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*
Felipe Souza *National Institute for Space Research (INPE), Brazil*
Lorena Santos *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*
Karine Ferreira *National Institute for Space Research (INPE), Brazil*
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**. The package support use of machine learning techniques for classification image time series obtained from data cubes. Methods available 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 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 previously 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.

Workflow

The main aim of **sits** is to support land cover and land change classification of image data cubes using machine learning methods. The basic workflow is:

1. Create a data cube using image collections available in the cloud or in local machines.
2. Extract time series from the data cube which are used as training data.
3. Perform quality control and filtering on the samples.
4. Train a machine learning model using the extracted samples.
5. Classify the data cube using the trained model.
6. Post-process the classified images.

Handling Data Cubes in sits

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".

Using STAC 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 information provided by STAC (Spatio-temporal Access Catalogue).

Currently, SITS supports data cubes available in the following cloud services:

1. Sentinel-2/2A level 2A images in AWS.
2. Collections of Sentinel, Landsat and CBERS images in the Brazil Data Cube (BDC).
3. Collections available in Digital Earth Africa.
4. Data cubes produced by the "gdalcubes" package.
5. Local image collections.

The user can define a data cube by selecting a collection in a cloud service and then defining a space-time extent. For example, the following code will define a data cube of Sentinel-2/2A images using AWS.

```
s2_cube <- sits_cube(
    type = "S2_L2A_AWS",
    name = "T20LKP_2018_2019",
    satellite = "SENTINEL-2",
    sensor = "MSI",
    tiles = "20LKP",
    s2_aws_resolution = "20m",
    start_date = as.Date("2018-08-01"),
    end_date = as.Date("2019-07-31")
)
```

In the above example, the user has selected the "Sentinel-2 Level 2" collection in the AWS cloud services. The geographical area of the data cube is defined by the tile "20LKP", and the temporal extent by a start and end date. Access to other cloud services works in similar ways.

Users can derive data cubes from ARD data which have pre-defined temporal resolutions. For example, a user may want to define the best Sentinel-2 pixel in a one month period, as shown below. This can be done in SITS by the `sits_regularize` which calls the "gdalcubes" package. For details in gdalcubes, please see https://github.com/appelmar/gdalcubes.

```r
gc_cube <- sits_regularize(cube         = s2_cube,
                           name         = "T20LKP_2018_2019_1M",
                           path_db      = "/my/path/cube.db",
                           path_images = "/my/path/images/",
                           period       = "P1M",
                           agg_method  = "median",
                           resampling  = "bilinear"
)
```

Defining a data cube using files

To define a data cube using plain files (without STAC information), all image files should have the same spatial resolution and same projection. Each file contains a single image band for a single date. Since raster files in popular formats (e.g., GeoTiff and Jpeg2000) do not include time information, the name of each file needs to include date and band information. Timeline and bands are deduced from filenames. For example, "CBERS-4_AWFI_B13_2018-02-02.tif" is a valid name. The user has to provide parsing information to allow **sits** to extract the band and the date. In the example above, the parsing info is c("X1", "X2", "band", "date) and the delimiter is"_".

```r
# Create a cube based on a stack of CBERS data
data_dir <- system.file("extdata/raster/cbers", package = "sits")

# files are named using the convention
# "CBERS-4_AWFI_B13_2018-02-02.tif"
cbers_cube <- sits_cube(
      source = "LOCAL",
      name = "022024",
      satellite = "CBERS-4",
      sensor = "AWFI",
      data_dir = data_dir,
      delim = "_",
      parse_info = c("X1", "X2", "band", "date")
)
# print the timeline of the cube
sits_timeline(cbers_cube)
```

```
##  [1] "2018-02-02" "2018-02-18" "2018-03-06" "2018-03-22" "2018-04-07"
##  [6] "2018-04-23" "2018-05-09" "2018-05-25" "2018-06-10" "2018-06-26"
## [11] "2018-07-12" "2018-07-28" "2018-08-13" "2018-08-29"
```

```r
# print the bounding box of the cube
sits_bbox(cbers_cube)
```

```
##     xmin    xmax    ymin    ymax
## 5794837 5798037 9773148 9776348
```

Handling satellite image time series in sits

Data structure

Training a machine learning model in **sits** requires a set of time series, 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 1,218 labelled samples of land cover in Mato Grosso state of Brazil, with four classes: "Forest", "Cerrado", "Pasture", "Soybean-Corn".

```
# data set of samples
data(samples_modis_4bands)
samples_modis_4bands[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     -55.2   -10.8  2013-09-14 2014-08-29 Pasture MOD13Q1 <tibble [23 x 5]>
## 2     -57.8    -9.76 2006-09-14 2007-08-29 Pasture MOD13Q1 <tibble [23 x 5]>
## 3     -51.9   -13.4  2014-09-14 2015-08-29 Pasture MOD13Q1 <tibble [23 x 5]>
```

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 "Pasture", at location (−55.1852, −10.8378), and is valid for the period (2013-09-14, 2014-08-29).

```
# print the first time series records of the first sample
sits_time_series(samples_modis_4bands[1,])[1:3,]
```

```
## # A tibble: 3 x 5
##   Index       NDVI   EVI   NIR   MIR
##   <date>     <dbl> <dbl> <dbl> <dbl>
## 1 2013-09-14 0.388 0.253 0.316 0.307
## 2 2013-09-30 0.491 0.277 0.275 0.170
## 3 2013-10-16 0.527 0.318 0.286 0.205
```

Obtaining time series data

To get a time series in **sits**, one has to first create a data cube first. 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 `plot()`.

```
library(sits)
data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
modis_cube <- sits_cube(
    source = "LOCAL",
    name = "sinop-2014",
    satellite = "TERRA",
    sensor = "MODIS",
    data_dir = data_dir,
    delim = "_",
    parse_info = c("X1", "X2", "band", "date")
)

# obtain a set of locations defined by a CSV file
csv_raster_file <- system.file("extdata/samples/samples_sinop_crop.csv",
                               package = "sits")

# retrieve the points from the data cube
points <- sits_get_data(modis_cube, file = csv_raster_file)
```

```
## All points have been retrieved
```

```
# plot the first point
plot(points[1,])
```

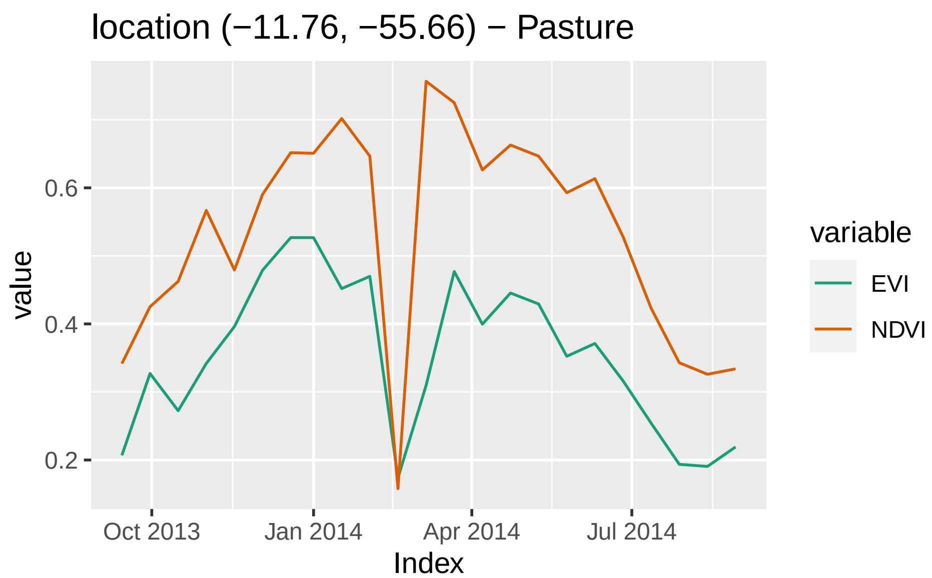

Figure 1: A one year time series of MOD13Q1 data for bands NDVI and EVI

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`)

The SITS package uses a common interface to all filter functions with the `sits_filter`. The function has two parameters: 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.

```
# Take a NDVI time series, apply Whittaker filter and plot the series
point_ndvi <- sits_select(point_mt_6bands, bands = "NDVI")
point_whit <- sits_whittaker(point_ndvi, lambda = 5.0)
# merge with original data and plot the original and the filtered data
point_whit %>%
  sits_merge(point_ndvi) %>%
  plot()
```

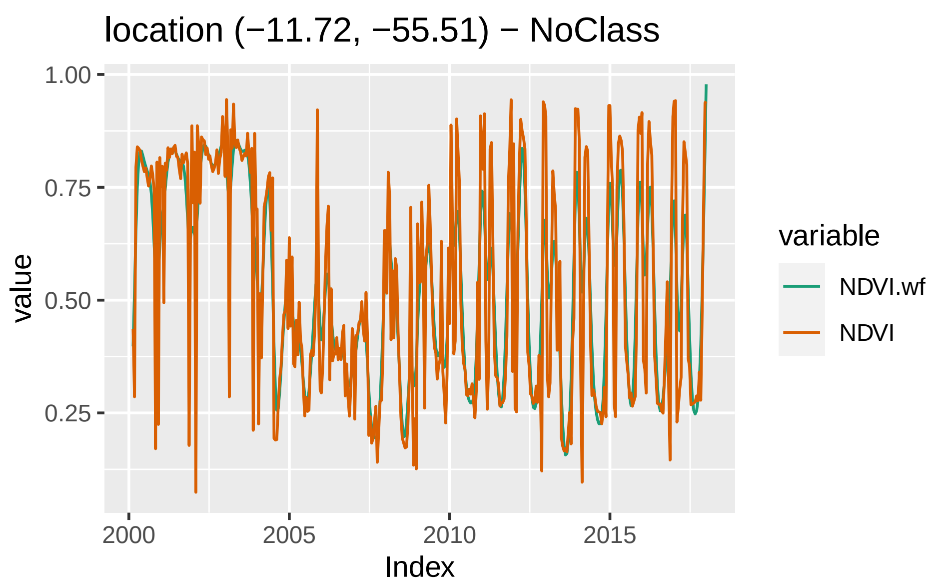

Figure 2: Whittaker smoother filter applied on one-year NDVI time series. The example uses default $\lambda = 3$ parameter.

Clustering for sample quality control using self-organizing maps

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'.

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 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 `plot`.

```r
#select the data for classification

# Train a machine learning model using Random Forest
rfor_model <- sits_train(data = samples_modis_4bands,
                         ml_method = sits_rfor(num_trees = 1000))

# get a point to be classified
point_4bands <- sits_select(point_mt_6bands,
                            bands = c("NDVI", "EVI", "NIR", "MIR"))

# Classify using random forest model and plot the result
class.tb <- sits_classify(point_4bands, rfor_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
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`)

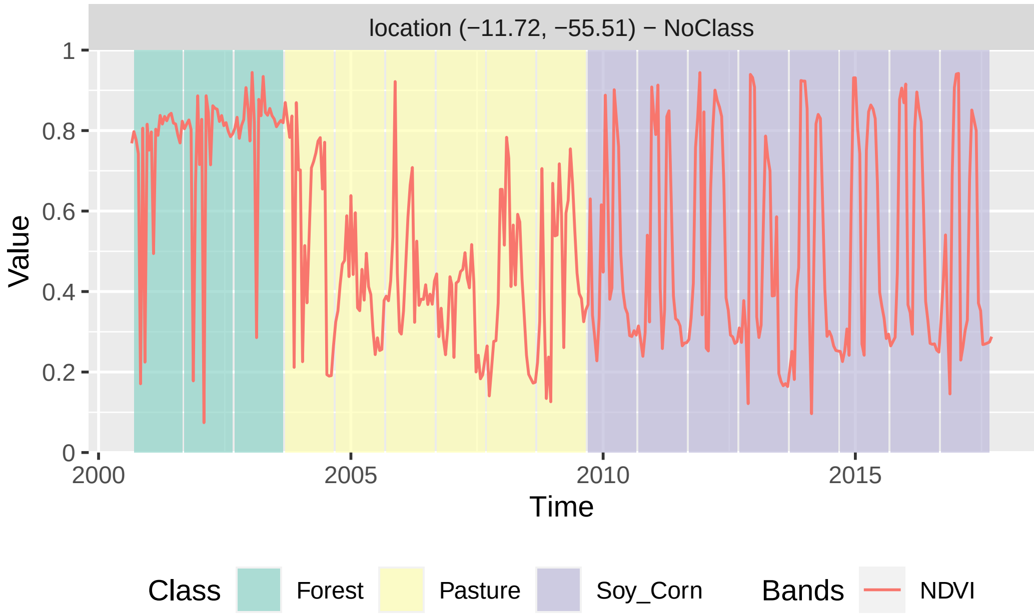

Figure 3: Random forest classification of a 16 years time series. The location (latitude, longitude) shown at the top of the graph is in geographic coordinate system (WGS84 *datum*).

- Deep learning (DL) using multi-layer perceptrons (`sits_deeplearning`)
- 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`).
- DL using a combination of long-short term memory (LSTM) and 1D CNN (`sits_LSTM_FCN`)

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
# Random Forest machine learning method using default parameters
acc <- sits_kfold_validate(cerrado_2classes,
                           folds = 5,
                           ml_method = sits_rfor(num_trees = 1000))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Cerrado Pasture
##    Cerrado     394      12
##    Pasture       6     334
##
##           Accuracy : 0.9759
##             95% CI : (0.9621, 0.9856)
##
##              Kappa : 0.9514
##
##  Prod Acc  Cerrado : 0.9850
##  Prod Acc  Pasture : 0.9653
##  User Acc  Cerrado : 0.9704
##  User Acc  Pasture : 0.9824
##
```

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 with data cubes built with the 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 Sentinel-2 tile at 20 meter resolution (5490*5490 pixels), with six bands and 66 time instances covering one year of data, using SVM with a training data set of about 10,000 samples, takes about 3 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.

```
# Retrieve the set of samples for the Mato Grosso region
# Select the data for classification
samples_2bands  <- sits_select(samples_modis_4bands,
                               bands = c("NDVI", "EVI"))

# build a machine learning model for this area
svm_model <- sits_train(samples_2bands, sits_svm())

# create a data cube to be classified
# Cube is composed of MOD13Q1 images from the Sinop region in Mato Grosso (Brazil)
data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
sinop <- sits_cube(
    source = "LOCAL",
    name = "sinop-2014",
    satellite = "TERRA",
    sensor = "MODIS",
    data_dir = data_dir,
    delim = "_",
    parse_info = c("X1", "X2", "band", "date")
```

```
)
# Classify the raster cube, generating a probability file
probs_cube <- sits_classify(sinop,
                            ml_model = svm_model,
                            output_dir = tempdir(),
                            memsize = 16,
                            multicores = 4,
                            verbose = FALSE)

# plot the probabilities cubes
plot(probs_cube)
```

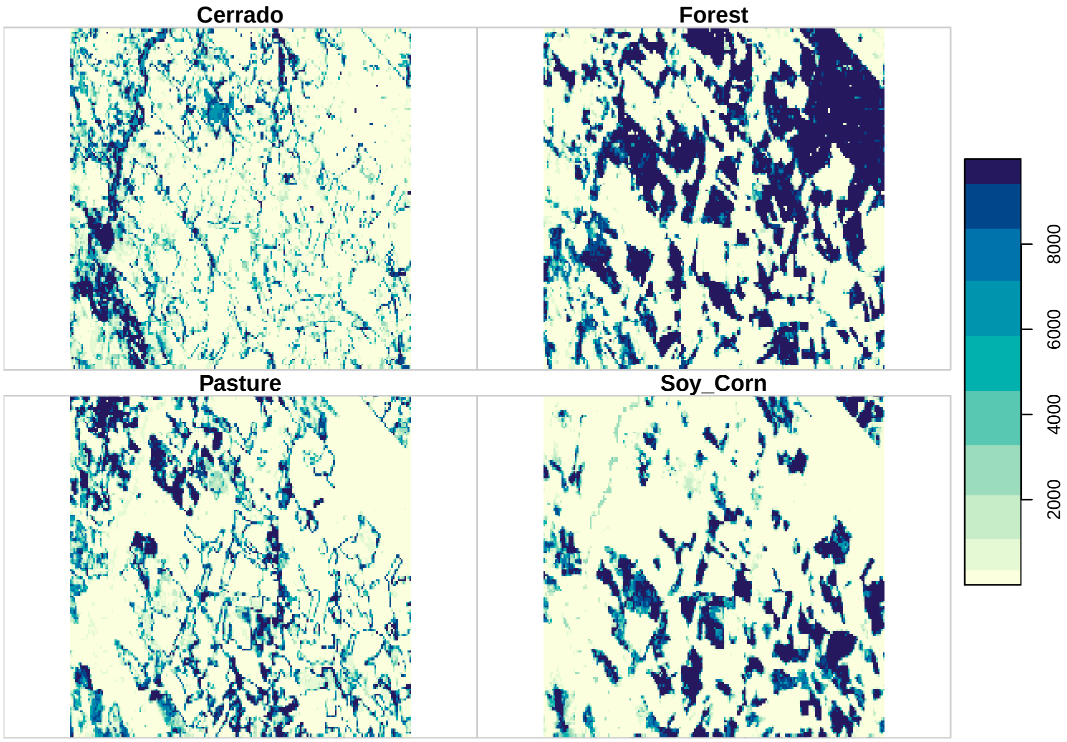

Figure 4: Class probabilities for each pixel

Smoothing and Labelling 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".

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_smooth` function. By default, this function selects the most likely class for each pixel considering a Bayesian estimator that takes into account the neighbours. The following example takes the previously produced classification output and applies a Bayesian smoothing.

```
# smooth the result with a bayesian filter
sinop_bayes <- sits_smooth(probs_cube, output_dir = tempdir())
# label the resulting image
label_bayes <- sits_label_classification(sinop_bayes, output_dir = tempdir())
```

```
# plot the image
plot(label_bayes)
```

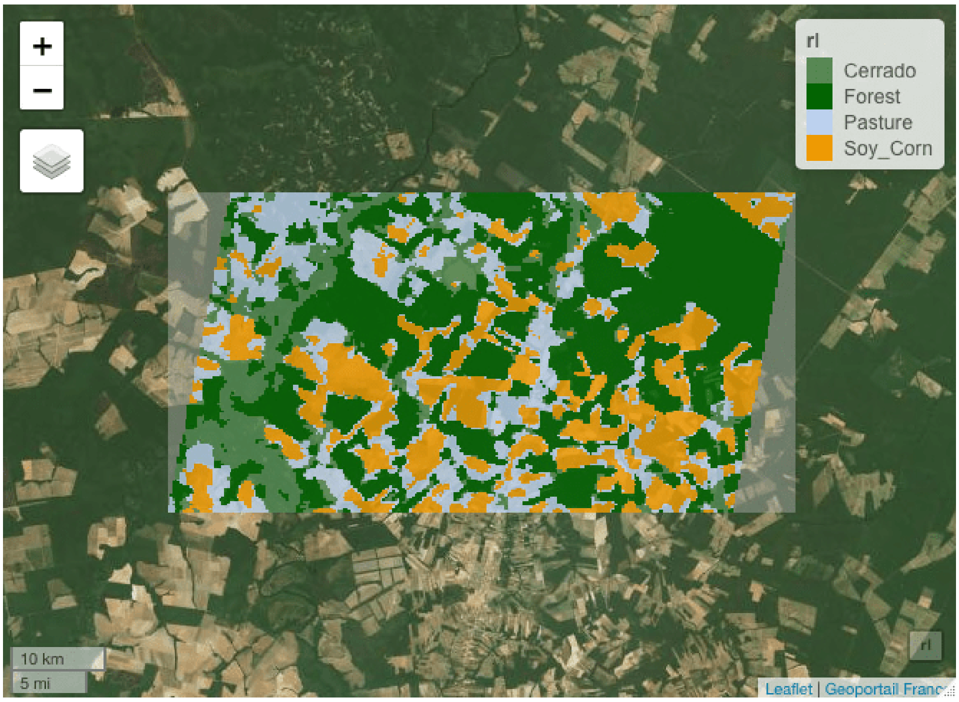

Figure 5: Classified image post-processed with Bayesian smoothing

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 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.

Additional information

For more information, please see the vignettes

- "SITS: Data analysis and machine learning for data cubes using satellite image time series"
- "Accessing time series information in SITS"
- "Clustering of satellite image time series with SITS"
- "Satellite image time series filtering with SITS"
- "Time series classification using machine learning"
- "Post classification smoothing using Bayesian techniques in SITS"

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 and Lúbia Vinhas 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.

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.

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