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

Rolf Simoes *National Institute for Space Research (INPE), Brazil*
Gilberto Camara *National Institute for Space Research (INPE), Brazil*
Alexandre Carvalho *Institute for Applied Economics Research (IPEA), Brazil*
Victor Maus *International Institute for Applied System Analysis (IIASA), Austria*
Pedro R. Andrade *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`. Package `sits` provides support on how to use statistical learning techniques with image time series. These methods include linear and quadratic discrimination analysis, support vector machines, random forests, and 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. The time series are 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].

In this paper, we present `sits`, an open source R package for satellite image time series analysis. It provides support on how to use statistical learning techniques with image time series. In a broad sense, statistical learning refers to a class of algorithms for classification and regression analysis [Hastie et al., 2009]. These methods include linear and quadratic discrimination analysis, support vector machines, random forests, and neural networks. In a typical classification problem, we have measures that capture class attributes. Based on these measures, referred as training data, one's task is to select a predictive model that allows inferring classes of a larger data set.

In what follows, we describe the main characteristics of the `sits` package. The first part describes its basic data structures and the tools used for visualization and data exploration. Then we describe data acquisition from external sources, with an emphasis on the WTSS (an acronym for Web Time Series Service) [Queiroz et al., 2015]. The next sections describe filtering and clustering techniques. We then discuss machine learning techniques for satellite image time series data and how to apply them to image time series. Finally, we present validation methods.

Data Handling and Visualisation Basics in `sits`

The basic data unit of `sits` package is the "`sits` tibble", which is a way of organizing a set of time series data with associated spatial information. In R, a `tibble` differs from the traditional `data.frame`, insofar as a `tibble` can contain lists embedded as column arguments. Tibbles are part of the `tidyverse`, a collection of R packages designed to work together in data manipulation [Wickham and Grolemund, 2017]. The `sits` makes extensive use of the `tidyverse`. For a better explanation of how the "`sits` tibble" works, we will read a data set containing 2,115 labelled samples of land cover in Mato Grosso state of Brazil. This state has 903,357km^2 of extension, being the third largest state of Brazil. It includes three of Brazil's biomes: Amazonia, Cerrado, and Pantanal. 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, we selected 2,115 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_9classes)
samples_mt_9classes[1:3,]
```

```
## # A tibble: 3 x 7
##   longitude latitude start_date end_date   label   coverage 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~
## 2    -57.8    -9.76 2006-09-14 2007-08-29 Pasture MOD13Q1  <tibble [23 x~
## 3    -51.9   -13.4  2014-09-14 2015-08-29 Pasture MOD13Q1  <tibble [23 x~
```

A `sits` tibble contains data and metadata. The first six columns contain the metadata: spatial and temporal location, label assigned to the sample, and coverage 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, −10.8387), and is considered valid for the period (2013-09-14, 2014-08-29). 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
samples_mt_9classes$time_series[[1]][1:3,]
```

```
## # A tibble: 3 x 7
##   Index        mir   blue   nir    red   evi  ndvi
##   <date>     <dbl>  <dbl> <dbl>  <dbl> <dbl> <dbl>
## 1 2013-09-14 0.307 0.0538 0.316 0.139  0.253 0.388
## 2 2013-09-30 0.170 0.0277 0.275 0.0939 0.277 0.491
## 3 2013-10-16 0.205 0.0354 0.286 0.0884 0.318 0.527
```

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.

```
sits_labels(samples_mt_9classes)
```

```
## # A tibble: 9 x 3
##   label         count   freq
##   <chr>         <int>  <dbl>
## 1 Cerrado         400 0.189
## 2 Fallow_Cotton    34 0.0161
## 3 Forest          138 0.0652
## 4 Pasture         370 0.175
## 5 Soy_Corn        398 0.188
## 6 Soy_Cotton      399 0.189
## 7 Soy_Fallow       88 0.0416
## 8 Soy_Millet      235 0.111
## 9 Soy_Sunflower    53 0.0251
```

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.

```
# select NDVI band
samples_ndvi.tb <- sits_select_bands(samples_mt_9classes, ndvi)
# select only samples with Cerrado label
samples_cerrado.tb <-
    dplyr::filter(samples_ndvi.tb, label == "Cerrado")
# plot the first 15 samples (different dates for a single point)
sits_plot(samples_cerrado.tb[1:15,])
```

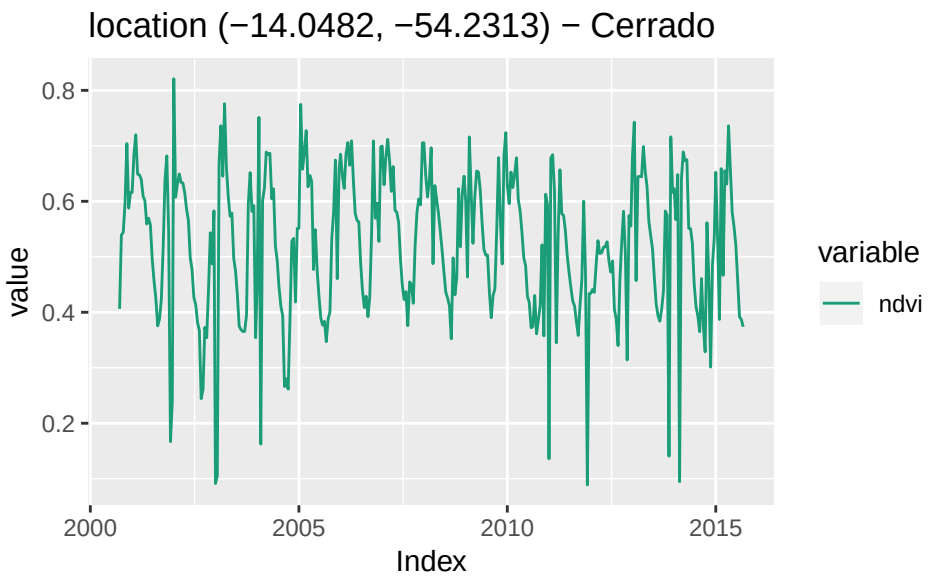


Figure 1: Plot of the first 15 'Cerrado' samples from data set `samples_mt_9classes` (different dates for the same point location).

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.

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

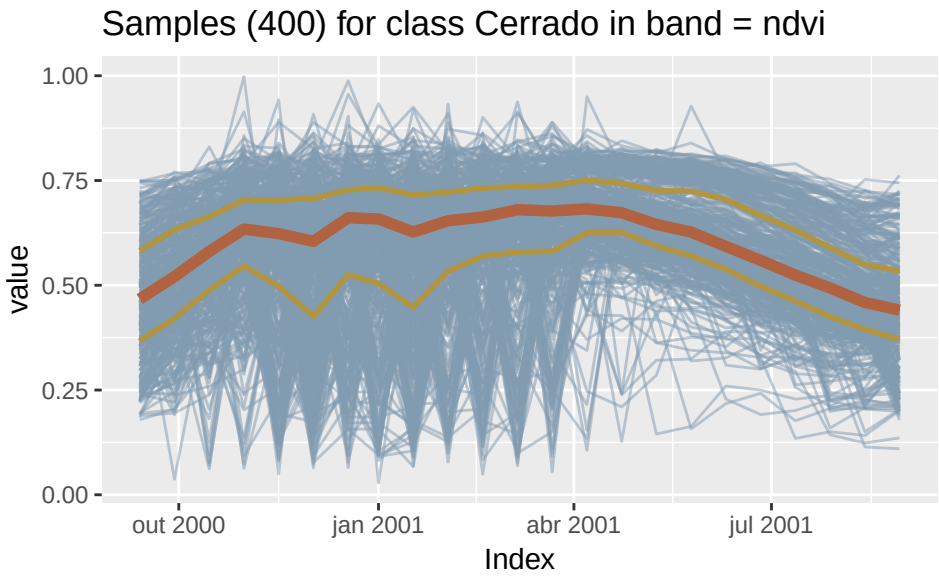


Figure 2: Plot of all Cerrado samples from data set `samples_mt_9classes`.

Importing Data into `sits`

Usually, samples are provided by experts that take *in-loco* field observations or recognize land classes through high resolution images. In any case, we need access to a data source to fetch time series data regarding a spatiotemporal location of interest. The process of importing data samples is discussed in this section.

Package `sits` allows different methods of data input, including: (a) obtain data from a WTSS (Web Series Time Service); (b) obtain data from the SATVEG service developed by EMBRAPA (Brazil's Agriculture Research Agency); (c) read data stored in a time series in the ZOO format [Zeileis and Grothendieck, 2005]; (d) read a time series from a RasterBrick [Hijmans, 2015]. Option (d) will be described in the section were we present raster processing. WTSS is a light-weight service that retrieves time series data for selected locations and periods [Vinhas et al., 2016]. SATVEG service provides NDVI and EVI time series vegetation indices from MODIS image for the whole Brazilian territory [EMBRAPA, 2014]. Function `sits_services()` provides information on the coverages available in the servers.

```
sits_services()
```

```
## Service: "WTSS-INPE"
##    Coverage: "MOD13Q1"
```

```
##       Bands: "mir", "blue", "nir", "red", "evi", "ndvi"
##    Coverage: "MOD13Q1_M"
##       Bands: "quality", "reliability"
## Service: "SATVEG"
##    Coverage: "terra"
##       Bands: "ndvi", "evi"
##    Coverage: "aqua"
##       Bands: "ndvi", "evi"
##    Coverage: "comb"
##       Bands: "ndvi", "evi"
```

After finding out which coverages are available at the different time series services, one may request specific information on each coverage by using `sits_coverage()`. This function lists the contents of the data set, including source, bands, spatial extent and resolution, time range, and temporal resolution. This information is returned as a tibble.

```
# get information about a specific coverage from WTSS
coverage_wtss <- sits_coverage(service = "WTSS-INPE",
                               name    = "MOD13Q1")

coverage_wtss %>% dplyr::select(xmin, xmax, ymin, ymax, timeline)
```

```
## # A tibble: 1 x 5
##    xmin  xmax  ymin  ymax timeline
##   <dbl> <dbl> <dbl> <dbl> <list>
## 1 -81.2 -30.0 -40.0 10.00 <list [1]>
```

The user can request one or more time series points from a coverage by using `sits_get_data()`. This function provides a general means of access to image time series. In its simplest fashion, the user provides the latitude and longitude of the desired location, the product and coverage names, 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
series.tb <- sits_get_data(longitude = -55.57320,
                           latitude  = -11.50566,
                           coverage  = coverage_wtss,
                           bands     = c("ndvi", "evi"))
sits_plot(series.tb)
```

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.

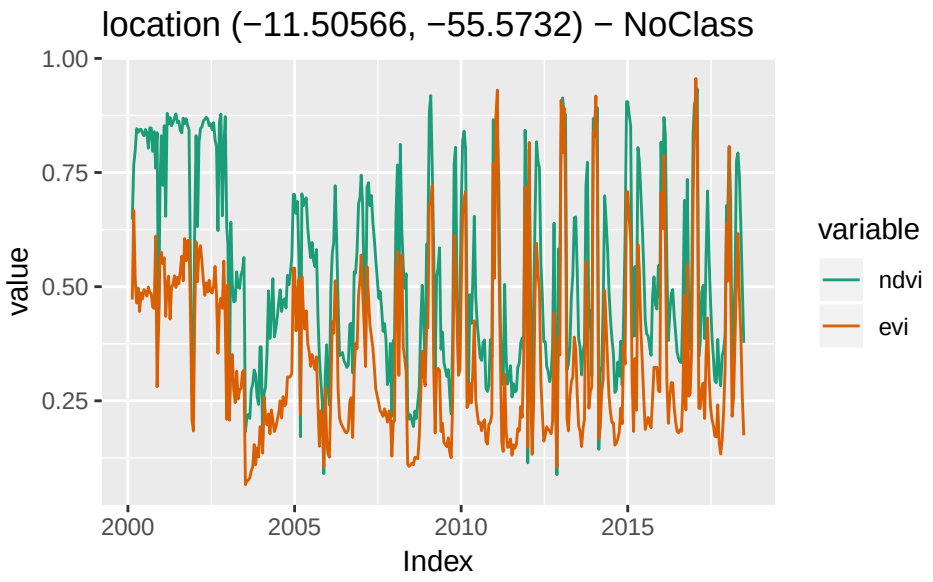


Figure 3: NDVI and EVI time series fetched from WTSS service.

After importing the samples time series, it is useful to explore the data to see how it is structured and look for its inter-class separability. For example, we can note in the figure above the variability of 400 time series collected from different years and locations. The scattering behavior is intrinsic to remote sensing data. Atmospheric noise, sun angle, interferences on observations or different equipment specifications, as well as the very nature of the climate-land dynamics can be sources of such variability [Atkinson et al., 2012]. One helpful technique to explore such properties is *cluster analysis*. In the following section we present a clustering technique supported by `sits`.

Clustering satellite image time series

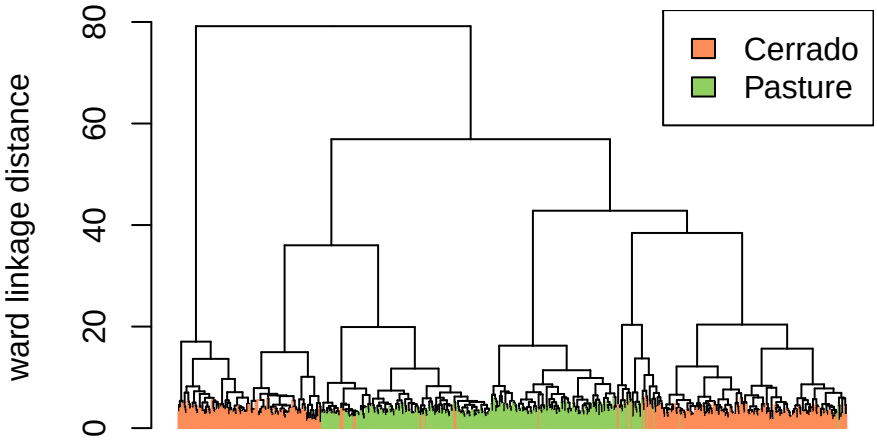
Cluster analysis has been used for many purposes in satellite image time series literature ranging from unsupervised classification [Petitjean et al., 2011], and pattern detection [Romani et al., 2011]. Here, we are interested in the second use of clustering, using it as a way to improve training data to feed machine learning classification models. In this regard, cluster analysis can assist the identification of structural *time series patterns* and anomalous samples [Romani et al., 2011], [Chandola et al., 2009]. `sits` provides support for the Agglomerative Hierarchical Clustering (AHC).

Hierarchical clustering is a family of methods that groups elements using a distance function to associate a real value to a pair of elements. From this distance measure, we can compute the dissimilarity between any two elements from a data set. Depending on the distance functions and linkage criteria, the algorithm decides which two clusters are merged at each iteration. AHC approach is suitable for the purposes of samples data exploration due to its visualization power and ease of use [Keogh et al., 2003]. Moreover, AHC does not require a predefined number of clusters as an initial parameter. This is an

important feature in satellite image time series clustering since defining the number of clusters present in a set of multi-attribute time series is not straightforward [Aghabozorgi et al., 2015].

The main result of AHC method is a *dendrogram*. It is the ultrametric relation formed by the successive merges in the hierarchical process that can be represented by a tree. Dendrograms are quite useful to decide the number of clusters to partition the data. It shows the height where each merging happens, which corresponds to the minimum distance between two clusters defined by a *linkage criterion*. The most common linkage criteria are: *single-linkage*, *complete-linkage*, *average-linkage*, and *Ward-linkage*. Complete-linkage prioritizes the within-cluster dissimilarities, producing clusters with shorter distance samples. Complete-linkage clustering can be sensitive to outliers, which can increase the resulting intracluster data variance. As an alternative, Ward proposes a criteria to minimize the data variance by means of either *sum-of-squares* or *sum-of-squares-error* [Ward, 1963]. Ward's intuition is that clusters of multivariate observations, such as time series, should be approximately elliptical in shape [Hennig, 2015]. In `sits`, a dendrogram can be generated by `sits_dendrogram()`. The following codes illustrate how to create, visualize, and cut a dendrogram (for details, see `?sits_dendrogram()`).

```
# take a set of patterns for 2 classes
# create a dendrogram object with default clustering parameters
dendro <- sits_dendrogram(cerrado_2classes)
# plot the resulting dendrogram
sits_plot_dendrogram(cerrado_2classes, dendro)
```



After creating a dendrogram, an important question emerges: *where to cut the dendrogram?* The answer depends on what are the purposes of the cluster analysis [Hennig, 2015]. If one is interested in an unsupervised classification, it is common to use *internal validity indices*, such as silhouettes [Rousseeuw, 1987], to help determine the best number of clusters. However, if one is interested in understanding the structure of a labeled data set, or in the identifying sample anomalies, as we are here, one can use *external validity indices* to assist the semi-supervised procedure in order to achieve the optimal correspondence between clusters and classes partitions. In this regard, we need to balance two objectives: get clusters as large as possible, and get clusters as homogeneous as possible with respect to their known classes. To help this process, `sits` provides `sits_dendro_bestcut()` function that computes an external validity index *Adjusted Rand Index* (ARI) for a series of different number of generated clusters. This function returns the height where the cut of the dendrogram maximizes the index.

```
# search for the best height to cut the dendrogram
sits_dendro_bestcut(cerrado_2classes, dendro)
```

```
##        k   height
##  6.00000 20.39655
```

In this example, the height optimizes the ARI and generates 6 clusters. The ARI considers any pair of distinct samples and computes the following counts: (a) the number of distinct pairs whose samples have the same label and are in the same cluster; (b) the number of distinct pairs whose samples have the same label and are in different clusters; (c) the number of distinct pairs whose samples have different labels and are in the same cluster; and (d) the number of distinct pairs whose samples have the different labels and are in different clusters. Here, a and d consist in all agreements, and b and c all disagreements. The ARI is obtained by:

$$ARI = \frac{a+d-E}{a+d+b+c-E},$$

where E is the expected agreement, a random chance correction calculated by

$$E = (a+b)(b+c)+(c+d)(b+d).$$

Unlike other validity indexes such as Jaccard ($J = a/(a+b+c)$), Fowlkes-Mallows ($FM = a/(a^2+a(b+c)+bc)^{1/2}$), and Rand (the same as ARI without the E adjustment) indices, ARI is more appropriate either when the number of clusters is outweighed by the number of labels (and *vice versa*) or when the amount of samples in labels and clusters is imbalanced [Hubert and Arabie, 1985], which is usually the case.

```
# create 6 clusters by cutting the dendrogram at
# the linkage distance 20.39655
clusters.tb <- sits_cluster(cerrado_2classes, dendro, k = 6)
# 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
```

Note in this example that almost all clusters has a predominance of either "Cerrado" or "Pasture" classes with the exception of cluster 3. The contingency table plotted by `sits_cluster_frequency()` shows how the samples are distributed across the clusters and helps to identify two kinds of confusions. The first is relative to those small amount of samples in clusters dominated by another class (*e.g.* clusters 1, 2, 4, 5, and 6), while the second is relative to those samples in non-dominated clusters (*e.g.* cluster 3). These confusions can be an indication of samples with poor quality, an inadequacy of selected parameters for cluster analysis, or even a natural confusion due to the inherent variability of the land classes.

In an case, it is possible to remove outliers using `sits_cluster_clean()` or `sits_cluster_remove()`. The first one removes all those minority samples that do not reach a minimum percentage close to 0%, whereas the second removes an entire cluster if its dominant class does not reach a minimum percentage, close to 100%. The example below illustrates the second approach.

```
# clear those samples with a high confusion rate in a cluster
# (those clusters which majority class does not reach 90% of
# samples in that cluster)
cleaned.tb <- sits_cluster_remove(clusters.tb, min_perc = 0.9)
# show clusters samples frequency
sits_cluster_frequency(cleaned.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
```

Along the process of cluster analysis, it may be a good practice to measure the correspondence between clusters and labels partitions through computation of external validity indices. These metrics help comparing different procedures and assist decision-making. Function `sits_cluster_validity()` provides a way to compute some external validation indices other than ARI. Moreover, these indices capture the samples structure by deriving metrics from its partitions [Hubert and Arabie, 1985].

Filtering techniques

Satellite image time series generally is contaminated by atmospheric influence, geolocation error, and directional effects [Lambin and Linderman, 2006]. Inter-annual climate variability also changes the phenological cycles of the vegetation, resulting in time series

whose periods and intensities do not match on an year to year basis. To make the best use of available satellite data archives, methods for satellite image time series analysis need to deal with *noisy* and *non-homogeneous* data sets. In this section, we discuss filtering techniques to improve time series data that present missing values or noise.

The literature on satellite image time series have several applications of filtering to correct or smooth vegetation index data. The `sits` have support for Savitzky–Golay (`sits_sgolay()`), Whitaker (`sits_whittaker()`), envelope (`sits_envelope()`) and, the "cloud filter" (`sits_cloud_filter()`) filters. The first two filters are commonly used in the literature, while the remaining two are adapted from other methods and, for our knowledge, its use has not been reported in the literature.

Various somewhat conflicting results have been expressed in relation to the time series filtering techniques for phenology applications. For example, in an investigation of phenological parameter estimation, Atkinson et al. [2012] found that the Whittaker and Fourier transform approaches were preferable to the double logistic and asymmetric Gaussian models. They applied the filters to preprocess MERIS NDVI time series for estimating phenological parameters in India. Comparing the same filters as in the previous work, Shao et al. [2016] found that only Fourier transform and Whittaker techniques improved interclass separability for crop classes and significantly improved overall classification accuracy. The authors used MODIS NDVI time series from the Great Lakes region in North America. Zhou et al. [2016] found that asymmetric Gaussian model outperforms other filters over high latitude boreal biomes, while the Savitzky-Golay model gives the best reconstruction performance in tropical evergreen broadleaf forests. In the remaining biomes, Whittaker gives superior results. The authors compare all previous mentioned filters plus Savitzky-Golay method for noise removal in MODIS NDVI data from sites spread worldwide in different climatological conditions. Many other techniques can be found in applications of satellite image time series such as curve fitting [Bradley et al., 2007], wavelet decomposition [Sakamoto et al., 2005], mean-value iteration, ARMD3-ARMA5, and 4253H [Hird and McDermid, 2009]. Therefore, any comparative analysis of smoothing algorithms depends on the adopted performance measurement.

One of the main uses of time series filtering is to reduce the noise and miss data produced by clouds in tropical areas. The following examples use data produced by the PRODES project [INPE, 2017], which detects deforestation in the Brazilian Amazon rain forest through visual interpretation. `sits` provides 617 samples from a region corresponding to the standard Landsat Path/Row 226/064. This is an area in the East of the Brazilian Pará state. It was chosen because of its huge cloud coverage from November to March, which is a significant factor in degrading time series quality. Its NDVI and EVI time series were extracted from a combination of MOD13Q1 and Landsat8 images (to best visualize the effects of each filter, we selected only NDVI time series).

Savitzky–Golay filter

The Savitzky-Golay filter works by fitting a successive array of $2n+1$ adjacent data points with a d-degree polynomial through linear least squares. The central point i of the window array assumes the value of the interpolated polynomial. An equivalent and

much faster solution than this convolution procedure is given by the closed expression

$$\hat{x}_i = \sum_{j=-n}^{n} C_j \, x_{i+j},$$

where $\hat{x}$ is the the filtered time series, C_j are the Savitzky-Golay smoothing coefficients, and x is the original time series.

The coefficients C_j depend uniquely on the polynomial degree (d) and the length of the window data points (given by parameter n). If $d = 0$, the coefficients are constants $C_j = 1/(2n+1)$ and the Savitzky-Golay filter will be equivalent to moving average filter. When the time series is equally spaced, the coefficients have analytical solution. According to Madden [1978], for $d \in [2,3]$ each C_j smoothing coefficients can be obtained by

$$C_j = \frac{3(3n^2+3n-1-5j^2)}{(2n+3)(2n+1)(2n-1)}.$$

In general, the Savitzky-Golay filter produces smoother results for a larger value of n and/or a smaller value of d [Chen et al., 2004]. The optimal value for these two parameters can vary from case to case. For example, Zhou et al. [2016] set $d = 2$ and $n = 10$. Hird and McDermid [2009] tests the filter for $n \in [5,6,7]$ using quadratic polynomial.

`sits` Savitzky-Golay function The following example shows the effect of Savitsky-Golay filter on the original time series.

```
# Take NDVI band of the first sample data set
point.tb <- sits_select_bands(prodes_226_064[1,], ndvi)
# apply SavitzkyGolay filter
point_sg.tb <- sits_sgolay(point.tb)
# plot the series
sits_merge(point_sg.tb, point.tb) %>% sits_plot()
```

Whittaker filter

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]. The Whittaker filter is a balancing between the residual to the original data and the "smoothness" of the fitted curve. The residual, as measured by the sum of squares of all n time series points deviations, is given by

$$RSS = \sum_i (x_i - \hat{x}_i)^2,$$

where x and $\hat{x}$ are the original and the filtered time series vectors, respectively. The smoothness is assumed to be the measure of the the sum of the squares of the third order differences of the time series [Whittaker, 1922] which is given by

$$\begin{aligned} SSD = {} & (\hat{x}_4 - 3\hat{x}_3 + 3\hat{x}_2 - \hat{x}_1)^2 + (\hat{x}_5 - 3\hat{x}_4 + 3\hat{x}_3 - \hat{x}_2)^2 \\ & + \ldots + (\hat{x}_n - 3\hat{x}_{n-1} + 3\hat{x}_{n-2} - \hat{x}_{n-3})^2. \end{aligned}$$

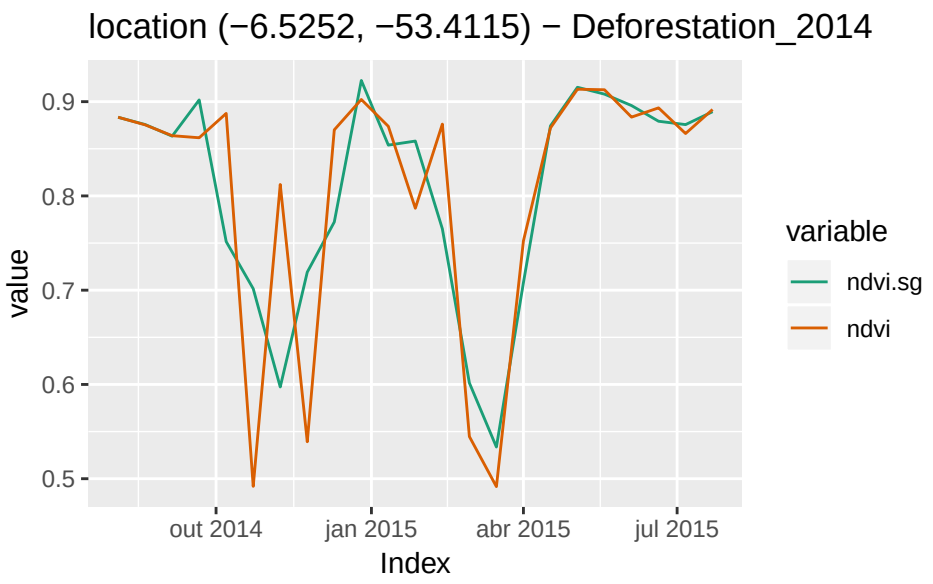


Figure 4: Savitzky-Golay filter applied on a one-year NDVI time series.

The filter is obtained by finding a new time series $\hat{x}$ whose points minimize the expression

$$RSS + \lambda SSD,$$

where λ, a scalar, works as an "smoothing weight" parameter. The minimization can be obtained by differentiating the expression with respect to $\hat{x}$ and equating it to zero. The solution of the resulting linear system of equations gives the filtered time series which, in matrix form, can be expressed as

$$\hat{x} = (\mathrm{I} + \lambda D^{\intercal} D)^{-1} x,$$

where I is the identity matrix and

$$D = \begin{bmatrix} 1 & -3 & 3 & -1 & 0 & 0 & \cdots \\ 0 & 1 & -3 & 3 & -1 & 0 & \cdots \\ 0 & 0 & 1 & -3 & 3 & -1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

is the third order difference matrix. The Whitakker filter can be a large but sparse optimization problem, as we can note from D matrix.

Whittaker smoother has been used only recently in satellite image time series investigations. According to Atzberger and Eilers [2011], the smoother has an advantage over other filtering techniques such as Fourier and wavelets as it does not assume signal periodicity. Moreover, the authors argue that is enables rapid processing of large amounts of data, and handles incomplete time series with missing values.

In the `sits` package, the Whittaker smoother has two parameters: `lambda` controls the degree of smoothing and `differences` the order of the finite difference penalty. The default values are `lambda = 1` and `differences = 3`. Users should be aware that

increasing `lambda` results in much smoother data. When dealing with land use/land cover classes that include both natural vegetation and agriculture, a strong smoothing can reduce the amount of noise in natural vegetation (e.g., forest) time series; however, higher values of `lambda` reduce the information present in agricultural time series, since they reduce the peak values of crop plantations.

The fact that it has only one parameter (λ) facilitates its calibration/comparison process. Zhou et al. [2016] found that $\lambda = 15$ gives the best result when compared with $\lambda = 2$. Larger values of λ produces smoother results.

```
# Take EVI band of the first sample data set
point.tb <- sits_select_bands(prodes_226_064[1,], evi)
# apply Whitaker filter
point_whit.tb <- sits_whittaker(point.tb)
# plot the series
sits_merge(point_whit.tb, point.tb) %>% sits_plot()
```

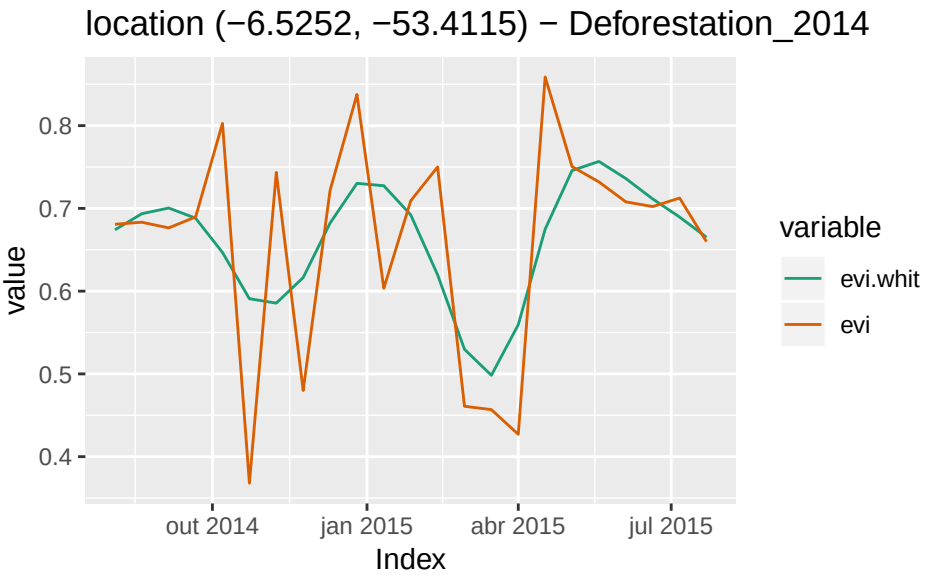


Figure 5: Whittaker smoother filter applied on one-year NDVI time series. The example uses default $\lambda = 1$ parameter.

Envelope filter

This filter can generate a time series corresponding to the superior (inferior) bounding of the input signal. This is accomplished through a convoluting window (odd length) that attributes to the point i, in the resulting time series, the maximum (minimum) value of the points in the window. The i point corresponds to the central point of the window. It can be defined as

$$u_i = \max_k(\{x_k : |i-k| \leq 1\}),$$

whereas an lower dilation is obtained by

$$l_i = \min_k(\{x_k : |i-k| \leq 1\}).$$

Here, x is the input time series and, k and i are vector indices.

The `sits_envelope()` can combine both maximum and minimum window sequentially. The function can receive a string sequence with `"U"` (for maximization) and `"L"` (for minimization) characters passed to its parameter. A repeated sequence of the same character is equivalent to one operation with a larger window. The sequential operations on the input time series produces the final filtered result that is returned.

The envelope filter can be viewed through mathematical morphology lenses, a very common field in digital image processing [Haralick et al., 1987]. Here the operations of `"U"` and `"L"` corresponds to the *dilation* and *erosion* morphological operators applied to univariate arrays [Vávra et al., 2004]. Furthermore, the compounds operation of *opening* and *closing* can be obtained by `"UL"` and `"LU"`, respectively. This technique has been applied on time series analysis in other fields [Accardo et al., 1997] but, for our knowledge, there is no application in satellite image time series literature.

In the following example we can see an application of `sits_envelope()` function. There, we performs the *opening filtration* and *closing filtration* introduced by Vávra et al. [2004]. The correspondent operations sequence are `"ULLULUUL"` and `"LUULULLU"`.

```
# Take the NDVI band of the first sample data set
point.tb <- sits_select_bands(prodes_226_064[1,], ndvi)
# apply envelope filter (remove downward and upward noises)
point_env1.tb <-
    sits_envelope(point.tb,
                  "ULLULUUL",
                  bands_suffix = "OF")
point_env2.tb <-
    sits_envelope(point.tb,
                  "LUULULLU",
                  bands_suffix = "CF")
# plot the series
sits_merge(point_env1.tb, point_env2.tb) %>%
    sits_merge(point.tb) %>%
    sits_plot()
```

ARIMA filter for cloud removal

The cloud filter makes use of the well known autoregressive integrated moving average (ARIMA) model. The algorithm looks to the first order difference time series for points where the value is above a certain threshold. This procedure selects only those points with large variations in the original time series, probably associated with noise. Finally, these points are replaced by the ARIMA correspondent values.

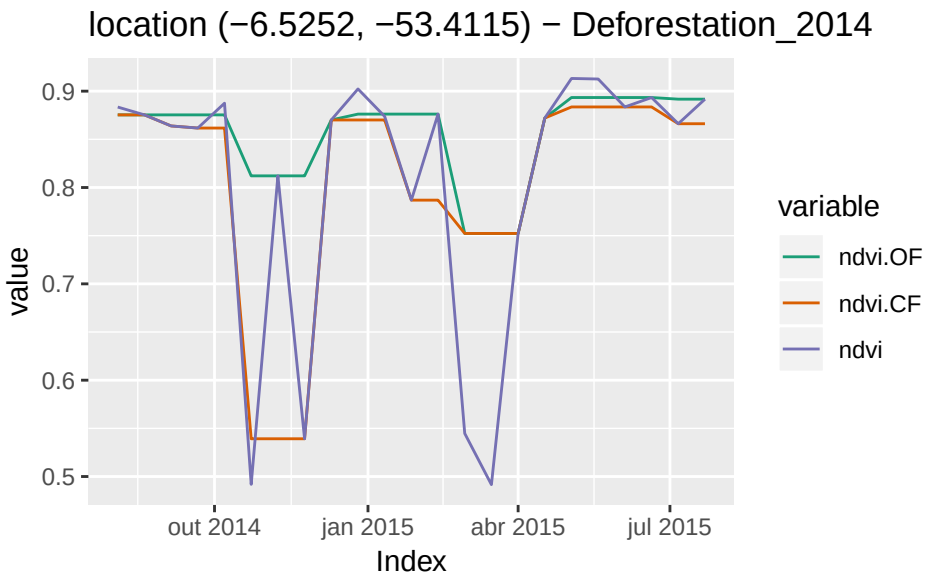
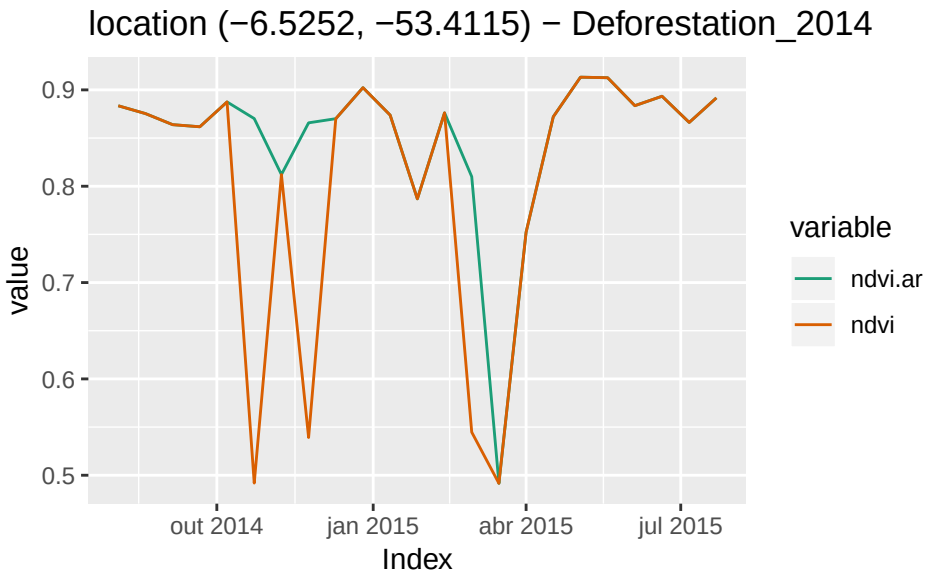


Figure 6: Envelope filter applied on one-year NDVI time series. The examples uses two morfological filters: opening filtration (.OF) and closing filtration (.CF).

The parameters of the ARIMA model can be set by the user. Please see `arima` for the detailed description of parameters p, d, and q.

```
# Take the NDVI band of the first sample data set
point.tb <- sits_select_bands(prodes_226_064[1,], ndvi)
# apply ARIMA filter
point_cf.tb <- sits_ndvi_arima_filter(point.tb, apply_whit = FALSE)
# plot the series
sits_merge(point_cf.tb, point.tb) %>% sits_plot()
```



Machine learning classification for land use and land cover using satelite image time series

The main advantage using satellite image time series in land use studies is that the time series is methodologically consistent with the very nature of the land covers. Using this kind of data allows focusing on land changes through time. Currently, most studies that use satellite image time series for land classification still use variations of the classical remote sensing image classification methods. Given a series of images, researchers use methods that produce a single composite for the whole series [Gomez et al., 2016]. In their review on this subject, Gomez et al. [2016] discuss 12 papers that use satellite image time series to derive image composites that are later used for classification. Câmara et al. [2016] denote these works as taking a *space-first, time-later* approach.

An example of *space-first, time-later* work on big Earth observation data analysis is the work by Hansen et al. [2013]. Using more than 650,000 Landsat images and processing more than 140 billion pixels, the authors compared data from 2000 to 2010 to produce maps of global forest loss during the decade. A pixel-based classification algorithm was used to process each image to detect forest cover. The method classifies each 2D image one by one.

In our view, these methods do not use the full potential of satellite image time series. The benefits of remote sensing time series analysis arise when the temporal resolution of the big data set is able to capture the most important changes. Here, the temporal autocorrelation of the data can be stronger than the spatial autocorrelation. Given data with adequate repeatability, a pixel will be more related to its temporal neighbors than to its spatial ones. In this case, *time-first, space-later* methods lead to better results than the *space-first, time-later* approach [Câmara et al., 2016].

The `sits` package provides functionality to explore the full depth of satellite image time series data. It treat 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. `sits` has support for a variety of machine learning techniques: linear discriminant analysis, quadratic discriminant analysis, multinomial logistic regression, . In the following sections we discuss on support vector machine and random forest machine learning techniques with more detail.

Support Vector Machine

In its base model, the Support Vector Machine (SVM) is a binary classifier that finds a linear boundary in some feature space that not only divides the points of two classes but maximizes the distance between the boundary and the observations. A boundary is a $(p-1)$-dimensional *hyperplane* that defines two complementary subspaces in a p-dimensional feature space.

If the sample observations are linearly separable in the feature space, the hyperplane has a perfect classification property. However, this is hardly what one may expect in a typical satellite image time series scenario. In this regard, the hyperplane optimization problem has a *softner term* that allows some observations to be closer to the margin, or even in the wrong side of its boundary. This relaxation increases the robustness of SVM as it decreases the influence of individual observations on the hyperplane determination.

Moreover, the solution for the hyperplane coefficients depends only on those samples that violates the maximum margin criteria, the so-called *support vectors*. All other points far away from the hyperplane does not exert any influence on the hyperplane coefficients which let SVM less sensitive to outliers.

Hyperplanes are linear $(p-1)$-dimensional boundaries and define linear partitions in the feature space. However, one can enlarge the input attribute space by transforming it into a higher degree feature space. In this manner, the new classification model, despite having a linear boundary on the enlarged feature space, generally translates its hyperplane to a nonlinear boundaries in the original attribute space. As that enlargement can be computationally expensive, SVM makes use of *kernels* functions to overcome such limitation. The use of kernels are an efficient computational strategy to produce nonlinear boundaries in the input attribute space an hence can improve training-class separation.

In `sits`, SVM is the default machine learning model. As a wrapper of `e1071` R package that uses the `LIBSVM` implementation [Chang and Lin, 2011], `sits` adopts the *one-against-one* method for multiclass classification. For a q class problem, this method creates $q(q-1)/2$ SVM binary models, one for each class pair combination and tests any unknown input vectors throughout all those models. The overall result is computed by a voting scheme. The following example illustrate how to classify an individual sample (in this case a series of 16 one-year NDVI time series from the same location). We used the NDVI time series from Mato Grosso Brazilian state as a training data set.

```
# Retrieve the set of samples (provided by EMBRAPA) from the
# Mato Grosso region for train the SVM model
data(samples_mt_ndvi)
# train a machine learning model using SVM
svm_model <- sits_train(samples_mt_ndvi,
                        sits_svm(kernel = "radial",
                                 cost = 10))
# get a point to be classified
data(point_ndvi)
# Classify using SVM model
class.tb <- sits_classify(point_ndvi, svm_model)
sits_plot(class.tb)
```

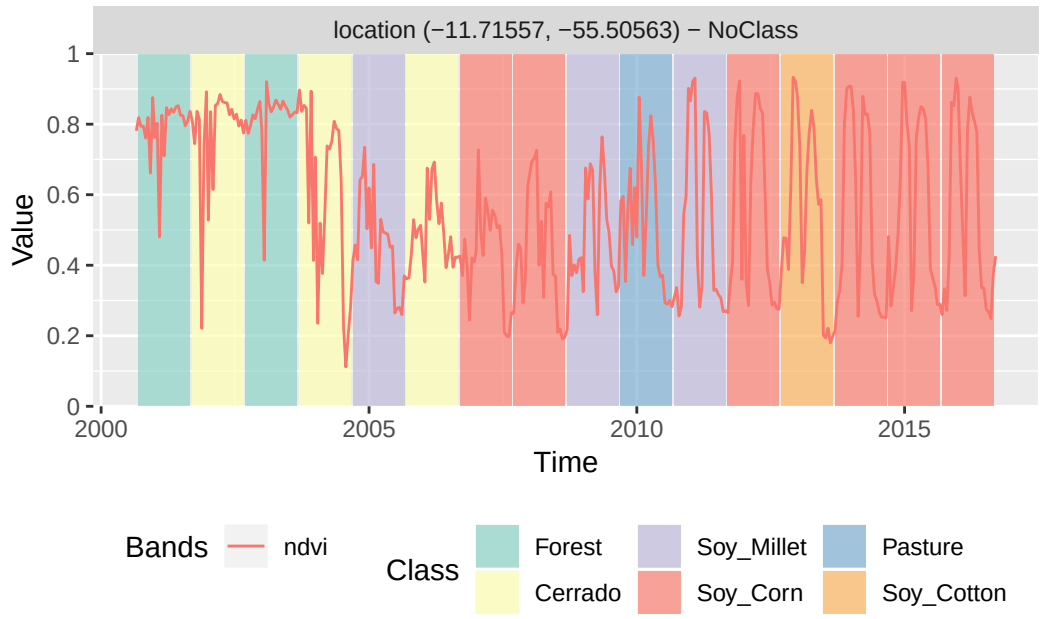


Figure 7: SVM classification of a 16 years time series. The location (latitude, longitude) shown at the top of the graph is in geographic coordinate system (WGS84 *datum*).

Random forest

Random forest uses the idea *decision tree* as its base model. It combines many decision trees via *bootstrap* procedure and *stochastic feature selection*, developing a population of somewhat uncorrelated base models. The final classification model is obtained by a majority voting schema. This procedure decreases the classification variance, improving prediction of individual decision trees.

Random forest training process is essentially nondeterministic. It starts by growing trees through repeatedly random sampling-with-replacement the observations set. At each growing tree, the random forest considers only a fraction of the original attributes to decide where to split a node, according to a *purity criterion*. This decreases the correlation among trees and improves the prediction performance. The most used impurity criterion are *Gini*, *cross-entropy*, and *misclassification error*. The splitting process continues until the tree reaches some given minimum nodes size or a minimum impurity index value.

Random forest provides better performances than *bagged trees*, a similar procedure that does not implement stochastic feature selection (*i.e.* when $m = p$). Bagged trees suffer from high correlation among trees introduced by an eventual presence of strong predictors that tends to be chosen as the splitting criterion [James et al., 2013]. However, the random forest classification performance can vary according to the tunning of the model parameters. The main random forest parameters are the number of attributes sampled as candidates at each split, the number of decision trees to grow, the minimum node size, and the sample fraction to be drawn at each bootstrap iteration.

```
# Retrieve the set of samples (provided by EMBRAPA) from the
# Mato Grosso region for train the Random Forest model.
data(samples_mt_ndvi)
# train a machine learning model using random forest
rfor_model <- sits_train(samples_mt_ndvi, sits_rfor(num.trees = 1000))
# get a point to be classified
data(point_ndvi)
# Classify using Random Forest model
class.tb <- sits_classify(point_ndvi, rfor_model)
sits_plot(class.tb)
```

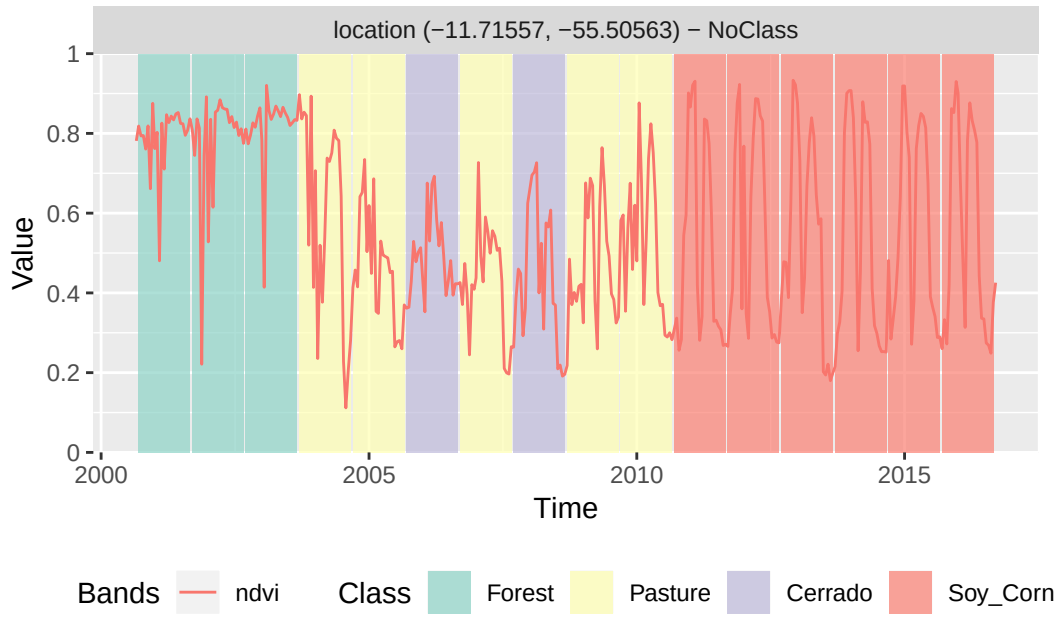


Figure 8: Random forest classification of a 16 years time series. The location (latitude, longitude) shown at the top of the graph are in geographic coordinate system (WGS84 *datum*).

Deep learning methods

In `sits`, the interface to deep learning models is done using the `keras` package [Chollet and Allaire, 2018]. This package implements different deep learning techniques. Currently, `sits` provides access to deep feedforward networks. Also called feedforward neural networks, or multilayer perceptrons (MLPs), these are the quintessential deep learning models. The goal of a feedforward network is to approximate some function f. For example, for a classifier $y = f(x)$ maps an input x to a category y. A feedforward network defines a mapping $y = f(x;\theta)$ and learns the value of the parameters θ that result in the best function approximation. These models are called feedforward because information flows through the function being evaluated from x, through the intermedi-

ate computations used to define f, and finally to the output y. There are no feedback connections in which outputs of the model are fed back into itself [Goodfellow et al., 2016].

Specifying a MLP requires some work on customization, which requires some amount of trial-and-error by the user, since there is no proven model for classification of satellite image time series. The most important decision is the number of layers in the model. Initial tests indicate that 3 to 5 layers are enough to produce good results. The choice of the number of layers depends on the inherent separability of the data set to be classified. For data sets where the classes have different signatures, a shallow model (with 3 layers) may provide appropriate responses. More complex situations require models of deeper hierarchy. The user should be aware that some models with many hidden layers may take a long time to train and may not be able to converge. The suggestion is to start with 3 layers and test different options of number of neurons per layer, before increasing the number of layers.

Three other important parameters for an MLP are: (a) the activation function; (b) the optimization method; (c) the dropout rate. The activation function the activation function of a node defines the output of that node given an input or set of inputs. Following standard practices [Goodfellow et al., 2016], we recommend the use of the "relu" and "elu" functions. The optimization method is a crucial choice, and the most common choices are gradient descent algorithm. These methods aim to maximize an objective function by updating the parameters in the opposite direction of the gradient of the objective function [Ruder, 2016]. Based on experience with image time series, we recommend that users start by using the default method provided by `sits`, which is the `optimizer_adam` method. Please refer to the `keras` package documentation for more information.

The dropout rates have a huge impact on the performance of deep learning classifiers. Dropout is a technique for randomly dropping units from the neural network during training [Srivastava et al., 2014]. By randomly discarding some neurons, dropout reduces overfitting. It is a counter-intuitive idea that works well. Since the purpose of a cascade of neural nets is to improve learning as more data is acquired, discarding some of these neurons may seem a waste of resources. In fact, as experience has shown [Goodfellow et al., 2016], this procedures prevents an early convergence of the optimization to a local minimum. Thus, in practice, dropout rates between 50% and 20% are recommended for each layer.

In the following example, we classify the same data set using a simple example of the `deep learning` method, for fast processing: (a) Two layers with 512 neurons each; (b) Using the 'elu' activation function and 'optimizer_adam'; (c) dropout rates of 40% and 30% for the layers.

```
# Retrieve the set of samples (provided by EMBRAPA) from the
# Mato Grosso region for train the  model.
data(samples_mt_ndvi)
# train a machine learning model using deep learning
train_dl <- sits_deeplearning(
    units            = c(512, 512),
```

```
    activation       = "elu",
    dropout_rates    = c(0.40, 0.30),
    optimizer        = keras::optimizer_adam(lr = 0.001),
    epochs           = 50,
    batch_size       = 128,
    validation_split = 0.2)

dl_model <- sits_train(samples_mt_ndvi, train_dl)
# get a point to be classified
data(point_ndvi)
class.tb <- sits_classify(point_ndvi, dl_model)
sits_plot(class.tb)
```

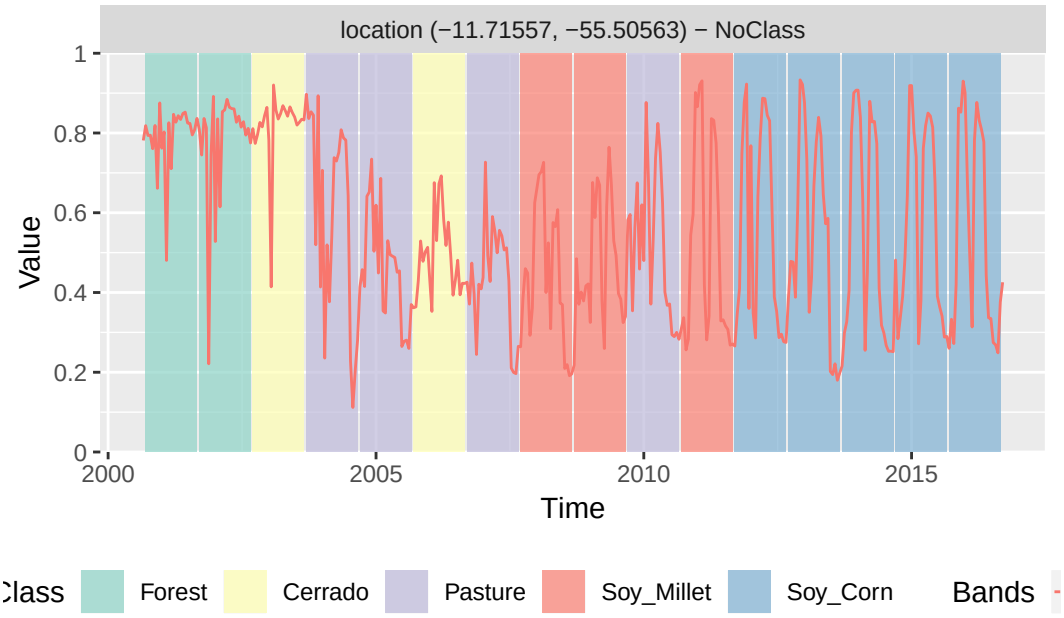


Figure 9: Deep learning classification of a 16 year time series. The location (latitude, longitude) shown at the top of the graph are in geographic coordinate system (WGS84 *datum*).

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).

```
# read a set of samples
data(cerrado_2classes)

# perform a five fold validation with the
# SVM machine learning method using default parameters
prediction.mx <-
    sits_kfold_validate(cerrado_2classes,
                        folds = 5,
                        ml_method = sits_svm())
# prints the output confusion matrix and statistics
sits_conf_matrix(prediction.mx)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Cerrado Pasture
##    Cerrado     390      22
##    Pasture      10     324
##
##           Accuracy : 0.9571
##             95% CI : (0.94, 0.9705)
##
##              Kappa : 0.9136
##
```

```
##  Prod Acc  Cerrado : 0.9750
##  Prod Acc  Pasture : 0.9364
##  User Acc  Cerrado : 0.9466
##  User Acc  Pasture : 0.9701
##
```

Raster 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]. `sits` implements an "out of the box" classification scheme based on *raster bricks* where stacked images have spatial and time dimensions.

Our tested approach (see example illustrated bellow) is `GeoTIFF` bricks, where the temporal dimension is stored in the `GeoTIFF` bands. For a multiband, multitemporal image, all time instances of each band should be stored together in a single `GeoTIFF` file. Different bands for the same images should be stored The classification algorithm implemented in `sits_classify_raster()` allows one 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.

```
# Retrieve the set of samples for the Mato Grosso region
data(samples_mt_ndvi)

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

# read a raster file and put it into a vector
file <- system.file("extdata/raster/mod13q1/sinop-crop-ndvi.tif",
                    package = "sits")

# define the timeline
data("timeline_modis_392")

# create a raster metadata file based on the
# information about the files
raster.tb <-
    sits_coverage(service  = "RASTER",
                  name     = "Sinop-crop",
                  timeline = timeline_modis_392,
                  bands    = "ndvi",
```

```
                  files    = file)
```

```
## Scale factors not provided - will use default values: please check they are valid
```

```
## Data in Sinusoidal projection - assuming MODIS-compatible images
```

```
## Missing values not provided - will use default values: please check they are valid
```

```
## Data in Sinusoidal projection - assuming MODIS-compatible images
```

```
# classify the raster file
raster_class.tb <-
    sits_classify_raster(file = paste0(tempdir(), "/raster-class"),
                         raster.tb,
                         ml_model = svm_model,
                         memsize = 2,
                         multicores = 1)
```

```
## Starting classification at 2018-11-22 09:47:54
```

```
## Elapsed time 0.1 minute(s). Estimated total process time 1.1 minute(s)...
```

```
## Elapsed time 0.1 minute(s). Estimated total process time 0.8 minute(s)...
```

```
## Elapsed time 0.1 minute(s). Estimated total process time 0.7 minute(s)...
```

```
## Elapsed time 0.2 minute(s). Estimated total process time 0.7 minute(s)...
```

```
## Elapsed time 0.2 minute(s). Estimated total process time 0.6 minute(s)...
## Elapsed time 0.2 minute(s). Estimated total process time 0.6 minute(s)...
```

```
## Elapsed time 0.3 minute(s). Estimated total process time 0.6 minute(s)...
## Elapsed time 0.3 minute(s). Estimated total process time 0.6 minute(s)...
## Elapsed time 0.3 minute(s). Estimated total process time 0.6 minute(s)...
```

```
## Elapsed time 0.4 minute(s). Estimated total process time 0.6 minute(s)...
## Elapsed time 0.4 minute(s). Estimated total process time 0.6 minute(s)...
## Elapsed time 0.4 minute(s). Estimated total process time 0.6 minute(s)...
```

```
## Elapsed time 0.5 minute(s). Estimated total process time 0.6 minute(s)...
## Elapsed time 0.5 minute(s). Estimated total process time 0.6 minute(s)...
## Elapsed time 0.5 minute(s). Estimated total process time 0.6 minute(s)...
```

```
## Classification finished at 2018-11-22 09:48:28. Total elapsed time: 0.6 minute(s).
```

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

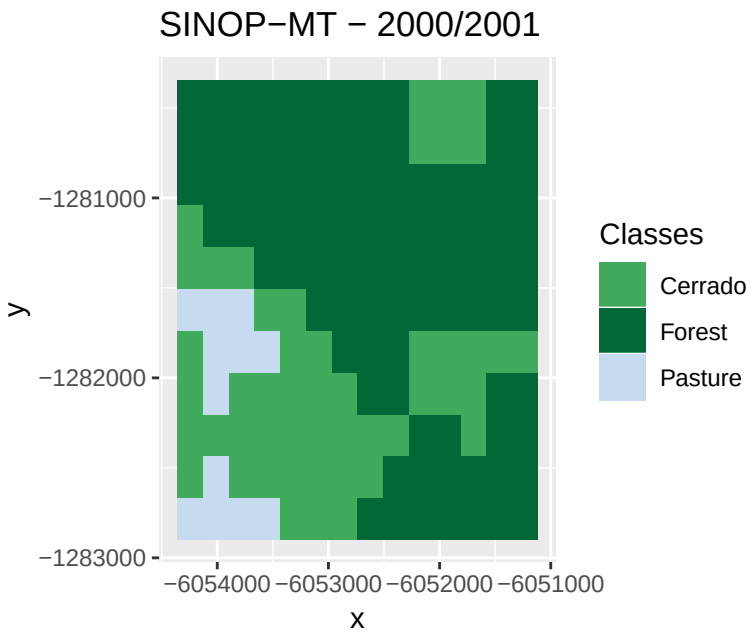


Figure 10: Image (11 × 14 pixels) classified using SVM. The image coordinates (*meters*) shown at vertical and horizontal axis are in MODIS sinusoidal projection.

The classified files can also be visualized with applications such as QGIS. Note that we create two coverage `tibbles` with metadata information, one for the input data and other for the output. To create the input data, we need 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_raster()`. The machine learning model and the training data to be used are passed by the arguments `sits_svm()` (default) and `samples_mt_ndvi` parameters. The classification result is stored as a set of files beginning with `file` prefix.

Processing large raster data files

One of the challenges in land use classification is being able to process large data files. To reduce processing time, it is necessary to adjust `sits_classify_raster()` 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, 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.

Smoothing of raster data during classification

For some applications, users may want to smooth the input data, due to the presence of clouds and noise. To enable this action, `sits_classify_raster()` has three parameters. The `smoothing` parameter is a boolean value, that should be set to `TRUE`. The `sits` package provides the `whittaker` smoother, that is considered to provide the best compromise between noise reduction and information loss [Atzberger and Eilers, 2011]. It requires two additional parameters to be provided: `lambda` and `differences`. Please refer to the discussion on the Whittaker filter above for more details.

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` uses 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

Agostino Accardo, M Affinito, M Carrozzi, and F Bouquet. Use of the fractal dimension for the analysis of electroencephalographic time series. *Biological cybernetics*, 77(5): 339–350, 1997.

Saeed Aghabozorgi, Ali Seyed Shirkhorshidi, and Teh Ying Wah. Time-series clustering–a decade review. *Information Systems*, 53:16–38, 2015.

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.

Peter M Atkinson, C Jeganathan, Jadu Dash, and Clement Atzberger. Inter-comparison of four models for smoothing satellite sensor time-series data to estimate vegetation phenology. *Remote sensing of environment*, 123:400–417, 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.

Bethany A Bradley, Robert W Jacob, John F Hermance, and John F Mustard. A curve fitting procedure to derive inter-annual phenologies from time series of noisy satellite NDVI data. *Remote Sensing of Environment*, 106(2):137–145, 2007.

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, Luiz Fernando Assis, Gilberto Ribeiro, Karine Reis Ferreira, Eduardo Llapa, and Lubia Vinhas. Big earth observation data analytics: matching requirements to system architectures. In *Proceedings of the 5th ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data*, pages 1–6, Burlingname, CA, USA, 2016. ACM.

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.

Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):27, 2011.

Jin Chen, Per. Jönsson, Masayuki Tamura, Zhihui Gu andBunkei Matsushita, and Lars Eklundh. A simple method for reconstructing a high-quality NDVI time-series data set based on the Savitzky–Golay filter. *Remote Sensing of Environment*, 91(3-4):332 – 344, 2004.

Francois Chollet and J.J. Allaire. *Deep Learning with R*. Manning Publications, New York, NY, 2018.

EMBRAPA. Sistema de Análise Temporal da Vegetação (SATVEG), 2014. URL www.satveg.cnptia.embrapa.br.

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.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

M. C. Hansen, P. V. Potapov, R. Moore, M. Hancher, S. A. Turubanova, A. Tyukavina, D. Thau, S. V. Stehman, S. J. Goetz, T. R. Loveland, A. Kommareddy, A. Egorov, L. Chini, C. O. Justice, and J. R. G. Townshend. High-resolution global maps of 21st-century forest cover change. *Science*, 342(6160):850–853, 2013.

Robert M Haralick, Stanley R Sternberg, and Xinhua Zhuang. Image analysis using mathematical morphology. *IEEE transactions on pattern analysis and machine intelligence*, (4):532–550, 1987.

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

Christian Hennig. Clustering strategy and method selection. In Christian Hennig, Marina Meila, Fionn Murtagh, and Roberto Rocci, editors, *Handbook of cluster analysis*. CRC Press, 2015.

Robert J. Hijmans. *raster: Geographic Data Analysis and Modeling*, 2015.

Jennifer N Hird and Gregory J McDermid. Noise reduction of NDVI time series: An empirical comparison of selected techniques. *Remote Sensing of Environment*, 113(1): 248–258, 2009.

Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of classification*, 2 (1):193–218, 1985.

INPE. Amazon Deforestation Monitoring Project (PRODES), 2017. URL www.obt.inpe.br/prodes.

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.

Eamonn Keogh, Jessica Lin, and Wagner Truppel. Clustering of time series subsequences is meaningless: Implications for previous and future research. In *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, pages 115–122, 2003.

E.F. Lambin and M. Linderman. Time series of remote sensing data for land change science. *IEEE Transactions on Geoscience and Remote Sensing*, 44(7):1926–1928, 2006.

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.

Hannibal H Madden. Comments on the Savitzky-Golay convolution method for least-squares-fit smoothing and differentiation of digital data. *Analytical chemistry*, 50(9): 1383–1386, 1978.

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.

François Petitjean, Jordi Inglada, and Pierre Gançarskv. Clustering of satellite image time series under time warping. In *Analysis of Multi-temporal Remote Sensing Images (Multi-Temp), 2011 6th International Workshop on the*, pages 69–72. IEEE, 2011.

Gilberto Ribeiro de Queiroz, Karine Reis Ferreira, Lubia Vinhas, Gilberto Camara, Raphael Willian da Costa, Ricardo Cartaxo Modesto de Souza, Victor Wegner Maus, and Alber Sanchez. WTSS: um serviço web para extração de séries temporais de imagens de sensoriamento remoto. In *Proceeding of the XVII Remote Sensing Brazilian Symposium*, pages 7553–7560, 2015.

LAS Romani, RRV Gonçalves, BF Amaral, DYT Chino, J Zullo, C Traina, EPM Sousa, and AJM Traina. Clustering analysis applied to NDVI/NOAA multitemporal images to improve the monitoring process of sugarcane crops. In *Analysis of Multi-temporal Remote Sensing Images (Multi-Temp), 2011 6th International Workshop on the*, pages 33–36. IEEE, 2011.

Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

Toshihiro Sakamoto, Masayuki Yokozawa, Hitoshi Toritani, Michio Shibayama, Naoki Ishitsuka, and Hiroyuki Ohno. A crop phenology detection method using time-series MODIS data. *Remote Sensing of Environment*, 96(3-4):366–374, 2005.

Yang Shao, Ross S. Lunetta, Brandon Wheeler, John S. Iiames, and James B. Campbell. An evaluation of time-series smoothing algorithms for land-cover classifications using MODIS-NDVI multi-temporal data. *Remote Sensing of Environment*, 174:258–265, 2016.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

F. Vávra, P. Nový, H. Mašková, M. Kotlíková, and A. Netrvalová. Morphological filtration for time series. In *Proceedings of the 3rd International Conference APLIMAT*, pages 983–989, Bratislava, Slovakia, 2004. Slovak University of Technology in Bratislava.

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.

Joe H Ward. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, 58(301):236–244, 1963.

Edmund T Whittaker. On a new method of graduation. *Proceedings of the Edinburgh Mathematical Society*, 41:63–75, 1922.

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.

Jie Zhou, Li Jia, Massimo Menenti, and Ben Gorte. On the performance of remote sensing time series reconstruction methods – a spatial comparison. *Remote Sensing of Environment*, 187:367–384, 2016.