp_to_csv(shpfile, csvfile, label, cube_timeline, start_date, end_date, interval = "1
# read the first three samples from the CSV file
csv_data <- sits_get_data(cube = wtss_cube, file = csvfile, .n_max = 3)
csv_data
```

```
## # A tibble: 3 x 7
##   longitude latitude start_date end_date   label       cube   time_series
##       <dbl>    <dbl> <date>      <date>     <chr>        <chr>  <list>
## 1    -47.1    -11.2 2002-08-29 2003-08-13 Cerrado_Fo~ MOD13~ <tibble [23 ~
## 2    -47.1    -11.2 2003-08-29 2004-08-12 Cerrado_Fo~ MOD13~ <tibble [23 ~
## 3    -47.1    -11.2 2004-08-28 2005-08-13 Cerrado_Fo~ MOD13~ <tibble [23 ~
```

Filtering techniques

The literature on satellite image time series have several applications of filtering to correct or smooth vegetation index data. The following filters are available in SITS and are described in more detail in the vignette "Satellite Image Time Series Filtering with SITS":

- Savitzky–Golay filter (`sits_sgolay`)
- Whittaker filter (`sits_whittaker`)
- Envelope filter (`sits_envelope`)
- ARIMA filter for cloud removal in NDVI band (`sits_ndvi_arima`)
- Cloud filter (`sits_cloud_removal`)
- Kalman filter (`sits_kalman`)

The SITS package uses a common interface to all filter functions with the `sits_filter`. The function has two parameters: `data` for the dataset to be filtered and `filter` for the filter to be applied. To aid on data visualisation, all bands which are filtered have a suffix which is appended, as shown in the examples below. Here we show an example using the Whittaker smoother, which has been proposed in literature [Atzberger and Eilers, 2011] as arguably the most appropriate one to use for satellite image time series. The Whittaker smoother attempts to fit a curve that represents the raw data, but is penalized if subsequent points vary too much [Atzberger and Eilers, 2011]. As such, it balances between the residual to the original data and the "smoothness" of the fitted curve. It uses the parameter `lambda` to control the degree of smoothing. I

```
# Take a NDVI time series of 16 years, apply Whittaker filter and plot the series
point_whit <- sits_filter(point_ndvi, filter = sits_whittaker(lambda = 5.0))
# merge with original data and plot the original and the filtered data
point_whit %>%
  sits_merge(point_ndvi) %>%
  sits_plot()
```

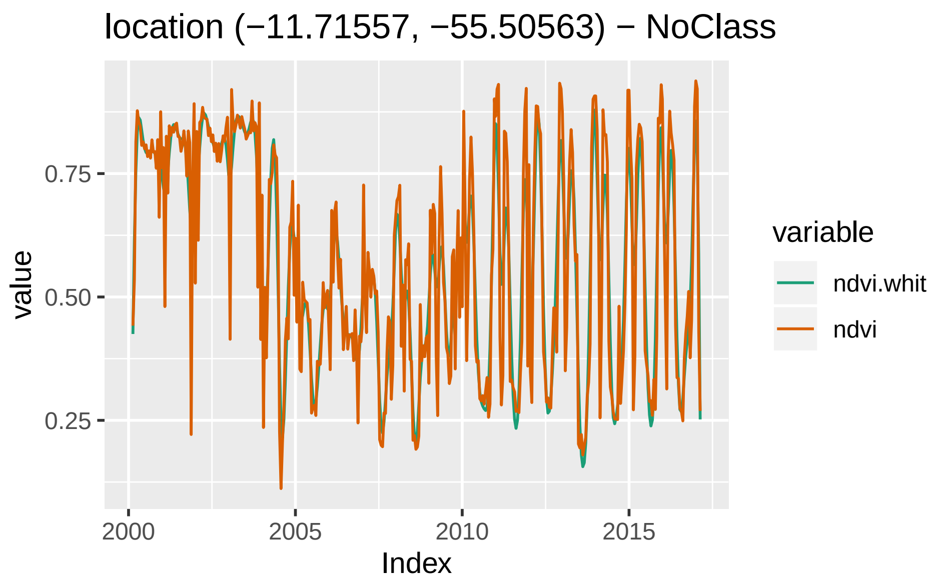


Figure 4: Whittaker smoother filter applied on one-year NDVI time series. The example uses default $\lambda = 3$ parameter.

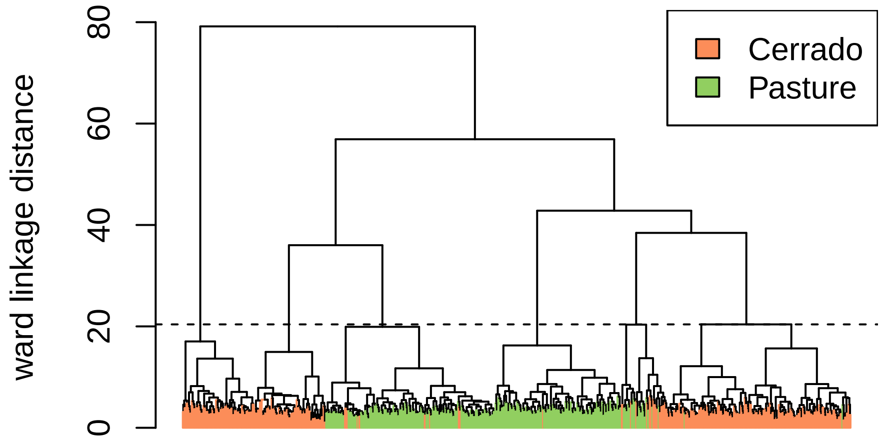
Clustering for sample quality control

One of the key challenges of machine learning classification models is assessing the quality of the training data sets. It is useful to apply pre-processing methods to improve the quality of the samples and to remove those that might have been wrongly labeled or that have low discriminatory power. Good samples lead to good classification maps. `sits` provides support for two clustering methods to test sample quality: (a) Agglomerative Hierarchical Clustering (AHC); (b) Self-organizing Maps (SOM). Full details of the cluster methods used in SITS are available in the vignette 'Clustering of Satellite Image Time Series with SITS'.

Hierarchical clustering

Agglomerative hierarchical clustering (AHC) is a method that computes the dissimilarity between any two elements from a data set and builds a dendrogram, which is useful to decide the number of clusters to partition the data. In `sits`, AHC is implemented by the `sits_cluster_dendro` function, which generates a dendrogram and then determines a possible best cut using the *Adjusted Rand Index* (ARI). Based on this cut, it generates a set of clusters. The cluster information is appended as an additional column of the tibble, called "cluster".

```
# take a set of patterns for 2 classes
# create a dendrogram object, plot, and get the optimal cluster based on ARI index
clusters.tb <- sits_cluster_dendro(cerrado_2classes, bands = c("ndvi", "evi"))
```




```
# show clusters samples frequency
sits_cluster_frequency(clusters.tb)
```

```
##
##              1   2   3   4   5   6 Total
##   Cerrado 203  13  23  80   1  80   400
##   Pasture   2 176  28   0 140   0   346
##   Total   205 189  51  80 141  80   746
```

After the clusters have been determined, the function `sits_cluster_frequency()` shows how the samples are distributed across the clusters. It helps to identify two problems: (a) small amount of samples in clusters dominated by another class (*e.g.* clusters 1, 2, 4, 5, and 6); and (b) samples in mixed clusters (*e.g.* cluster 3). It is possible to remove clusters with mixed classes using standard `R` fucntions such as those in the `dplyr` package. In the example above, removing cluster 3 can be done using the `dplyr::filter` function.

```
# remove cluster 3 from the samples
clusters_new.tb <- dplyr::filter(clusters.tb, cluster != 3)

# show new clusters samples frequency
sits_cluster_frequency(clusters_new.tb)
```

```
##
##              1   2   4   5   6 Total
##   Cerrado 203  13  80   1  80   377
##   Pasture   2 176   0 140   0   318
##   Total   205 189  80 141  80   695
```

The resulting clusters still contain mixed labels, possibly resulting from outliers. In this case, users may want to remove the outliers and leave only the most frequent class. To do this, one can use `sits_cluster_clean()`, which removes all minority samples, as shown below. The resulting set of samples can be used for classification, as discussed in the “machine learning” section.

```r
# clear clusters, leaving only the majority class in each cluster
cleaned.tb <- sits_cluster_clean(clusters_new.tb)
# show clusters samples frequency
sits_cluster_frequency(cleaned.tb)
```

```
##
##              1   2   4   5   6 Total
##   Cerrado 203   0  80   0  80   363
##   Pasture   0 176   0 140   0   316
##   Total   203 176  80 140  80   679
```

Self-organizing maps

As an alternative for hierachical clustering for quality control of training samples, SITS provides a clustering technique based on self-organizing maps (SOM). SOM is a dimensionality reduction technique [Kohonen et al., 2001], where high-dimensional data is mapped into two dimensions, keeping the topological relations between data patterns. The input data is a set of tranining samples which are typically of a high-dimension. For example, a time series of 25 instances of 4 spectral bands is a 100-dimensional data set. The output layer is a 2D grid of neurons, each associated to a weight vector of the same dimension as the input space. The general idea of SOM-based clustering is that, by projecting the high-dimensional data set of training samples into a 2D map, good quality samples of each class should be close together in the resulting map. SITS provides the convenience function `sits_cluster_som` to perform clustering using SOM, This function uses self-organized maps to find clusters in satellite image time series for quality control of the samples.

```r
# clustering time series using SOM
# return a tibble with all samples and which cluster it belongs
som_cluster.tb <-
    sits::sits_cluster_som(prodes_226_064,
        grid_xdim = 10,
        grid_ydim = 10,
        alpha = 1.0,
        distance = "euclidean",
        iterations = 50,
        prob_label_change = 0.8,
        min_cluster_prob = 0.6)
```

Classification using machine learning

There has been much recent interest in using classifiers such as support vector machines [Mountrakis et al., 2011] and random forests [Belgiu and Dragut, 2016] for remote sensing images. Most often, researchers use a *space-first, time-later* approach, in which the dimension of the decision space is limited to the number of spectral bands or their transformations. Sometimes, the decision space is extended with temporal attributes. To do this, researchers filter the raw data to get smoother time series [Brown et al., 2013, Kastens et al., 2017]. Then, using software such as TIMESAT [Jönsson and Eklundh, 2004], they derive a small set of phenological parameters from vegetation indexes, like the beginning, peak, and length of the growing season [Estel et al., 2015, Pelletier et al., 2016].

In a recent review of machine learning methods to classify remote sensing data [Maxwell et al., 2018], the authors note that many factors influence the performance of these classifiers, including the size and quality of the training dataset, the dimension of the feature space, and the choice of the parameters. We support both *space-first, time-later* and *time-first, space-later* approaches. Therefore, the `sits` package provides functionality to explore the full depth of satellite image time series data.

When used in *time-first, space-later* approache, `sits` treats time series as a feature vector. To be consistent, the procedure aligns all time series from different years by its time proximity considering an given cropping schedule. Once aligned, the feature vector is formed by all pixel "bands". The idea is to have as many temporal attributes as possible, increasing the dimension of the classification space. In this scenario, statistical learning models are the natural candidates to deal with high-dimensional data: learning to distinguish all land cover and land use classes from trusted samples exemplars (the training data) to infer classes of a larger data set.

The SITS package provides a common interface to all machine learning models, using the `sits_train` function. this function takes two parameters: the input data samples and the ML method (`ml_method`), as shown below. After the model is estimated, it can be used to classify individual time series or full data cubes using the `sits_classify` function. In the examples that follow, we show how to apply each method for the classification of a single time series. Then, we disscuss how to classify full data cubes.

When a dataset of time series organised as a SITS tibble is taken as input to the classifier, the result is the same tibble with one additional column ("predicted"), which contains the information on what labels are have been assigned for each interval. The following example illustrate how to train a dataset and classify an individual time series. First we use the `sits_train` function with two parameters: the training dataset (described above) and the chosen machine learning model (in this case, a random forest classifier). The trained model is then used to classify a time series from Mato Grosso Brazilian state, using `sits_classify`. The results can be shown in text format using the function `sits_show_prediction` or graphically using `sits_plot`.

```
#select the data for classification
mato_grosso_samples <- inSitu::br_mt_1_8K_9classes_6bands
mato_grosso_4bands  <- sits_select_bands(mato_grosso_samples, ndvi, evi, nir, mir)

# get a point to be classified
point_mt_4bands <- sits_select_bands(point_mt_6bands, ndvi, evi, nir, mir)

# Train a machine learning model for the mato grosso dataset using Random Forest
```

```r
model <- sits_train(data = mato_grosso_4bands, ml_method = sits_rfor())

# Classify using random forest model and plot the result
class.tb <- sits_classify(point_mt_4bands, model)
# show the results of the prediction
sits_show_prediction(class.tb)
```

```
## # A tibble: 17 x 3
##    from       to         class
##    <date>     <date>     <chr>
##  1 2000-09-13 2001-08-29 Forest
##  2 2001-09-14 2002-08-29 Forest
##  3 2002-09-14 2003-08-29 Forest
##  4 2003-09-14 2004-08-28 Pasture
##  5 2004-09-13 2005-08-29 Pasture
##  6 2005-09-14 2006-08-29 Pasture
##  7 2006-09-14 2007-08-29 Pasture
##  8 2007-09-14 2008-08-28 Pasture
##  9 2008-09-13 2009-08-29 Pasture
## 10 2009-09-14 2010-08-29 Soy_Corn
## 11 2010-09-14 2011-08-29 Soy_Corn
## 12 2011-09-14 2012-08-28 Soy_Corn
## 13 2012-09-13 2013-08-29 Soy_Corn
## 14 2013-09-14 2014-08-29 Soy_Corn
## 15 2014-09-14 2015-08-29 Soy_Corn
## 16 2015-09-14 2016-08-28 Soy_Corn
## 17 2016-09-13 2017-08-29 Soy_Corn
```

```r
# plot the results of the prediction
sits_plot(class.tb)
```

The following methods are available in SITS for training machine learning models:

- Linear discriminant analysis (`sits_lda`)
- Quadratic discriminant analysis (`sits_qda`)
- Multinomial logit and its variants 'lasso' and 'ridge' (`sits_mlr`)
- Support vector machines (`sits_svm`)
- Random forests (`sits_rfor`)
- Extreme gradient boosting (`sits_xgboost`)
- Deep learning (DL) using multi-layer perceptrons (`sits_deeplearning`)

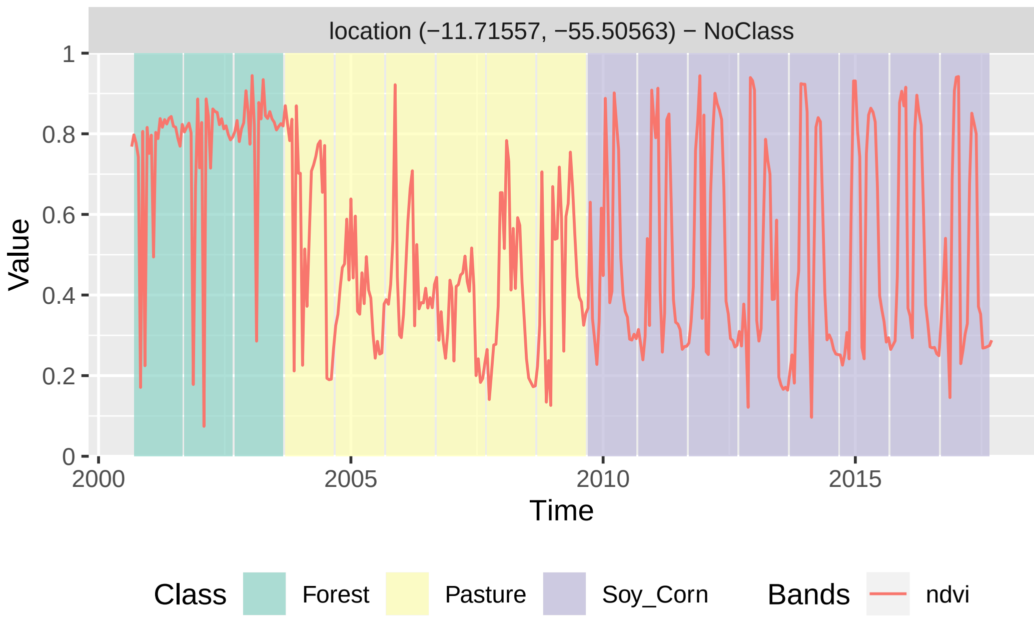


Figure 5: SVM classification of a 16 years time series. The location (latitude, longitude) shown at the top of the graph is in geographic coordinate system (WGS84 *datum*).

- DL with 1D convolutional neural networks (`sits_CNN`),
- DL combining 1D CNN and multi-layer perceptron networks (`sits_tempCNN`)
- DL using 1D version of ResNet (`sits_ResNet`).

For more details on each method, please see the vignette "Machine Learning for Data Cubes using the SITS package".

Validation techniques

Validation is a process undertaken on models to estimate some error associated with them, and hence has been used widely in different scientific disciplines. Here, we are interested in estimating the prediction error associated to some model. For this purpose, we concentrate on the *cross-validation* approach, probably the most used validation technique [Hastie et al., 2009].

To be sure, cross-validation estimates the expected prediction error. It uses part of the available samples to fit the classification model, and a different part to test it. The so-called *k-fold* validation, we split the data into k partitions with approximately the same size and proceed by fitting the model and testing it k times. At each step, we take one distinct partition for test and the remaining $k-1$ for training the model, and calculate its prediction error for classifying the test partition. A simple average gives us an estimation of the expected prediction error.

A natural question that arises is: *how good is this estimation?* According to Hastie et al. [2009], there is a bias-variance trade-off in choice of k. If k is set to the number of samples, we obtain the so-called *leave-one-out* validation, the estimator gives a low bias for the true expected error, but produces a high variance expectation. This can be computational expensive as it requires the same number of fitting process as the number of samples. On the other hand, if we choose $k = 2$, we get a high biased expected prediction error estimation that overestimates the true prediction error, but has a low variance. The recommended choices of k are 5 or 10 [Hastie et al., 2009], which somewhat overestimates the true prediction error.

`sits_kfold_validate()` gives support the k-fold validation in `sits`. The following code gives an example on how to proceed a k-fold cross-validation in the package. It perform a five-fold validation using SVM classification model as a default classifier. We can see in the output text the corresponding confusion matrix and the accuracy statistics (overall and by class).

```
# perform a five fold validation for the "cerrado_2classes" data set
# XGBoost machine learning method using default parameters
prediction.mx <- sits_kfold_validate(cerrado_2classes, folds = 5, ml_method = sits_xgboost(
# prints the output confusion matrix and statistics
sits_conf_matrix(prediction.mx)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction Cerrado Pasture
##    Cerrado     388      15
##    Pasture      12     331
##
##            Accuracy : 0.9638
##              95% CI : (0.9478, 0.976)
##
##               Kappa : 0.9272
##
##  Prod Acc  Cerrado : 0.9700
##  Prod Acc  Pasture : 0.9566
##  User Acc  Cerrado : 0.9628
##  User Acc  Pasture : 0.9650
##
```

Cube classification

The continuous observation of the Earth surface provided by orbital sensors is unprecedented in history. Just for the sake of illustration, a unique tile from MOD13Q1 product, a square of 4800 pixels provided every 16 days since February 2000 takes around 18GB of uncompressed data to store only one band or vegetation index. This data deluge puts the field into a big data era and imposes challenges to design and build technologies that allow the Earth observation community to analyse those data sets [Câmara et al., 2017].

To classify a data cube, use the function `sits_classify()` as described below. This function works both with cubes built from raster bricks and those built with services such as "EOCUBES". The classification algorithms allows users to choose how many process will run the task in parallel, and also the size of each data chunk to be consumed at each iteration. This strategy enables `sits` to work on average desktop computers without depleting all computational resources. The code bellow illustrates how to classify a small raster brick image that accompany the package.

Steps for cube classification

Once a data cube which has associated files is defined, the steps for classification are:

1. Select a set of training samples.
2. Train a machine learning model
3. Classify the data cubes using the model, producing a data cube with class probabilities.
4. Label the cube with probabilities, including data smoothing if desired.

Adjustments for improved performance

To reduce processing time, it is necessary to adjust `sits_classify()` according to the capabilities of the server. The package tries to keep memory use to a minimum, performing garbage collection to free memory as often as possible. Nevertheless, there is an inevitable trade-off between computing time, memory use, and I/O operations. The best trade-off has to be determined by the user, considering issues such disk read speed, number of cores in the server, and CPU performance.

The first parameter is `memsize`. It controls the size of the main memory (in GBytes) to be used for classification. The user must specify how much free memory will be available. The second factor controlling performance of raster classification is `multicores`. Once a block of data is read from disk into main memory, it is split into different cores, as specified by the user. In general, the more cores are assigned to classification, the faster the result will be. However, there are overheads in switching time, especially when the server has other processes running.

Based on current experience, the classification of a MODIS tile (4800 x 4800) with four bands and 400 time instances, covering 15 years of data, using SVM with a training data set of about 10,000 samples, takes about 24 hours using 20 cores and a memory size of 60 GB, in a server with 2.4GHz Xeon CPU and 96 GB of memory to produce the yearly classification maps.

```r
# Retrieve the set of samples for the Mato Grosso region
# Select the data for classification
mato_grosso_samples <- inSitu::br_mt_1_8K_9classes_6bands
mato_grosso_2bands  <- sits_select_bands(mato_grosso_samples, ndvi, evi)

# build a machine learning model for this area
xbg_model <- sits_train(mato_grosso_2bands, sits_xgboost())

# Use the raster cube created in the section "Defining a data cube using files" above
# Classify the raster cube, generating a probability file
probs_cube <- sits_classify(raster_cube, ml_model = xbg_model, memsize = 2, multicores = 1)
```

```
## Starting classification at 2019-09-11 14:03:52

## Classification finished at 2019-09-11 14:04:10. Total elapsed time: 0.3 minute(s).
```

```r
# label the probability file (by default selecting the class with higher probability)
label_cube <- sits_label_classification(probs_cube)

# plot the first raster object with a selected color pallete
# make a title, define the colors and the labels)
sits_plot_raster(label_cube, time = 1, title = "SINOP-MT - 2000/2001")
```

Smoothing of raster data after classification

Post-processing is a desirable step in any classification process. Most statistical classifiers use training samples derived from "pure" pixels, that have been selected by users as representative of the desired output classes. However, images contain many mixed pixels irrespective of the resolution. Also, there is a considerable degree of data variability in each class. These effects lead to outliers whose chance of misclassification is significant. To offset these problems, most post-processing methods use the "smoothness assumption" [Schindler, 2012]: nearby pixels tend to have the same label. To put this assumption in practice, smoothing methods use the neighbourhood information to remove outliers and enhance consistency in the resulting product.

Smoothing methods are an important complement to machine learning algorithms for image classification. Since these methods are mostly pixel-based, it is useful to complement them with post-processing smoothing to include spatial information in the result. For each pixel, machine learning and other statistical algorithms provide the probabilities of that pixel belonging to each of the classes. As a first step in obtaining a result, each pixel is assigned to the class whose probability is higher. After this step, smoothing methods use class probabilities to detect and correct outliers or misclassified pixels. SITS uses a Bayesian smoothing method, which provides the means to incorporate prior knowledge in data analysis. For more details on the smoothing procedure, please see the vignette "Post classification smoothing using Bayesian techniques in SITS".

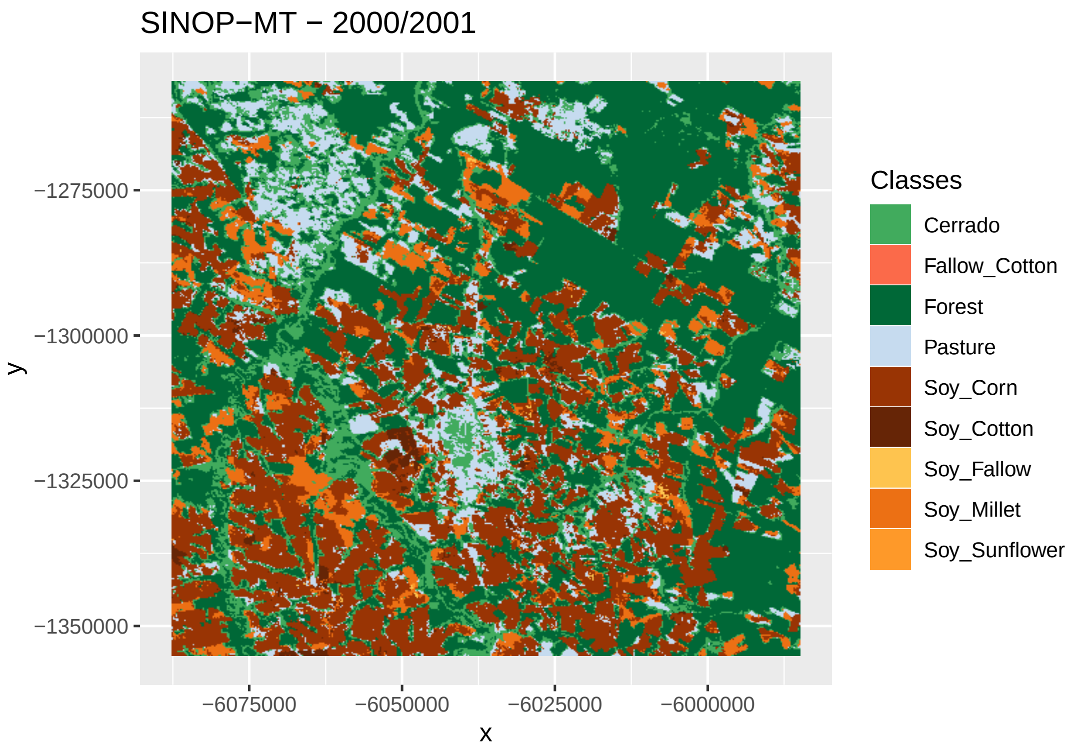


Figure 6: Image classified with XGBoost. The image coordinates (*meters*) shown at vertical and horizontal axis are in MODIS sinusoidal projection.

Doing post-processing using Bayesian smoothing in SITS is straightforward. The result of the `sits_classify` function applied to a data cube is set of more probability images, one per requested clasification interval. The next step is to apply the `sits_label_classification` function. By default, this function selects the most likely class for each pixel considering only the probabilities of each class for each pixel. To allow for Bayesian smooting, it suffices to include the `smoothing = bayesian` parameter. If desired, the `variance` parameter (associated to the hyperparameter σ^2_k described above) can control the degree of smoothness. The following example takes the previously produced classification output and applies a Bayesian smoothing.

```
# smooth the result with a bayesian filter
label_bayes <- sits_label_classification(probs_cube, smoothing = "bayesian")

# plot the smoothened image
sits_plot_raster(label_bayes, time = 1, title = "Sinop-smooth")
```

Final remarks

Current approaches to image time series analysis still use limited number of attributes. A common approach is deriving a small set of phenological parameters from vegetation indices, like beginning, peak, and length of growing season [Brown et al., 2013], [Kastens

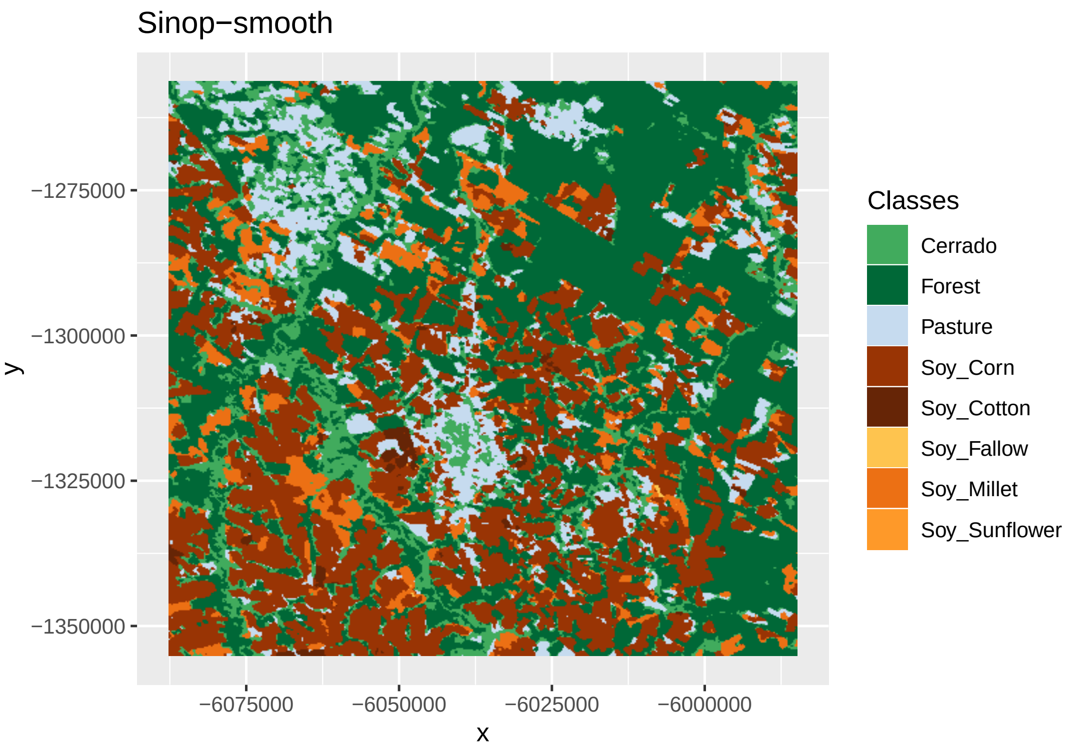


Figure 7: Image post-processed with Bayesian smoothing. The image coordinates (*meters*) shown at vertical and horizontal axis are in MODIS sinusoidal projection.

et al., 2017], [Estel et al., 2015], [Pelletier et al., 2016]. These phenological parameters are then fed in specialized classifiers such as TIMESAT [Jönsson and Eklundh, 2004]. These approaches do not use the power of advanced statistical learning techniques to work on high-dimensional spaces with big training data sets [James et al., 2013].

Package `sits` can use the full depth of satellite image time series to create larger dimensional spaces. We tested different methods of extracting attributes from time series data, including those reported by Pelletier et al. [2016] and Kastens et al. [2017]. Our conclusion is that part of the information in raw time series is lost after filtering. Thus, the method we developed uses all the data available in the time series samples. The idea is to have as many temporal attributes as possible, increasing the dimension of the classification space. Our experiments found out that modern statistical models such as support vector machines, and random forests perform better in high-dimensional spaces than in lower dimensional ones.

Acknowledgements

The authors would like to thank all the researchers that provided data samples used in the examples: Alexandre Coutinho, Julio Esquerdo and Joao Antunes (Brazilian Agricultural Research Agency, Brazil) who provided ground samples for “soybean-fallow”, “fallow-cotton”, “soybean-cotton”, “soybean-corn”, “soybean-millet”, “soybean-sunflower”, and "pasture" classes; Rodrigo Bergotti (National Institute for Space Research, Brazil) who provided samples for "cerrado" and "forest" classes; and Damien Arvor (Rennes University, France) who provided ground samples for "soybean-fallow" class.

This work was partially funded by the São Paulo Research Foundation (FAPESP) through eScience Program grant 2014/08398-6. We thank the Coordination for the Improvement of Higher Education Personnel (CAPES) and National Council for Scientific and Technological Development (CNPq) grants 312151/2014-4 (GC) and 140684/2016-6 (RS). We thank Ricardo Cartaxo, Lúbia Vinhas, and Karine Ferreira who provided insight and expertise to support this paper.

This work has also been supported by the International Climate Initiative of the Germany Federal Ministry for the Environment, Nature Conservation, Building and Nuclear Safety under Grant Agreement 17-III-084-Global-A-RESTORE+ ("RESTORE+: Addressing Landscape Restoration on Degraded Land in Indonesia and Brazil").

References

Marius Appel and Edzer Pebesma. On-demand processing of data cubes from satellite image collections with the gdalcubes library. *Data*, 4, 2019.

D. Arvor, M. Meirelles, V. Dubreuil, A. Bégué, and Y. E. Shimabukuro. Analyzing the agricultural transition in Mato Grosso, Brazil, using satellite-derived indices. *Applied Geography*, 32(2):702–713, 2012.

Clement Atzberger and Paul HC Eilers. Evaluating the effectiveness of smoothing algorithms in the absence of ground reference measurements. *International Journal of Remote Sensing*, 32(13):3689–3709, 2011.

Mariana Belgiu and Lucian Dragut. Random Forest in remote sensing: A review of applications and future directions. *ISPRS Journal of Photogrammetry and Remote Sensing*, 114:24–31, 2016.

J. Christopher Brown, Jude H. Kastens, Alexandre Camargo Coutinho, Daniel de Castro Victoria, and Christopher R. Bishop. Classifying multiyear agricultural land use data from Mato Grosso using time-series MODIS vegetation index data. *Remote Sensing of Environment*, 130:39–50, 2013.

Gilberto Câmara, Gilberto Queiroz, Lubia Vinhas, Karine Ferreira, Ricardo Cartaxo, Rolf Simoes, Eduardo Llapa, Luiz Assis, and Alber Sanchez. The e-sensing architecture for big earth observation data analysis. In *Big Data from Space (BiDS'17)*, pages 48–51, 2017.

Stephan Estel, Tobias Kuemmerle, Camilo Alcantara, Christian Levers, Alexander Prishchepov, and Patrick Hostert. Mapping farmland abandonment and recultivation across Europe using MODIS NDVI time series. *Remote Sensing of Environment*, 163: 312–325, 2015.

Gillian L Galford, John F Mustard, Jerry Melillo, Aline Gendrin, Carlos C Cerri, and Carlos E Cerri. Wavelet analysis of MODIS time series to detect expansion and intensification of row-crop agriculture in Brazil. *Remote sensing of environment*, 112(2): 576–587, 2008.

Cristina Gomez, Joanne C. White, and Michael A. Wulder. Optical remotely sensed time series data for land cover classification: A review. *{ISPRS} Journal of Photogrammetry and Remote Sensing*, 116:55 – 72, 2016. ISSN 0924-2716.

T. Hastie, R. Tibshirani, and Friedman J. *The Elements of Statistical Learning. Data Mining, Inference, and Prediction.* Springer, New York, 2009.

G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning: with Applications in R.* Springer, New York, EUA, 2013.

Per Jönsson and Lars Eklundh. TIMESAT—a program for analyzing time-series of satellite sensor data. *Computers & Geosciences*, 30(8):833 – 845, 2004. ISSN 0098-3004.

J. Kastens, J. Brown, A. Coutinho, C. Bishop, and J. Esquerdo. Soy moratorium impacts on soybean and deforestation dynamics in Mato Grosso, Brazil. *PLOS ONE*, 12(4): e0176168, 2017.

Robert E. Kennedy, Zhiqiang Yang, and Warren B. Cohen. Detecting trends in forest disturbance and recovery using yearly Landsat time series. *Remote Sensing of Environment*, 114(12):2897–2910, 2010.

T. Kohonen, M. R. Schroeder, and T. S. Huang. *Self-Organizing Maps*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 3rd edition, 2001.

Eric F Lambin, Helmut J Geist, and Erika Lepers. Dynamics of land-use and land-cover change in tropical regions. *Annual review of environment and resources*, 28(1):205–241, 2003.

Victor Maus, Gilberto Camara, Ricardo Cartaxo, Alber Sanchez, Fernando M Ramos, and Gilberto R de Queiroz. 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, 2016.

Aaron E. Maxwell, Timothy A. Warner, and Fang Fang. Implementation of machine-learning classification in remote sensing: an applied review. *International Journal of Remote Sensing*, 39(9):2784–2817, 2018. doi: 10.1080/01431161.2018.1433343.

G. Mountrakis, J. Im, and C. Ogole. Support vector machines in remote sensing: A review. *ISPRS Journal of Photogrammetry and Remote Sensing*, 66(3):247–259, 2011.

Valerie J. Pasquarella, Christopher E. Holden, Les Kaufman, and Curtis E. Woodcock. From imagery to ecology: leveraging time series of all available LANDSAT observations to map and monitor ecosystem state and dynamics. *Remote Sensing in Ecology and Conservation*, 2(3):152–170, 2016. ISSN 2056-3485. doi: 10.1002/rse2.24.

Charlotte Pelletier, Silvia Valero, Jordi Inglada, Nicolas Champion, and Gerard Dedieu. Assessing the robustness of Random Forests to map land cover with high resolution satellite image time series over large areas. *Remote Sensing of Environment*, 187: 156–168, 2016.

F. Petitjean, J. Inglada, and P. Gancarski. Satellite image time series analysis under time warping. *IEEE Transactions on Geoscience and Remote Sensing*, 50(8):3081–3095, 2012. ISSN 0196-2892. doi: 10.1109/TGRS.2011.2179050.

Konrad Schindler. An overview and comparison of smooth labeling methods for land-cover classification. *IEEE transactions on geoscience and remote sensing*, 50(11):4534–4545, 2012.

Jan Verbesselt, Rob Hyndman, Glenn Newnham, and Darius Culvenor. Detecting trend and seasonal changes in satellite image time series. *Remote sensing of Environment*, 114(1):106–115, 2010.

Lubia Vinhas, Gilberto Ribeiro, Karine Reis Ferreira, and Gilberto Camara. Web Services for big Earth observation data. In *Proceedings of the 17th Brazilian Symposium on GeoInformatics*, pages 26–35, Campos do Jordão, SP, Brazil, 2016. INPE.

Hadley Wickham and Garrett Grolemund. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data.* O'Reilly Media, Inc., 2017.

Achim Zeileis and Gabor Grothendieck. zoo: S3 infrastructure for regular and irregular time series. *Journal of Statistical Software*, 14(6):1–27, 2005.