sits: Data Analysis and Machine Learning on Earth Observation Data Cubes with Satellite Image Time Series

Rolf Simoes Gilberto Camara Felipe Souza
Lorena Santos Pedro R. Andrade Charlotte Peletier
Alexandre Carvalho Karine Ferreira Gilberto Queiroz
Victor Maus

2021-06-01

Contents

Preface

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 to classify land use and land cover since our growing demand for natural resources has caused significant environmental impacts.

This book presents **sits**, an open-source R package for land use and land cover change mapping using satellite image time series. The package uses machine learning techniques to classify image time series obtained from data cubes. Methods available include linear and quadratic discrimination analysis, support vector machines, random forests, boosting, deep learning, and convolutional neural networks. The package also provides functions for post-processing and sample quality assessment.

Who this book is for

The target audience for **sits** is the new generation of specialists who understand the principles of remote sensing and can write scripts in **R**. Ideally, users should have basic knowledge of data science methods using R. If you are new to R, we highly recommend the excellent self-learning book "Hands-On Programming with R" by Garrett Golemund from RStudio. If you want more information on the data science methods used in the book, please look at the following references:

- Wickham, H.; Golemund, G., "R for Data Science". O'Reilly, 2017.

- James, G.; Witten, D.; Hastie, T.; Tibshirani, R. "An Introduction to Statistical Learning with Applications in R". Springer, 2013.

- Goodfellow, I; Bengio, Y.; Courville, A. "Deep Learning". MIT Press, 2016.

How to use this book

This book describes **sits** version 0.11.2. Download and install the package as explained in the Setup. Start at Chapter 1 to get an overview of the package. Then feel free to browse the chapter for more information on topics you are interested in.

Chapter	Description
Chr 1	Provides an overview of **sits**.
Chr 2	Describes how to work with Earth observation data cubes.
Chr 3	Describes how to access information from time series.
Chr 4	Improving the quality of the samples used in training models
Chr 5	Presents the machine learning techniques available in sits.
Chr 6	Describes how to classify satellite images associated with Earth observation data cubes.
Chr 7	Describes smoothing method to reclassify the pixels based on the machine learning probabilities
Chr 8	Presents the validation and accuracy measures.
Chr 9	Presents case studies of LUCC classification.
Chr 10	How to develop extensions to **sits**.

Publications using sits

This section gathers the publications that have used **sits** to generate the results.

2021

- [1] Lorena Santos, Karine Ferreira at el., "Identifying Spatiotemporal Patterns in Land Use and Cover Samples from Satellite Image Time Series". Remote Sens. 2021, 13, 974.

2020

- [2] Rolf Simoes, Michelle Picoli, et al., "Land use and cover maps for Mato Grosso State in Brazil from 2001 to 2017". Sci Data 7, 34 (2020).

- [4] Michelle Picoli, Ana Rorato, et al., "Impacts of Public and Private Sector Policies on Soybean and Pasture Expansion in Mato Grosso – Brazil from 2001 to 2017". Land 2020, 9, 20.

- [5] Ferreira, K.R.; Queiroz, G.R. "Earth Observation Data Cubes for Brazil: Requirements, Methodology and Products". Remote Sens. 2020, 12, 4033.

2019

- [6] Alber Sanchez, Michelle Picoli, et al., "Land Cover Classifications of Clear-cut Deforestation Using Deep Learning". In: SIMPÓSIO

BRASILEIRO DE GEOINFORMÁTICA (GEOINFO), 2019, São José dos Campos. São José dos Campos: INPE, 2019. On-line.

- [7] Lorena Santos, Karine Ferreira, et al., "Self-Organizing Maps in Earth Observation Data Cubes Analysis". 13th International Workshop on Self-Organizing Maps and Learning Vector Quantization, Clustering and Data Visualization (WSOM+ 2019), Barcelona, Spain, June 26-28, 2019.

2018

- [8] Michelle Picoli, Gilberto Camara, et al., "Big Earth Observation Time Series Analysis for Monitoring Brazilian Agriculture". ISPRS Journal of Photogrammetry and Remote Sensing, 2018.

Reproducible papers used in building sits functions

We thank the authors of these papers for making their code available to be used in sits.

- [1] Appel, Marius, and Edzer Pebesma, "On-Demand Processing of Data Cubes from Satellite Image Collections with the Gdalcubes Library." Data 4 (3): 1–16, 2020.
- [2] Hassan Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller, "Deep learning for time series classification: a review". Data Mining and Knowledge Discovery, 33(4): 917–963, 2019.
- [3] Pebesma, Edzer, "Simple Features for R: Standardized Support for Spatial Vector Data". R Journal, 10(1):2018.
- [4] Pelletier, Charlotte, Geoffrey I. Webb, and Francois Petitjean. "Temporal Convolutional Neural Network for the Classification of Satellite Image Time Series." Remote Sensing 11 (5), 2019.
- [5] Wehrens, Ron and Kruisselbrink, Johannes. "Flexible Self-Organising Maps in kohonen 3.0". Journal of Statistical Software, 87, 7 (2018).

Setup

sits is currently available on GitHub. Thus, installing the package can be accomplished via devtools, as presented by the code snippet below:

Docker images

Installing the **sits** package has several dependencies that increase its installation and build time. To speed up the use of **sits** and the required dependencies in the R environment, the Brazil Data Cube (BDC) project maintains Docker images of the RStudio Server already configured with **sits**. The command below shows how this image can be used in Docker.

After the execution of above command, open the URL `http://127.0.0.1:8787` in a web browser, in order to access the RStudio:

    firefox http://127.0.0.1:8787

```
To login use 'sits' as user and password.
```

If you prefer a customized build of the SITS Docker images, please, visit the sits-docker GitHub repository.

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 the 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 work.

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").

The authors would like to acknowledge the contributions of Marius Appel, Tim Appelhans, Henrik Bengtsson, Matt Dowle, Robert Hijmans, Edzer Pebesma, and Ron Wehrens, respectively chief developers of the packages "gdalcubes", "mapview", "future", "data.table", "terra/raster", "sf"/"stars", and "kohonen". The code in "sits" is also much indebted to the work of the RStudio team, including the "tidyverse" and the "furrr" and "keras" packages. We also thank Charlotte Pelletier, and Hassan Fawaz for sharing the python code that has been reused for the "TempCNN" and "ResNet" machine learning models.

Part I

Overview

Chapter 1

A taste of sits

This chapter present an overview of **sits** by showing an application example. For detailed description of the functions, please see the following chapters.

Earth observation (EO) satellites provide a common and consistent set of information about the planet's land and oceans. 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 significant challenge: *How to design and build technologies that allow the Earth observation community to analyze big data sets?*

In this book, 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. The package supports the complete cycle of data analysis for time series classification, including data acquisition, visualization, filtering, clustering, classification, validation, and post-processing.

The *sits* package adopts a *time-first, space-later* approach, where each spatial location is associated to a time series. A set of locations with known labels is used to train a machine learning classifiers. The resulting model is applied to the data cube and each time series is classified separately. After the classification, spatial smoothing methods capture information from neighbors. This approach is illustrated in Figure 1.1.

1.1 Creating a Data Cube

In what follows, we introduce **sits** by showing a simple example fo land use and land cover classification. The first step is creating a data cube. The data

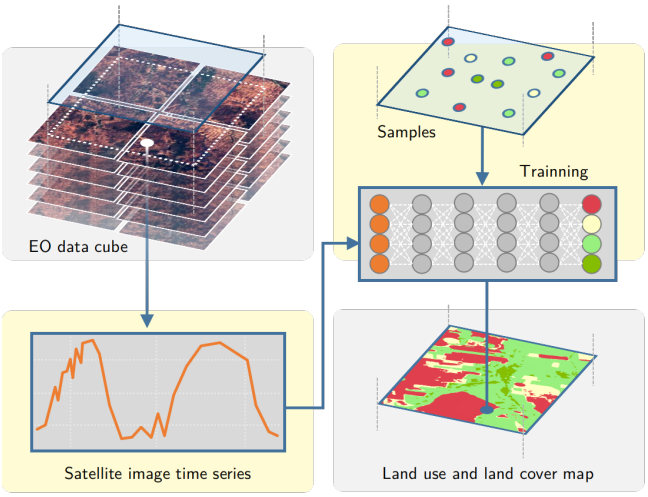


Figure 1.1: Using time series for land classification (source: authors)

cube is a set of analysis-ready MODIS MOD13Q1 images for the Sinop region of the State of Mato Grosso, Brazil, in the bands "NDVI" and "EVI", covering a one-year period from 2013-09-14.[1] Each band has 23 instances, each covering a 16-day period, which is the standard for the MOD13Q1 product[1]. The data is available in the R package "sitsdata", which contains data for the examples in this book. The code below shows how to create data cubes from local files. To work with data cubes in repositories such as AWS, MS/AZURE and the Brazil Data Cube, please see Chapter 2

The `sits_cube()` function defines a *data cube*, which is an organized collection of images covering a geographical area in a given time interval. Data cubes can be conceived as a 3D array of pixels, where each pixel is associated to a time series. All pixels share the same timeline and the same set of attributes (usually spectral bands). When a data cube is defined, the values of the images are not loaded in memory. The output of `sits_cube` is a object-relational data table, which contains the metadata that describes the actual image data. For more details on how to define and work with data cubes, please see

1.2 The time series table

To classify all of the time series associated to a data cube, **sits** uses machine learning models. FTo train the models, **sits** uses a tabular data structure that stores individual time series. The example below shows a table with 1,218 time series obtained from MODIS MOD13Q1 images. Each series has four attributes: two bands ("NIR" and "MIR") and two indexes ("NDVI" and "EVI"). This data

[1] In this book, we follow the convention "year-month-day" for dates

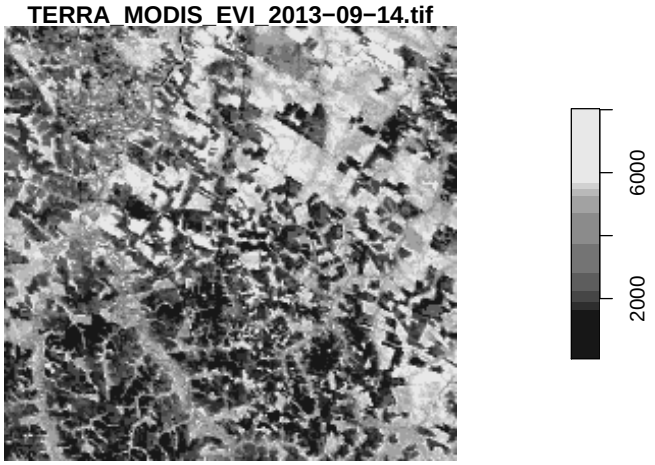


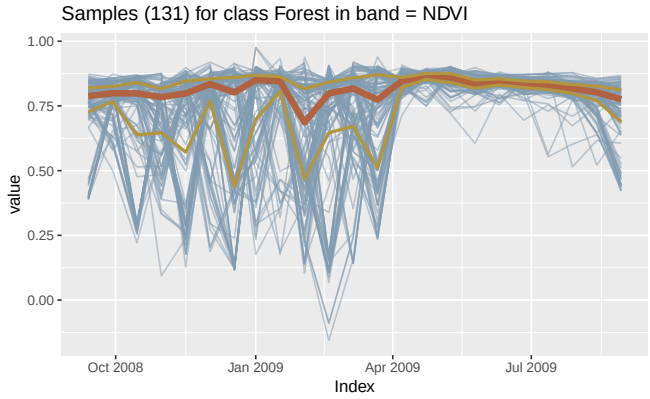
Figure 1.2: EVI band for 2013-09-14

set is available in package "sitsdata".

```
#> # 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]>
```

The data structure associated to the time series is a table that contains data and metadata. The first six columns contain the metadata: spatial and temporal information, the label assigned to the sample, and the data cube from where the data has been extracted. The `time_series` column contains the time series data for each spatiotemporal location. This data is also organized as a table, with a column with the dates and the other columns with the values for each spectral band. For more details on how to handle time series data, please see Chapter 3.

It is useful to visualise the disperson of the time series. In what follows, for brevity we will select only one label ("Forest") and one index ("EVI") to show. The resulting plot shows all of the time series associated to the label and attribute, highlighting the median and the first and third quartiles.



1.3 Training a machine learning model

After obtaining the time series, the next step is to select a suitable subset to use as training samples for a machine learning model. In this case, the time series data has four attributes ("EVI", "NDVI", "NIR", "MIR") and the data cube is composed only with data from the "NDVI" and "EVI" indexes. We extract the "NDVI" and "EVI" indexes from the time series data set and use the resulting data for training a model. To build the classification model, we have chosen `sits_TempCNN()` from the methods available. This method implements a 1D convolution neural network [2]. After training the model, we plot the result to how well it has converged to match the input data. For more details on the machine learning methods please see Chapter 5.

```
#> `geom_smooth()` using formula 'y ~ x'
```

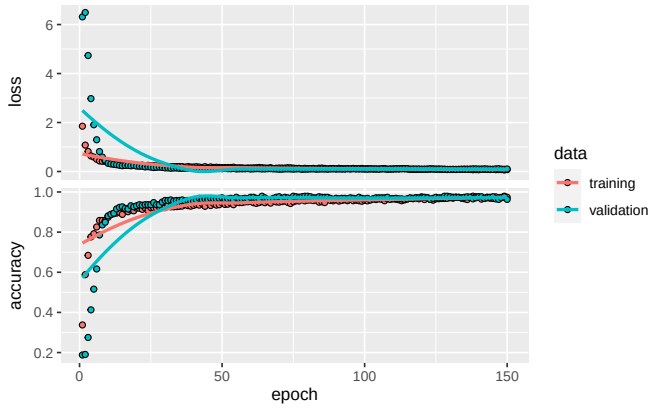


Figure 1.3: Validation of TempCNN model

1.4 Data cube classification

The next step is to classify the data cube. This is achived by using the `sits_classify()` function. The classification produces a set of probability maps, one for each class. For each map, the value of a pixel is proportional to the the probability that it belongs to the class. To visualise the result, we plot the probability maps. In the example below, for clarity's sake, we show the maps of two classes ("Forest" and "Pasture"). Details of the classification process are available in Chapter 6.

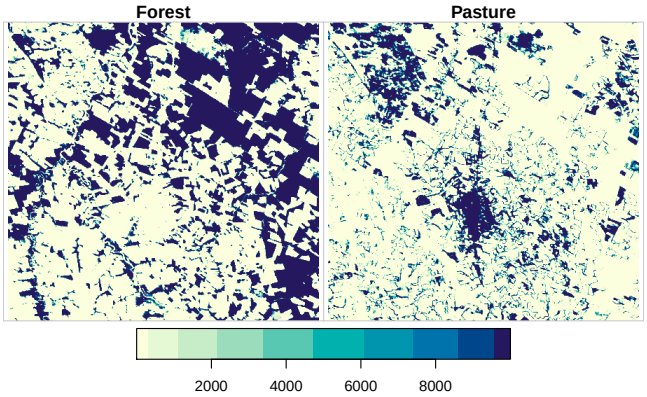
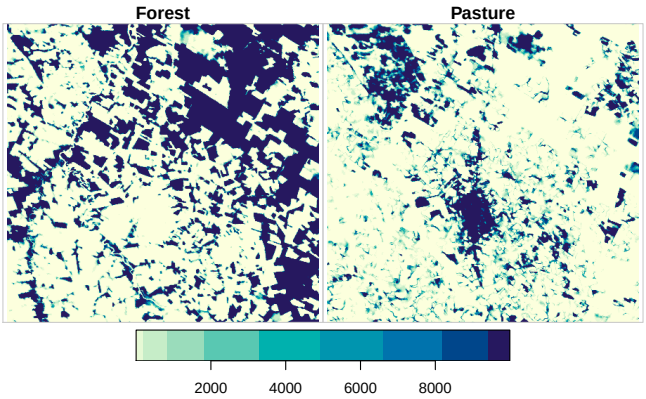


Figure 1.4: Probability map for classes of each pixel

1.5 Spatial smoothing

When working with big EO data sets, there is a considerable degree of data variability in each class. As a result, some pixels will be misclassified. These errors are more likely to occur in transition areas between classes or when dealing with mixed pixels. To offset these problems, **sits** includes a post-processing smoothing method based on Bayesian probability. The `sits_smooth()` function uses information from a pixel's neighborhood to reduce uncertainty about its label, which is illustrated below. After smoothing, we plot the probability maps for classes "Forest" and "Pasture" to compare with the previous plot. For more discussion on post-processing and smoothing methods, please see Chapter 7 7.



1.6 Labelling a probability data cube

After removal of outliers using local smoothing, one can obtain the labeled classification map using the `sits_label_classification()` function which assigns each pixel to the class with highest probability.

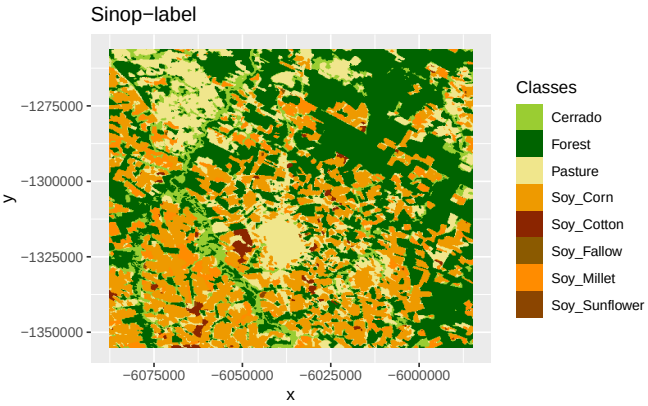


Figure 1.5: Labeled classification map

The resulting classification files can be read by QGIS. Links to the associated files are available in the `sinop_label` table in the column `file_info`.

```
#> # A tibble: 1 x 4
#>   band            start_date end_date   path
#>   <chr>           <date>     <date>     <chr>
#> 1 Sinop_probs_cl~ 2013-09-14 2014-08-29 /tmp/RtmpBY6Zua/Sinop_class_PROBS_bay~
```

1.7 How the sits API works

The core functions of the **sits** API are presented in Figure 1.6. Each function carries out one task of the land classification workflow. These functions are: (a) `sits_cube()` which creates a cube; (b) `sits_get_data()` which extracts training data from the cube; (c) `sits_train()` that trains a machine learning model; (d) `sits_classify()` which classifies the cube; (e) `sits_smooth()` that does the spatial smoothing; and (f) `sits_label_classication()` that produces the final labelled image. These six functions encapsulate the core of the package. Each of these core functions is described in a chapter in this book.

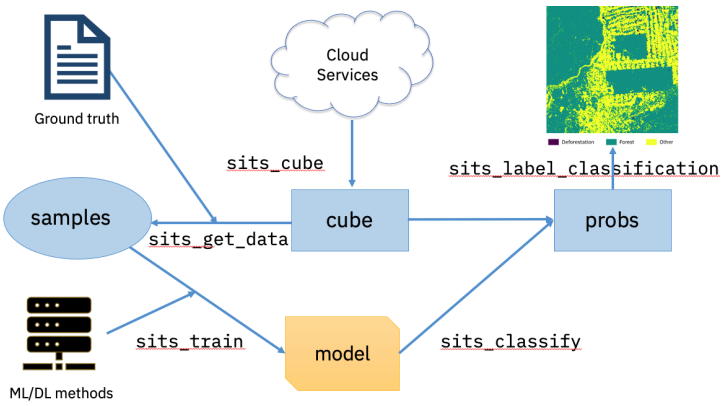


Figure 1.6: Main functions of the SITS API

1.8 Final remarks

The **sits** package provides an API to build EO data cubes from image collections available in cloud services, and to perform land classification of data cubes using machine learning. The classification models are built based on satellite image time series extracted from the cubes. The package provides additional function for sample quality control, post-processing and validation. The design of the API tries to reduce complexity for users and hide details such as how to do parallel processing, and to handle data cubes composed by tiles of different timelines.

Chapter 2

Earth observation data cubes

This chapter describes how to use Earth observation data cubes in SITS.

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

Given the large sizes of the collections of Earth observation data available, there is a clear trend to use cloud computing. In this configuration, cloud services archive large satellite-generated data sets and provide computing facilities to process them. Users can share big Earth observation databases and minimize data download. Investment in infrastructure is minimized, and sharing of data and software increases.

To take full advantage of the cloud computing model, Earth observation data needs to be available to users as *data cubes*, whose aim is to organize satellite data for a given area in a consistent spatiotemporal arrangement. Generalizing [3], we consider the following definition:

1. A data cube is a four-dimensional structure with dimensions x (longitude or easting), y (latitude or northing), time, and bands.
2. Its spatial dimensions refer to a single spatial reference system (SRS). Cells of a data cube have a constant spatial size with respect to the cube's SRS.
3. A set of intervals specifies the temporal dimension.
4. For every combination of dimensions, a cell has a single value.

Conceptually, a data cube defines a compact space. For all positions inside its spatiotemporal extent, it is possible to obtain a valid set of measures. For each position in space, the data cube should provide a valid time series. For each time interval, the data cube should provide a valid 2D image (see Figure 2.1). Data cubes provide a useful abstraction for algorithms that extract information from big EO data sets. Machine learning and deep learning algorithms require the input data to be consistent. The dimensionality of the data used for training the model has to be the same as that of the data to be classified. There should be no gaps in the input data and no missing values are allowed.

Currently, few cloud providers adhere to the full definition of data cubes. One exception is the Brazil Data Cube (BDC) [4], which provides data cubes which are regular in both spatial and temporal dimension.

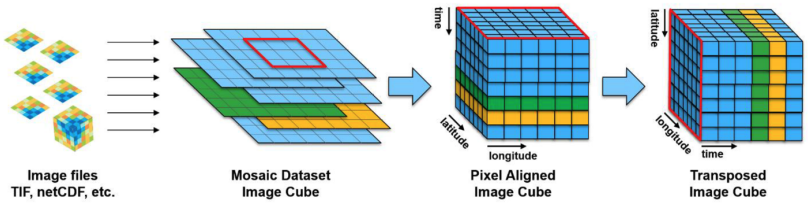


Figure 2.1: Conceptual view of data cubes (source: authors)

2.2 Analysis-ready data image collections

Data available in cloud services such as AWS, Microsoft Planetary Computer, and Digital Earth Africa does not adhere to the definition of data cubes stated above. These data sets are better described as collections of analysis-ready data (ARD) which have been processed by space agencies to improve multidate comparability. Such processing includes conversion from radiance measures at the top of the atmosphere to reflectance measures from ground areas. Variations in sun incidence angles are also compensated. The image is usually reprocessed to a well-known cartographic projection. We define *ARD image collections* as:

1. An ARD image collection is a set of files from a given sensor (or a combined set of sensors) that has been corrected to ensure comparability of measurements between different dates.
2. All images are reprojected to a cartographic projection following well-established standards.
3. Image collections are cropped into a tiling system. In general, the timeline of different tiles do not match.

ARD image collections do not guarantee that every pixel of an image has a valid value. Images still contains cloudy or missing pixels. Figure 2.2 shows images of

tile "20LKP" of the Sentinel-2 Level 2A collection available in AWS for different dates. Some of images have a significant number of clouds. The values of cloudy pixels should be replaced by valid values before being ingested into a data cube.

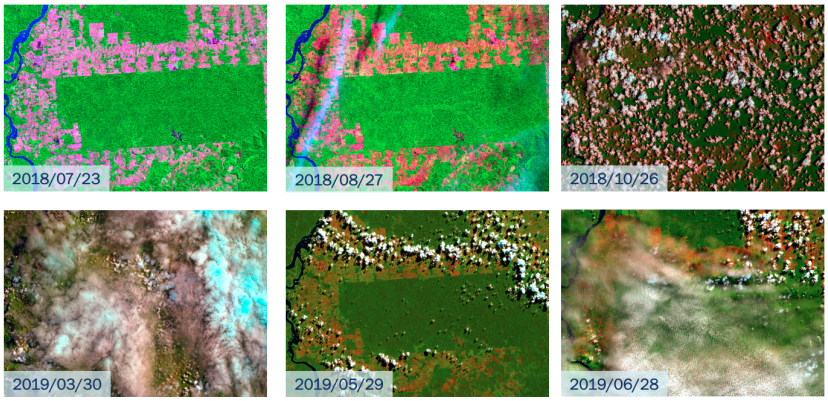


Figure 2.2: Sentinel-2 for tile 20LKP in different dates (source: authors).

A further point concerns the timeline of different tiles. Consider the neighboring Sentinel-2 tiles "20LLP" and "20LKP" for the period 2018-07-13 to 2019-07-28, as they are available in AWS. Tile 20LKP has 71 temporal instances and tile 20 LLP has 144 instances. To process large areas, all tiles have to be organized to follow the same timeline. Thus, users cannot rely on ARD image collections when working with large areas. Users that want to work on multiples tiles of an ARD image collection should use the `sits_regularize()` function to generate regularly spaced cubes in time from image collection. Please see the session "Regularizing data cubes" below.

2.3 Accessing Data Cubes and Image Collections in SITS

To obtain information on cloud image collection, *sits* uses information provided by implementations of the STAC (SpatioTemporal Asset Catalogue) protocol. [STAC] (https://stacspec.org/) is a specification of geospatial information which has been adopted by many large image collection providers (e.g., AWS, Microsoft, USGS). A 'spatiotemporal asset' is any file that represents information about the earth captured in a certain space and time. To access STAC endpoints, *sits* uses the rstac R package.

2.3.1 Accessing data cubes in Amazon Web Services

Users of Amazon Web Services (AWS) can access image collections available in the 'Earth on AWS' services using *sits*. For AWS, *sits* currently works with collection "sentinel-s2-l2a". This will be extended in later versions.

To work with AWS, users need to provide credentials using environment variables.

Sentinel-2 level 2A files in AWS are organized by sensor resolution. The AWS bands in 10m resolution are “B02”, “B03”, “B04”, and “B08”. The 20m bands are “B02”, “B03”, “B04”, “B05”, “B06”, “BO7”, B08“,”B8A“,”B11“, and”B12". All 12 bands are available at 60m resolution. To create data cubes in AWS using Sentinel-2 images, users need to specify the `s2_resolution` parameter. In the example below, the user selects two Sentinel-2 tiles. Each S2 tile is an 100x100 km2 orthoimage in UTM/WGS84 projection.

The output of the `sits_cube` function is composed of metadata about the images that satisfy the requirements stated in its parameters (spatiotemporal extent, resolution, and area of interest). The `s2_cube` object created in the above statement is a tibble that has the information required for further processing, but does not contain the actual data.

Instead of specifying the region of interest by listing the image collection tiles, users can also provide a bounding box (`bbox`) whose parameters allow a selection of an area of interest. Bounding boxes can be defined using: (a) a named vector (“xmin”, “ymin”, “xmax”, “ymax”) with lat/long values in WGS 84; (b) an *sf* object from the *sf* package, a data frame with feature attributes and feature geometries; or (c) a GeoJSON geometry (RFC 7946). When selecting images that compose a data cube based on a `bbox`, sits does not crop them directly; the software selects the images that intersect with it. The information is used later by `sits_classify()`, when only the pixels inside the bounding box will be processed.

2.3.2 Accessing the Brazil Data Cube

The Brazil Data Cube (BDC) is being developed by Brazil’s National Institute for Space Research (INPE). Its goal is to create multidimensional data cubes of analysis-ready data Brazil. The BDC uses three hierarchical grids based on the Albers Equal Area projection and SIRGAS 2000 datum. The three grids are generated taking -54 ° longitude as the central reference and defining tiles of 6×4, 3×2 and 1.5×1 degrees. The large grid is composed by tiles of 672×440 km^2 and is used for CBERS-4 AWFI collections at 64 meter resolution; each CBERS-4 AWFI tile contains images of $10,504 \times 6,865$ pixels. The medium grid is used for Landsat-8 OLI collections at 30 meter resolution; tiles have an extension of 336×220 km^2 and each image has $11,204 \times 7,324$ pixels. The small grid covers 168×110 km^2 and is used for Sentinel-2 MSI collections at 10m resolutions; each image has $16,806 \times 10,986$ pixels. The data cubes in the BDC are regularly spaced in time and cloud-corrected.

To access the Brazil Data Cube, users need to provide their credentials using environmental variables.

Creating a data cube using the BDC is similar to what is required for AWS. The user defines an image collection, a spatiotemporal extent, bands, and optionally

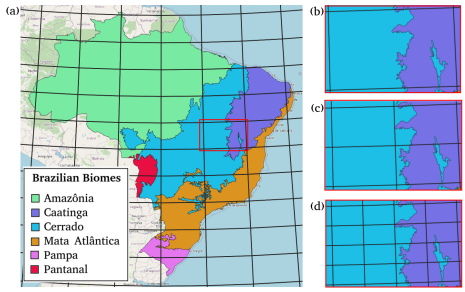


Figure 2.3: Hierarchical BDC tiling system showing overlayed on Brazilian Biomes (a), illustrating that one large tile (b) contains four medium tiles (c) and that medium tile contains four small tiles

a bounding box. In the example below, the data cube is defined as one tile ("022024") of "CB4_64_16D_STK-1" collection which holds CBERS AWFI images at 16 days resolution. Other collections include "LC8_30_16D_STK-1" (Landsat OLI images at 16 days) and "S2_10_16D_STK-1" (Sentinel-2 MSI images at 16 days).

2.3.3 Defining a data cube using files

In some cases, users have downloaded files from image collections and have them available in their computer or in a local network. As *sits* does not have access to STAC information that describe the files, they should be organized and named to allow SITS to create a data cube.

All files should be in the same directory and have the same spatial resolution and projection. Each file should contain a single image band for a single date. Since raster files in popular formats (e.g., GeoTiff and JPEG 2000) do not include temporal and band information, each file name needs to include date and band. For example, `CBERS-4_AWFI_B13_2018-02-02.tif` and `SENTINEL-2_MSI_L_20_KP_20m_B08_2021_03_29.jp2` are is a valid name. The user should 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 `"_"`.

2.4 Regularizing data cubes

Analysis-ready data (ARD) collections available in AWS and DE Africa do not have consistent timelines. In general, images in neighboring tiles have different timelines. This is a problem when classifying large areas, In this case, users may want to produce data cubes with regular time intervals. For example, a user may want to define the best Sentinel-2 pixel in a one-month period. This can be done in *sits* by the `sits_regularize`, which calls the "gdalcubes" package[3]. For details in gdalcubes, please see https://github.com/appelmar/gdalcubes.

```
gc_cube <- sits_regularize(
          cube       = s2_cube,
          name       = "T20LKP_20LLP_2018_2019_1M",
          dir_images = tempdir(),
          period     = "P1M",
          agg_method = "median",
          resampling = "bilinear",
          cloud_mask = TRUE
)
```

In the above example, the user has selected the `s2_cube` object defined using AWS (see example above). As described earlier in this chapter, because of the way ARD image collections are built, the timelines of tiles "20LLP" and "20LKP" associated with this cube are different. The `sits_regularize` function builds a new data cube, with the same temporal extent as the `s2_cube` but with the same timeline. In this function, the `period` parameter controls the temporal interval between two images. Values should abide by the ISO8601 time period specification, which states that time interval should be defined as "P[n]Y[n]M[n]D", where Y stands for "years", "M" for months and "D" for days. Thus, "P1M" stands for a one-month period, "P15D" for a fifteen-day period.

When joining different images to get the best image for a period, `sits_regularize` uses an aggregation method, defined by the parameter `agg_method`. It specifies how individual values of different pixels should be combined. The default is `median`, which select the most frequent value for the pixel during the desired interval. For more details, see `?sits_regularize`.

Chapter 3

Working with time series

This chapter describes how to access information from time series in SITS.

3.1 Data structures for satellite time series

The *sits* package requires a set of time series data, describing properties in spatiotemporal 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 organize data in tables. Tibbles are part of the `tidyverse`, a collection of R packages designed to work together in data manipulation [5]. As an example of how the sits tibble works, the following code shows the first three lines of a tibble containing $1,882$ labeled samples of land cover in Mato Grosso state of Brazil. The samples contain time series extracted from the MODIS MOD13Q1 product from 2000 to 2016, provided every 16 days at 250-meter spatial resolution in the Sinusoidal projection. Based on ground surveys and high-resolution imagery, it includes samples of nine classes: "Forest", "Cerrado", "Pasture", "Soybean-fallow", "Fallow-Cotton", "Soybean-Cotton", "Soybean-Corn", "Soybean-Millet", and "Soybean-Sunflower".

```
#> # 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, the 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 labeled "Cerrado, at location (-58.5631, -13.8844) and is considered valid for the period (2007-09-14, 2008-08-28). Informing the dates where the label is valid is crucial for correct classification. In this case, the researchers involved in labeling the samples chose to use the agricultural calendar in Brazil, where the spring crop is planted in the months of September and October, and the autumn crop is planted in the months of February and March. For other applications and other countries, the relevant dates will most likely be different from those used in the example. The `time_series` column contains the time series data for each spatiotemporal location. This data is also organized as a tibble, with a column with the dates and the other columns with the values for each spectral band.

3.2 Utilities for handling time series

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

```
#> # A tibble: 9 x 3
#>   label         count   prop
#>   <chr>         <int>  <dbl>
#> 1 Cerrado         379 0.200
#> 2 Fallow_Cotton    29 0.0153
#> 3 Forest          131 0.0692
#> 4 Pasture         344 0.182
#> 5 Soy_Corn        364 0.192
#> 6 Soy_Cotton      352 0.186
#> 7 Soy_Fallow       87 0.0460
#> 8 Soy_Millet      180 0.0951
#> 9 Soy_Sunflower    26 0.0137
```

In many cases, it is helpful 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 the embedded time series, one can use the functions from `dplyr`, `tidyr`, and `purrr`

packages of the `tidyverse` [5] to process the data. For example, the following code uses `sits_select()` 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".

3.3 Time series visualisation

Given a small number of samples to display, `plot` tries to group as many spatial locations together. In the following example, the first 12 samples of "Cerrado" class refer to the same spatial location in consecutive time periods. For this reason, these samples are plotted together.

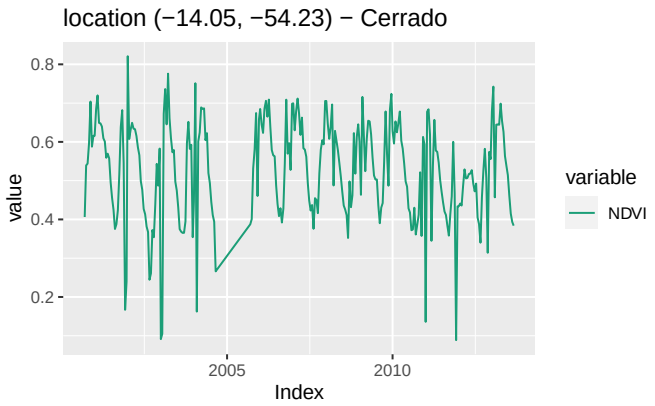


Figure 3.1: Plot of the first 'Cerrado' sample from data set

For a large number of samples, where the number 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 `plot.sits()` has more details about the different ways it can display data.

3.4 Obtaining time series data from data cubes

To get a time series in sits, one has to create a data cube, as described previously. 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 a data cube, 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

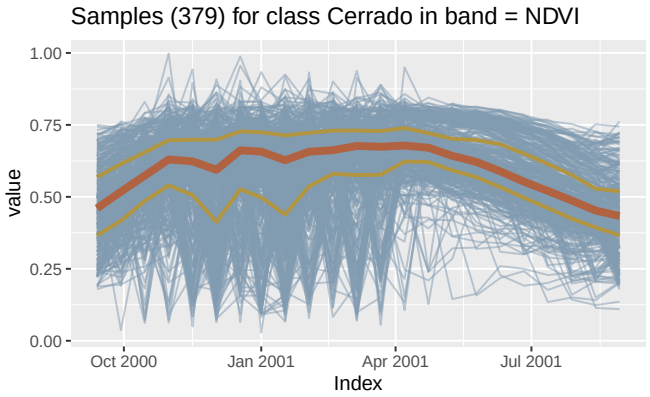


Figure 3.2: Plot of all Cerrado samples from data set

periods. The result is a tibble that can be visualized using `plot()`.

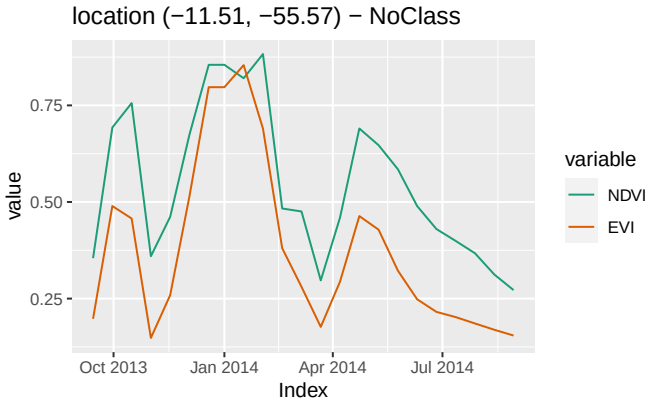


Figure 3.3: NDVI and EVI time series fetched from local raster cube.

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 that are labelled and commonly stored in plain text files in comma-separated values (CSV) or using shapefiles (SHP). Function `sits_get_data()` takes a CSV or SHP file path as an argument. In the case of CSV text file, they should provide, for each training sample, its latitude and longitude, the start and end dates, and a label associated with a ground sample. An example of a CSV file used is shown below.

```
#>   id longitude  latitude start_date   end_date   label
#> 1  1 -55.65931 -11.76267 2013-09-14 2014-08-29 Pasture
#> 2  2 -55.64833 -11.76385 2013-09-14 2014-08-29 Pasture
#> 3  3 -55.66738 -11.78032 2013-09-14 2014-08-29  Forest
```

The main difference between the files used by *sits* to retrieve training samples from those used traditionally in remote sensing data analysis is that users are expected to provide the temporal information (`start_date` and `end_date`). In the simplest case, all samples share the same start and end data. That is not a strict requirement. Users can specify different dates, as long as they have a compatible duration. For example, the data set `samples_modis_4bands` provided with the sits package contains samples from different years covering the same duration. These samples were obtained from the MOD13Q1 product, which contains the same number of images per year. Thus, all time series in the data set `samples_modis_4bands` have the same number of instances.

```
#> # A tibble: 5 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]>
#> 4     -56.0   -10.1  2005-09-14 2006-08-29 Pasture MOD13Q1 <tibble [23 x 5]>
#> 5     -54.6   -10.4  2013-09-14 2014-08-29 Pasture MOD13Q1 <tibble [23 x 5]>
```

Given a suitably built sample file, reading the time series in *sits* is achieved using the `sits_get_data`. This function has two mandatory parameters for CSV and SHP file; (a) `cube` which is the name of the R object that describes the data cube; (b) `file` which is the name of the CSV or SHP file.

```
#> # A tibble: 3 x 7
#>   longitude latitude start_date end_date   label   cube  time_series
#>       <dbl>    <dbl> <date>     <date>     <chr>   <chr> <list>
#> 1     -55.7    -11.8 2013-09-14 2014-08-29 Pasture Sinop <tibble [23 x 3]>
#> 2     -55.6    -11.8 2013-09-14 2014-08-29 Pasture Sinop <tibble [23 x 3]>
#> 3     -55.7    -11.8 2013-09-14 2014-08-29 Forest  Sinop <tibble [23 x 3]>
```

Users can also specify samples by providing shapefiles in point or polygon format. In this case, the geographical location is inferred from the geometries associated with the shapefile. For files containing points, the geographical location is obtained directly; for files with polygon, the parameter `.n_shp_pol` (defaults to 20) determines the number of samples to be extracted from each polygon. The temporal information is inferred from the data cube from which the samples are extracted or can be provided explicitly by the user. The label information is taken from the attribute file associated to the shapefile. The parameter `shp_attr` indicates the name of the column which contains the label to be associated with each time series.

```
#> # A tibble: 3 x 7
#>   longitude latitude start_date end_date   label cube  time_series
#>       <dbl>    <dbl> <date>     <date>     <chr> <chr> <list>
#> 1     -55.6    -11.8 2013-09-14 2014-08-29 <NA>  Sinop <tibble [23 x 3]>
#> 2     -55.6    -11.8 2013-09-14 2014-08-29 <NA>  Sinop <tibble [23 x 3]>
```

```
#> 3     -55.6    -11.8 2013-09-14 2014-08-29 <NA>  Sinop <tibble [23 x 3]>
```

3.5 Filtering techniques for time series

Satellite image time series generally is contaminated by atmospheric influence, geolocation error, and directional effects [6]. 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 variability [7]. Inter-annual climate variability also changes the phenological cycles of the vegetation, resulting in time series whose periods and intensities do not match on a 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 vignette, we discuss filtering techniques to improve time series data that present missing values or noise.

The literature on satellite image time series has 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()`) filters. The first two filters are commonly used in the literature, while the remaining two have been developed by the authors.

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, 7 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, 8 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. 9 found that the 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 previously mentioned filters plus the 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 [10], wavelet decomposition [11], mean-value iteration, ARMD3-ARMA5, and 4253H [12]. 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 [13], which detects deforestation in the Brazilian Amazon rain forest through visual interpretation. This data set

is called `samples_para_mixl8mod` and is provided together with the **sitsdata** package. It has 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 cover 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).

3.5.1 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 are equally spaced, the coefficients have an analytical solution. According to 14, 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 [15]. The optimal value for these two parameters can vary from case to case. In SITS, the user can set the order of the polynomial using the parameter `order` (default = 3), the size of the temporal window with the parameter `length` (default = 5), and the temporal expansion with the parameter `scaling` (default = 1). The following example shows the effect of Savitsky-Golay filter on a point extracted from the MOD13Q1 product, ranging from 2000-02-18 to 2018-01-01.

Notice that the resulting smoothed curve has both desirable and unwanted properties. For the period 2000 to 2008, the Savitsky-Golay filter remove noise resulting from clouds. However, after 2010, when the region has been converted to agriculture, the filter removes an important part of the natural variability from the crop cycle. Therefore, the `length` parameter is arguably too big and results in oversmoothing. Users can try to reduce this parameter and analyse the results.

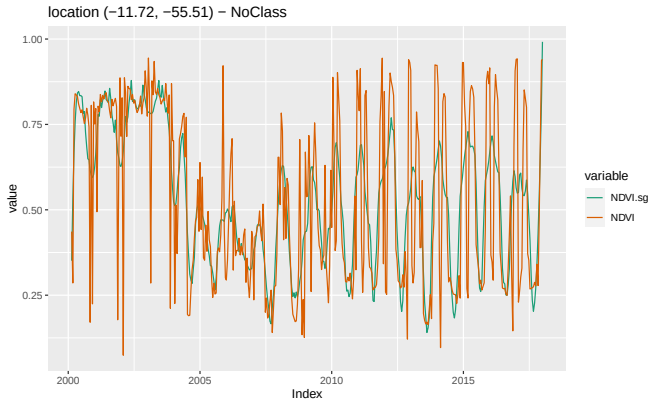


Figure 3.4: Savitzky-Golay filter applied on a multi-year NDVI time series.

3.5.2 Whittaker filter

The Whittaker smoother attempts to fit a curve that represents the raw data, but is penalized if subsequent points vary too much [16]. 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 sum of the squares of the third-order differences of the time series [17], which is given by

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

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 a "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

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

The following example shows the effect of Whitakker filter on a point extracted from the MOD13Q1 product, ranging from 2000-02-18 to 2018-01-01. The `lambda` parameter controls the smoothing of the filter. By default, it is set to 0.5, a small value. For illustrative purposes, we show the effect of a larger smoothing parameter

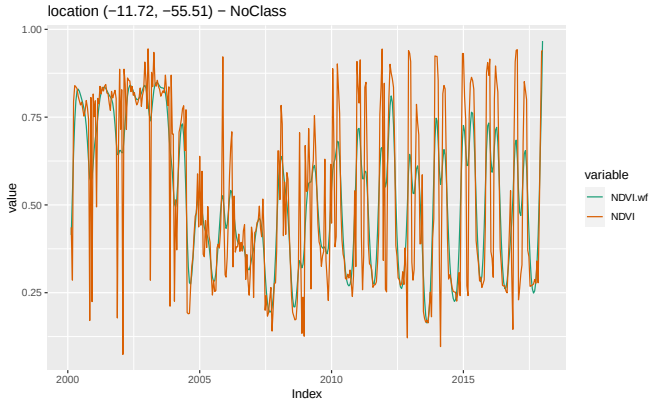


Figure 3.5: Whittaker filter applied on a one-year NDVI time series.

In the same way as what is observed in the Savitsky-Golay filter, high values of the smoothing parameter `lambda` produce an oversmoothed time series that reduces the capacity of the time series to represent natural variations on crop growth. For this reason, low smoothing values are recommended when using the `sits_whittaker` function.

Part II

Clustering

Chapter 4

Time Series Clustering to Improve the Quality of Training Samples

---

One of the key challenges when using samples to train machine learning classification models is assessing their quality. Noisy and imperfect training samples can have a negative effect on classification performance. Therefore, 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. Representative samples lead to good classification maps. `sits` provides support for two clustering methods to test sample quality, which is agglomerative hierarchical clustering (AHC) and self-organizing maps (SOM).

4.1 Clustering for sample quality control

Recent results show that it is feasible to apply machine learning methods to SITS analysis in large areas of 100 million ha or more [18]–[21]. Experience with machine learning methods has established that the limiting factor in obtaining good results is the number and quality of training samples. Large and accurate data sets are better, no matter the algorithm used [22]; increasing the training sample size results in better classification accuracy [23]. Therefore, using machine learning for SITS analysis requires large and good quality training sets.

One of the key challenges when using samples to train machine learning classification models is assessing their quality. Noisy and imperfect training samples

can have a negative effect on classification performance [24]. There are two main sources of noise and errors in satellite image time series. Feature noise is caused by clouds and inconsistencies in data calibration. Class noise occurs when the label assigned to the sample is wrongly attributed. Class noise effects are common on large data sets. In particular, interpreters tend to group samples with different properties in the same category. For this reason, one needs good methods for quality control of large training data sets associated with satellite image time series.

Many factors lead to class noise. One of the main problems is the inherent variability of class signatures in space and time. When training data is collected over a large geographic region, natural variability of vegetation phenology can result in different patterns being assigned to the same label. Phenological patterns can vary spatially across a region and are strongly correlated with climate variations. A related issue is the limitation of crisp boundaries to describe the natural world. Class definition use idealized descriptions (e.g., "a savanna woodland has tree cover of 50% to 90% ranging from 8 to 15 meters in height"). However, in practice, the boundaries between classes are fuzzy and sometimes overlap, making it hard to distinguish between them. Class noise can also result from labeling errors. Even trained analysts can make errors in class attributions. Despite the fact that machine learning techniques are robust to errors and inconsistencies in the training data, quality control of training data can make a significant difference in the resulting maps.

Therefore, 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. Representative samples lead to good classification maps. The package provides support for two clustering methods to test sample quality: (a) Agglomerative Hierarchical Clustering (AHC); (b) Self-organizing Maps (SOM).

4.2 Hierachical clustering for Sample Quality Control

4.2.1 Creating a dendogram

Cluster analysis is used in *sits* as a way to improve training data to feed machine learning classification models by identifying anomalous samples [24]. The package uses *agglomerative hierarchical clustering* (AHC) to 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 [25]. 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 [26].

The main result of the 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 criteria to minimize the data variance by means of either *sum-of-squares* or *sum-of-squares-error* [27]. Ward's intuition is that clusters of multivariate observations, such as time series, should be approximately elliptical in shape [28]. 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()`).

4.2.2 Using a dendrogram to evaluate sample quality

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. 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 the different number of generated clusters. This function returns the height where the cut of the dendrogram maximizes the index.

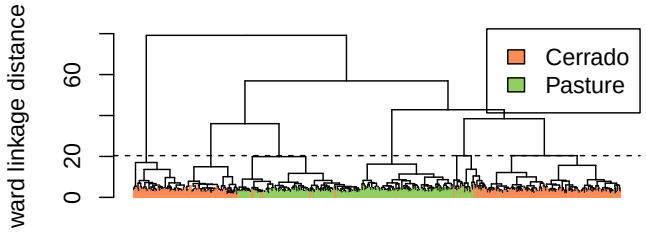
In this example, the height optimizes the ARI and generates 6 clusters. The ARI considers any pair of distinct samples and computes the following counts: (a) the number of distinct pairs whose samples have the same label and are in the same cluster; (b) the number of distinct pairs whose samples have 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 number of samples in labels and clusters are imbalanced [29], which is usually the case.




```
#>
#>              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 have 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 help to identify two kinds of confusion. The first is relative to those small amounts 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, and inadequacy of selected parameters for cluster analysis, or even a natural confusion due to the inherent variability of the land classes.

The result of the `sits_cluster` operation is a `sits_tibble` with one additional column, called "cluster". Thus, it is possible to remove clusters with mixed classes using standard `R` such as those in the `dplyr` package. In the example above, removing cluster 3 can be done using the `dplyr::filter` function.

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

The resulting clusters still contained mixed labels, possibly resulting from outliers. In this case, users may want to remove the outliers and leave only the most frequent class. To do this, one can use `sits_cluster_clean()`, which removes all minority samples, as shown below.

```
#>
```

```
#>             1   2   3   4   5   6 Total
#>   Cerrado 203   0   0  80   0  80   363
#>   Pasture   0 176  28   0 140   0   344
#>   Total   203 176  28  80 140  80   707
```

4.3 Using Self-organizing Maps for Sample Quality

4.3.1 Introduction to Self-organizing Maps

As an alternative for hierarchical clustering for quality control of training samples, SITS provides a clustering technique based on self-organizing maps (SOM). SOM is a dimensionality reduction technique [30], where high-dimensional data is mapped into two dimensions, keeping the topological relations between data patterns. The input data is a set of training samples that are typical of a high dimension. For example, a time series of 25 instances of 4 spectral bands is a 100-dimensional data set. The general idea of SOM-based clustering is that, by projecting the high-dimensional data set of training samples into a 2D map, the units of the map (called "neurons") compete for each sample. It is expected that good quality samples of each class should be close together in the resulting map. The neighbors of each neuron of a SOM map provide information on intra-class and inter-class variability.

The main steps of our proposed method for quality assessment of satellite image time series are shown in the figure below. The method uses self-organizing maps (SOM) to perform dimensionality reduction while preserving the topology of original datasets. Since SOM preserves the topological structure of neighborhoods in multiple dimensions, the resulting 2D map can be used as a set of clusters. Training samples that belong to the same class will usually be neighbors in 2D space. The neighbors of each neuron of a SOM map are also expected to be similar.

As the figure shows, a SOM grid is composed of units called *neurons*. The algorithm computes the distances of each member of the training set to all neurons and finds the neuron closest to the input, called the best matching unit (BMU). The weights of the BMU and its neighbors are updated so as to preserve their similarity [31]. This mapping and adjustment procedure is done in several iterations. At each step, the extent of the change in the neurons diminishes until a convergence threshold is reached. The result is a 2D mapping of the training set, where similar elements of the input are mapped to the same neuron or to nearby ones. The resulting SOM grid combines dimensionality reduction with topological preservation.

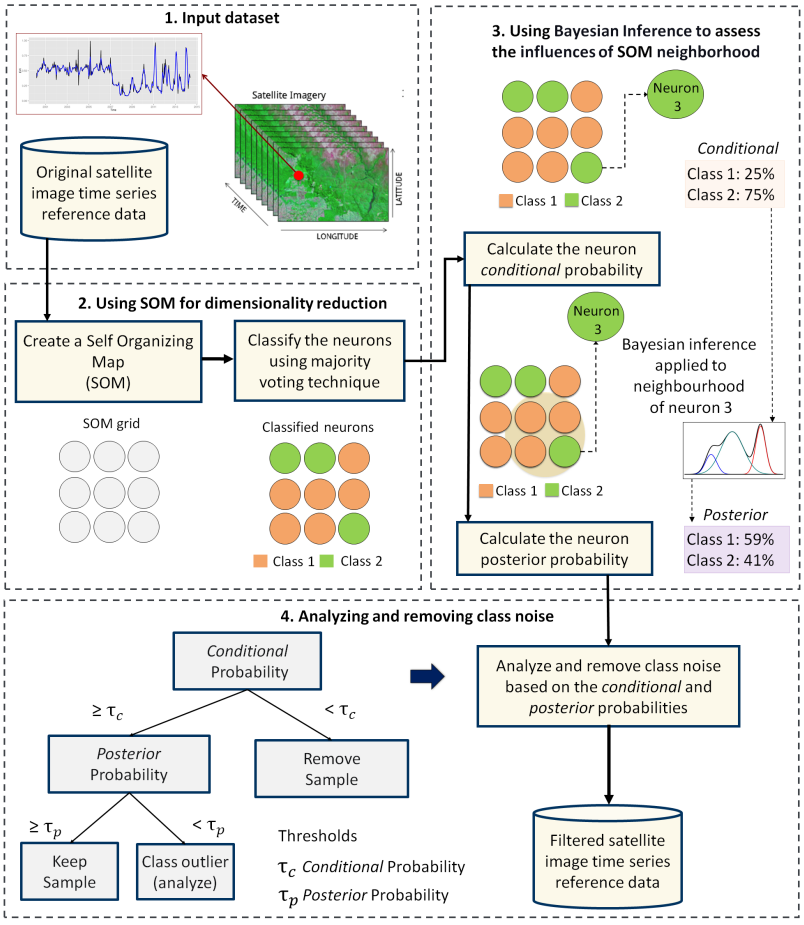


Figure 4.1: Using SOM for class noise reduction

4.3.2 Using SOM for removing class noise

The process of clustering with SOM is done by `sits_som_map()`, which creates a self-organizing map and assesses the quality of the samples. This function uses the "kohonen" R package [32] to compute a SOM grid. Each sample is assigned to a neuron, and neurons are placed in the grid based on similarity. The second step is the quality assessment. Each neuron will be associated with a discrete probability distribution. Homogeneous neurons (those with a single class) are assumed to be composed of good quality samples. Heterogeneous neurons (those with two or more classes with significant probability) are likely to contain noisy samples.

Considering that each sample of the training set is assigned to a neuron, the algorithm computes two values for each sample:

- prior probability: the probability that the label assigned to the sample is correct, considering only the samples in the same neuron. For example, if a neuron has 20 samples, of which 15 are labeled as "Pasture" and 5 as "Forest", all samples labeled "Forest" are assigned a prior probability of 25%. This is an indication that the "Forest" samples in this neuron are not of good quality.
- posterior probability: the probability that the label assigned to the sample is correct, considering the neighboring neurons. Take the case of the above-mentioned neuron whose samples labeled "Pasture" have a prior probability of 75%. What happens if all the neighboring samples have "Forest" as a majority label? Are the samples labeled "Pasture" in this neuron noisy? To answer this question, we use information from the neighbours. Bayesian inference we estimate if these samples are noisy based on the samples of the neighboring neurons [Santos2021].

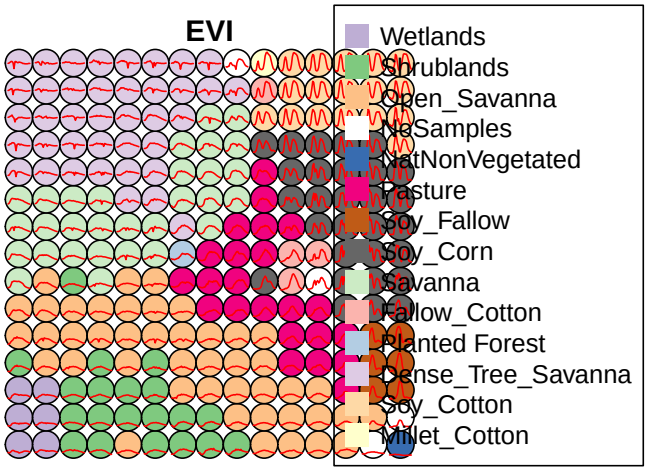
As an example of the use of SOM clustering for quality control of samples, we take a dataset containing a tibble with time series samples for the Cerrado region of Brazil, the second largest biome in South America with an area of more than 2 million km2. The training samples were collected by ground surveys and high-resolution image interpretation by experts from the Brazilian National Institute for Space Research (INPE) team and partners. This set ranges from 2000 to 2017 and includes 61,073 land use and cover samples divided into 14 classes: Natural Non-vegetated, Fallow-Cotton, Millet-Cotton, Soy-Corn, Soy-Cotton, Soy-Fallow, Pasture, Shrublands (in Portuguese "Cerrado Rupestre"), Savanna ("Cerrado"), Dense Tree Savanna ("Cerradão"), Open Savanna ("Campo Cerrado"), Planted Forest, and (14) Wetlands. In the example below, we take only 10% of the samples for faster processing. Users are encouraged to run the example with the full set of samples.

The output of the `sits_som_map` is a list with 4 tibbles:

- the original set of time series with two additional columns for each time series: `id_sample` (the original id of each sample) and `id_neuron` (the id

of the neuron to which it belongs).

- a tibble with information on the neuron. For each neuron, it gives the prior and posterior probabilities of all labels which occur in the samples assigned to it.
- the SOM grid To plot the SOM grid, use `plot()`. The neurons are labelled using the majority voting.



Looking at the SOM grid, one can see that most of the neurons of a class are located close to each other. There are outliers, e.g., some "Open Savanna" neurons are located amidst "Shrublands" neurons. This mixture is a consequence of the continuous nature of natural vegetation cover in the Brazilian Cerrado. The transition between areas of open savanna and shrublands is not always well defined; moreover, it is dependent on factors such as climate and latitude.

To identifies noisy samples, we take the result of the `sits_som_map` function as the first argument to the function `sits_som_clean_samples`. This function finds out which samples are noisy, those that are clean, and some that need to be further examined by the user. It uses the `prior_threshold` and `posterior_threshold` parameters according to the following rules:

- If the prior probability of a sample is less than `prior_threshold`, the sample is assumed to be noisy and tagged as "remove";
- If the prior probability is greater or equal to `prior_threshold` and the posterior probability is greater or equal to `posterior_threshold`, the sample is assumed not to be noisy and thus is tagged as "clean";
- If the prior probability is greater or equal to `prior_threshold` and the posterior probability is less than `posterior_threshold`, we have a situation the sample is part of the majority level of those assigned to its neuron, but its label is not consistent with most of its neighbors. This is an anomalous condition and is tagged as "analyze". Users are encouraged to inspect such samples to find out whether they are in fact noisy or not.

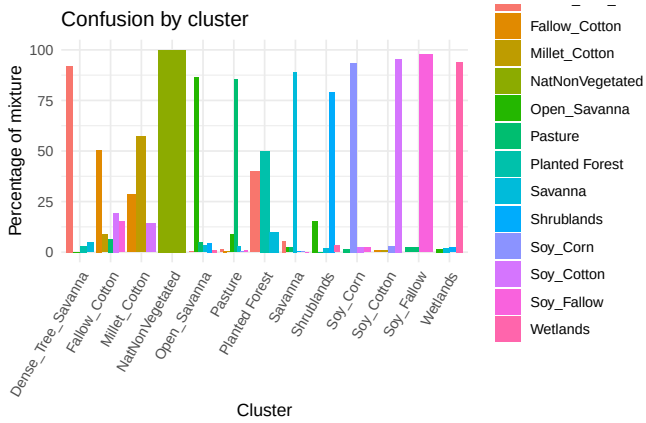
The default value for both `prior_threshold` and `posterior_threshold` is 60%. The `sits_som_clean_samples` has an additional parameter (`keep`) which indicates which samples should be kept in the set based on their prior and posterior probabilities of being noisy and the assigned label. The default value for `keep` is `c("clean", "analyze")`. As a result of the cleaning, about 900 samples have been considered to be noisy and thus removed.

```
#> # A tibble: 2 x 2
#>   eval    count
#>   <chr>   <int>
#> 1 analyze   652
#> 2 clean    4416
```

4.3.3 Comparing Global Accuracy of Original and Clean Samples

To compare the accuracy of the original and clean samples, we run a 5-fold validation on the original and on the cleaned sample. We use the function `sits_kfold_validate`. As the results show, the SOM procedure is useful, since the global accuracy improves from 91% to 95%.

An additional way of evaluating the quality of samples is to examine the internal mixture inside neurons with the same label. We call a group of neurons sharing the same label as a "cluster". Given a SOM map, the function `sits_som_evaluate_cluster` examines all clusters to find out the percentage of samples contained in it which do not share its label. This information is saved as a tibble and can also be visualized.



4.4 Conclusion

Machine learning methods are now established as a useful technique for remote sensing image analysis. Despite the well-known fact that the quality of the

training data is a key factor in the accuracy of the resulting maps, the literature on methods for detecting and removing class noise in SITS training sets is limited. To contribute to solving this challenge, this paper proposed a new technique. The proposed method uses the SOM neural network to group similar samples in a 2D map for dimensionality reduction. The method identifies both mislabeled samples and outliers that are flagged to further investigation. The results demonstrate the positive impact on the overall classification accuracy. Although the class noise removal adds an extra cost to the entire classification process, we believe that it is essential to improve the accuracy of classified maps using SITS analysis mainly for large areas.

Part III

Classification

Chapter 5

Machine Learning for Data Cubes using the SITS package

This chapter presents the machine learning techniques available in SITS. The main use for machine learning in SITS is for classification of land use and land cover. These machine learning methods available in SITS include linear and quadratic discrimination analysis, support vector machines, random forests, deep learning and neural networks.

5.1 Machine learning classification

There has been much recent interest in using classifiers such as support vector machines [33] and random forests [34] for remote sensing images. Most often, researchers use a *space-first, time-later* approach. 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 [35], [36]. Using software such as TIMESAT [37], they derive a small set of phenological parameters from vegetation indexes, like the beginning, peak, and length of the growing season [38], [39].

Most studies using satellite image time series for land cover classification use a *space-first, time-later* approach. For multiyear studies, most researchers first derive best-fit yearly composites and then classify each composite image [40]. As

an alternative, the *sits* package provides support for the classification of time series, preserving the full temporal resolution of the input data, using a *time-first, space-later* approach. The idea is to have as many temporal attributes as possible, increasing the classification space's dimension. Each temporal instance of a time series is taken as an independent dimension in the classifier's feature space. 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 provides filtering, clustering, and post-processing methods that have not been published in the literature.

Current approaches to image time series analysis still use a limited number of attributes. A common approach is deriving a small set of phenological parameters from vegetation indices, like the beginning, peak, and length of growing season [35], [36], [38], [39]. These phenological parameters are then fed in specialized classifiers such as TIMESAT [37]. These approaches do not use the power of advanced statistical learning techniques to work on high-dimensional spaces with big training data sets [41]. Package *sits* uses the full depth of satellite image time series to create larger dimensional spaces, an approach we consider to be more appropriate to use with machine learning.

When used in *time-first, space-later* approach, `sits` treats time series as a feature vector. It aligns all time series from different years to a single yearly span. 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 package has support for a variety of machine learning techniques: linear discriminant analysis, quadratic discriminant analysis, multinomial logistic regression, random forests, boosting, support vector machines, and deep learning. The deep learning methods include multi-layer perceptrons, 1D convolution neural networks and mixed approaches such as TempCNN [2]. The following machine learning methods are available in SITS:

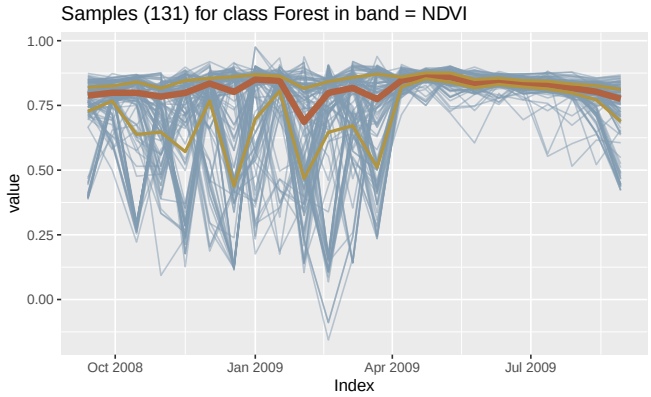
- Linear discriminant analysis (`sits_lda`)
- Quadratic discriminant analysis (`sits_qda`)
- Multinomial logit and its variants 'lasso' and 'ridge' (`sits_mlr`)
- Support vector machines (`sits_svm`)
- Random forests (`sits_rfor`)
- Extreme gradient boosting (`sits_xgboost`)
- Deep learning (DL) using multi-layer perceptrons (`sits_mlp`)
- DL using Deep Residual Networks (`sits_ResNet`)
- DL combining 1D convolutional neural networks and multi-layer perceptrons (`sits_TempCNN`)

For the machine learning examples, we use the data set "samples_matogrosso_mod13q1",

containing a sits tibble with time series samples from Brazilian Mato Grosso State (Amazon and Cerrado biomes), obtained from the MODIS MOD13Q1 product. The tibble with 1,892 samples and 9 classes ("Cerrado", "Fallow_Cotton", "Forest", "Millet_Cotton", "Pasture","Soy_Corn", "Soy_Cotton", "Soy_Fallow", "Soy_Millet"). Each time series comprehends 12 months (23 data points) with 6 bands ("NDVI", "EVI", "BLUE", "RED", "NIR", "MIR") . The dataset was used in the paper "Big Earth observation time series analysis for monitoring Brazilian agriculture" [18], and is available in the R package "sitsdata", which is downloadable from the website associated to the "e-sensing" project.

5.2 Visualizing Samples

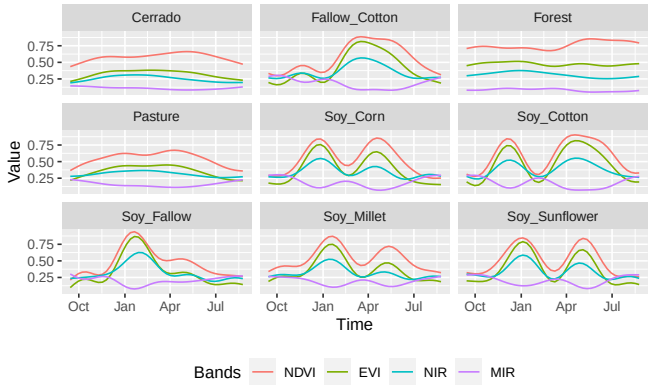
One useful way of describing and understanding the samples is by plotting them. A direct way of doing so is using the `plot` function, as discussed in Chapter 3. In the plot, the thick red line is the median value for each time instance and the yellow lines are the first and third interquartile ranges.



An alternative to visualise the samples is to estimate a statistical approximation to an idealized pattern based on a generalised additive model (GAM). A GAM is a linear model in which the linear predictor depends linearly on a smooth function of the predictor variables

$$y = \beta_i + f(x) + \epsilon, \epsilon \sim N(0, \sigma^2).$$

The function `sits_patterns` uses a GAM to predict a smooth, idealized approximation to the time series associated to the each label, for all bands. This function is based on the R package `dtwSat`[42], which implements the TWDTW time series matching method described in 43. The resulting patterns can be viewed using `plot`.



The resulting plots provide some insights over the time series behaviour of each class. While the response of the "Forest" class is quite distinctive, there are similarities between the double-cropping classes ("Soy-Corn", "Soy-Millet", "Soy-Sunflower" and "Soy-Corn") and between the "Cerrado" and "Pasture" classes. This could suggest that additional information, more bands, or higher-resolution data could be considered to provide a better basis for time series samples that can better distinguish the intended classes. Despite these limitations, the best machine learning algorithms can provide good performance even in the above case.

5.3 Common interface to machine learning and deeplearning models

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, in Chapter 6 we disscuss how to classify data cubes.

When a dataset of time series organised as a SITS tibble is taken as input to the classifier, the result is the same tibble with one additional column ("predicted"), which contains the information on what labels are have been assigned for each interval. The results can be shown in text format using the function `sits_show_prediction` or graphically using `plot`.

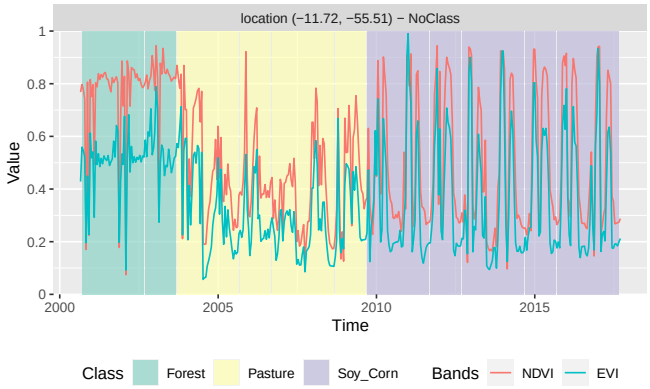
5.4 Random forests

The Random forest uses the idea of *decision trees* 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 criterion is used to identify relevant features and to perform variable selection. This decreases the correlation among trees and improves the prediction performance. Two often-used impurity criteria are the *Gini* index and the *permutation* measure. The Gini index considers the contribution of each variable which improves the spliting criteria for building tress. Permutation increases the importance of variables that have a positive effect on the prediction accuracy. The splitting process continues until the tree reaches some given minimum nodes size or a minimum impurity index value.

One of the advantages of the random forest model is that the classification performance is mostly dependent on the number of decision trees to grow and of the "importance" parameter, which controls the purity variable importance measures. SITS provides a `sits_rfor` function which is a front-end to the `randomForest` package[44]; its main parameters is `num_trees` (number of trees to grow).




```
#> # 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
```

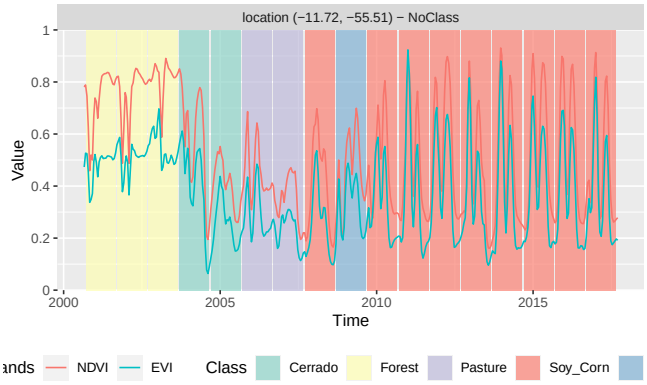
Random forest classifier are robust to outliers and able to deal with irrelevant inputs [45]. However, despite being robust, random forest tend to overemphasize some variables and thus rarely turn out to be the classifier with the smallest error. One reason is that the performance of random forest tends to stabilise after a part of the trees are grown [45]. Random forest classifiers can be quite useful to provide a baseline to compare with more sophisticated methods.

5.5 Support Vector Machines

Given a multidimensional data set, the Support Vector Machine (SVM) method finds an optimal separation hyperplane that minimizes misclassifications [46]. Hyperplanes are linear $(p-1)$-dimensional boundaries and define linear partitions in the feature space. 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.

For data that is not linearly separable, SVM includes kernel functions that map the original feature space into a higher dimensional space, providing nonlinear boundaries to the original 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. 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. SVM is one of the most widely used algorithms in machine learning applications and has been widely applied to classify remote sensing data [33].

In `sits`, SVM is implemented as a wrapper of `e1071` R package that uses the `LIBSVM` implementation [47], `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. Considering that SVM is not a robust to outliers as Random Forest, we apply a whittaker filter to smoothen the data by a small factor.



The result is mostly consistent of what one could expect by visualising the time series. The area started out as a forest in 2000, it was deforested from 2004 to 2005, used as pasture from 2006 to 2007, and for double-cropping agriculture from 2008 onwards. However, the result shows some inconsistencies. First, since the training dataset does not contain a samples of deforested areas, places where forest is removed will tend to be classified as "Cerrado", which is the nearest kind of vegetation cover where trees and grasslands are mixed. This misinterpretation needs to be corrected in post-processing by applying a time-dependent rule (see the main SITS vignette and the post-processing methods vignette). Also, the classification for year 2009 is "Soy-Millet", which is different from the "Soy-Corn" label assigned from the other years from 2008 to 2017. To test if this result is inconsistent, one could apply spatial post-processing techniques, as discussed in the main SITS vignette and in the post-processing one.

One of the drawbacks of using the `sits_svm` method is its sensitivity to its parameters. Using a linear or a polynomial kernel fails to produce good results. If one varies the parameter `cost` (cost of constraints violation) from 100 to 1, the results can be strinkgly different. Such sensitivity to the input parameters points to a limitation when using the SVM method for classifying time series.

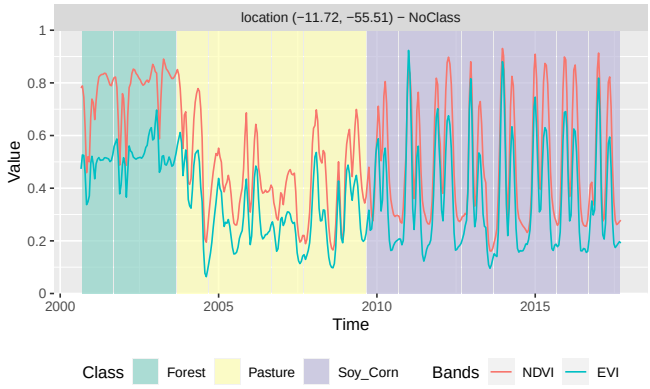
5.6 Extreme Gradient Boosting

Boosting techniques are based on the idea of starting from a weak predictor and then improving performance sequentially by fitting better model at each iteration. It starts by fitting a simple classifier to the training data. Then it uses the residuals of the regression to build a better prediction. Typically, the base classifier is a regression tree. Although both random forests and boosting use trees for classification, there is an important difference. In the random forest classifier, the same random logic for tree selections is applied at every step. Boosting trees are built to improve on previous result, by applying finer divisions that improve the performance [45]. The performance of random forests generally increases with the number of trees until it becomes stable; however, the number of trees grown by boosting techniques cannot be too large, at the

risk of overfitting the model.

Gradient boosting is a variant of boosting methods where the cost function is minimized by gradient descent algorithm. Extreme gradient boosting [48], called "XGBoost", improves by using an efficient approximation to the gradient loss function. The algorithm is fast and accurate. XGBoost is considered one of the best statistical learning algorithms available and has won many competitions; it is generally considered to be better than SVM and random forests. However, actual performance is controlled by the quality of the training dataset.

In SITS, the XGBoost method is implemented by the `sits_xbgoost()` function, which is based on "XGBoost" R package and has five parameters that require tuning. The learning rate `eta` varies from 0 to 1, but should be kept small (default is 0.3) to avoid overfitting. The minimim loss value `gamma` specifies the minimum reduction required to make a split. Its default is 0, but increasing it makes the algorithm more conservative. The maximum depth of a tree `max_depth` controls how deep tress are to be built. In principle, it should not be largem since higher depth trees lead to overfitting (default is 6.0). The `subsample` parameter controls the percentage of samples supplied to a tree. Its default is 1 (maximum). Setting it to lower values means that xgboost randomly collects only part of the data instances to grow trees, thus preventing overfitting. The `nrounds` parameters controls the maximum number of boosting interactions; its default is 100, which has proven to be sufficient in the SITS. In order to follow the convergence of the algorithm, users can turn the `verbose` parameter on.



In general, the results from the extreme gradient boosting model are similar to the Random Forest model. However, for each specific study, users need to perform validation. See the function `sits_kfold_validate` for more details.

5.7 Deep learning using multi-layer perceptrons

Using the `keras` package [49] as a backend, SITS supports the following =deep learning techniques, as described in this section and the next ones. The first method is that of feedforward neural networks, or multi-layer perceptron (MLPs).

These are the quintessential deep learning models. The goal of a multilayer perceptrons 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 intermediate 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 [50].

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.

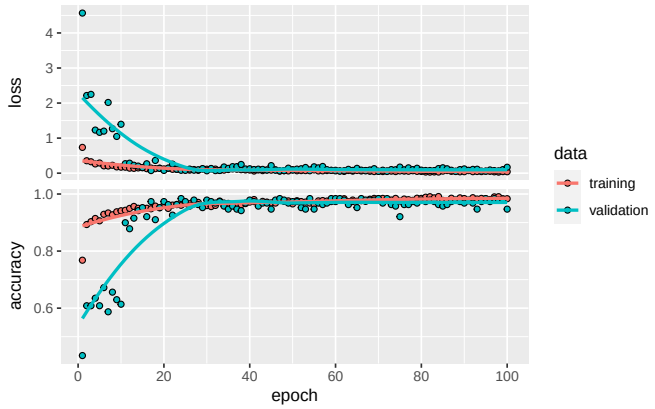
Three other important parameters for an MLP are: (a) the activation function; (b) the optimization method; (c) the dropout rate. The activation function the activation function of a node defines the output of that node given an input or set of inputs. Following standard practices [50], we recommend the 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 [51]. 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 MLP classifiers. Dropout is a technique for randomly dropping units from the neural network during training [52]. 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 [50], this procedures prevents an early convergence of the optimization to a local minimum. We suggest that users experiment with different dropout rates.

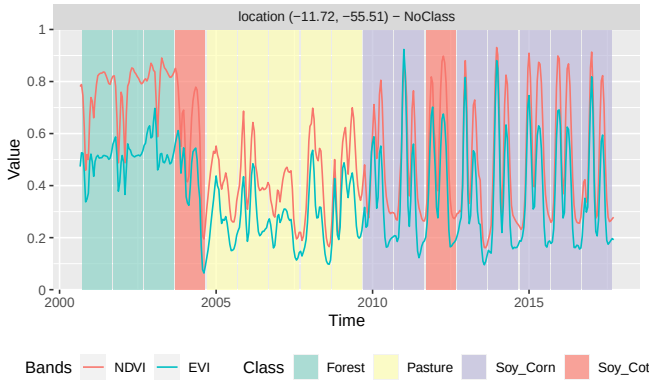
In the following example, we classify the same data set using an example of the `deep learning` method. The parameters for the MLP have been chosen based on the proposal by [53] to take multilayer perceptrons as the baseline for time

series classifications : (a) Four layers with 512 neurons each, specified by the parameter `layers`; (b) Using the 'elu' activation function; (c) dropout rates of 10%, 20%, 20%, and 30% for the layers; (d) the "optimizer_adam" as optimizer (default value); (e) a number of training steps (`epochs`) of 150; (f) a `batch_size` of 64, which indicates how many time series are used for input at a given steps; (g) a validation percentage of 20%, which means 20% of the samples will be randomly set side for validation. In practice, users may want to increase the number of epochs and the number of layers. In our experience, if the training dataset is of good quality, using 3 to 5 layers is a reasonable compromise. Further increase on the number of layers will not improve the model. To simplify the vignette generation, the `verbose` option has been turned off. The default value is on. After the model has been generated, we plot its training history.

In this and in the following examples of using deep learning classifiers, both the training samples and the point to be classified are filtered with `sits_whittaker` with a small smoothing parameter (lambda = 0.5). In our experiments, we found that deep learning classifiers are not as robust to noise as Random Forest or XGBoost. Thus, the right amount of smoothing appears to improve their detection power.



Then, we classify a 16-year time series using the DL model



Compared to the Random Forest and XGBoost models, the deep learning model captures more subtle changes. For example, the transition from Forest to Pasture as estimated by the model is not abrupt, but takes more than one year. In the year 2004, the time series corresponds to a degraded forest. Since there are no samples for "Forest Degradation", the model assigns this series to class that is neither "Forest" nor "Pasture". This indicates that users should include samples of "Forest Degradation" to improve classification. Moreover, while the RandomForest and XGBoost models consider that the agricultural production in the area started only in 2010, the MLP model indicates an earlier starting date of 2008. Although the model mixes the "Soy_Corn" and "Soy_Millet", the distinction between the classes is quite subtle, and thus indicates the need to improve the number of samples. Thus, in a first and coarse appreciation, the MLP model shows an increased sensitivity to the data variation than the previous models.

5.8 Combined 1D CNN and multi-layer perceptron networks

Convolutional neural networks (CNN) are a variety of deep learning methods where a convolution filter (sliding window) is applied to the input data. In the case of time series, a 1D CNN works by applying a moving window to the series. Using convolution filters is a way to incorporate temporal autocorrelation information in the classification. The result of the convolution is another time series. 54 states that the use of 1D-CNN for time series classification improves on the use of multi-layer perceptrons, since the classifier is able to represent temporal relationships. Also, 1D-CNNs with a suitable convolution window make the classifier more robust to moderate noise, e.g. intermittent presence of clouds.

The combination of 1D CNNs and multi-layer perceptron models for satellite image time series classification is proposed in 2. The so-called "tempCNN" architecture consists of a number of 1D-CNN layers, similar to the fullCNN model discussed above, whose output is fed into a set of multi-layer perceptrons. The original tempCNN architecture is composed of three 1D convolutional layers (each with 64 units), one dense layer of 256 units and a final softmax layer for classification (see figure). The kernel size of the convolution filters is set to 5. The authors use a combination of different methods to avoid overfitting and reduce the vanishing gradient effect, including dropout, regularization, and batch normalisation. In the tempCNN paper [2], the authors compare favourably the tempCNN model with the Recurrent Neural Network proposed by 55 for land use classification. The figure below shows the architecture of the tempCNN model.

The function `sits_tempCNN` implements the model. The code has been derived from the Python source provided by the authors (https://github.com/charlotte-

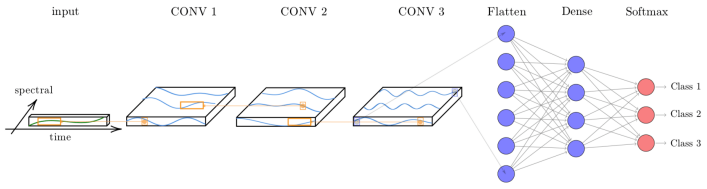
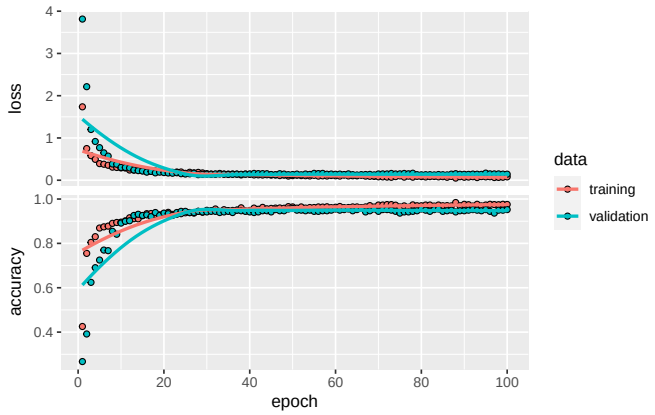
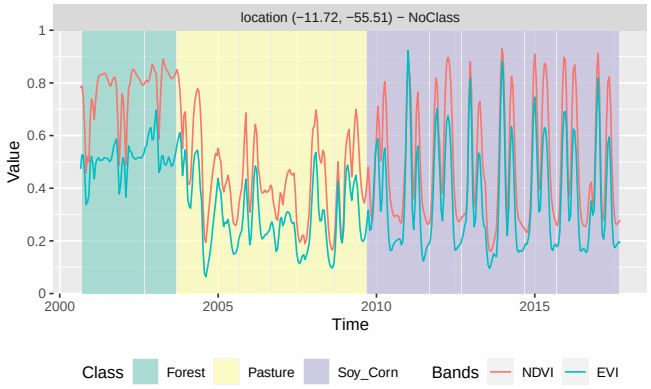


Figure 5.1: Structure of tempCNN architecture (source: Pelletier et al.(2019))

pel/temporalCNN). The parameter `cnn_layers` controls the number of 1D-CNN layers and the size of the filters applied at each layer; the parameter `cnn_kernels` indicates the size of the convolution kernels. Activation, regularisation for all 1D-CNN layers are set, respectively, by the `cnn_activation`, `cnn_L2_rate`. The dropout rates for each 1D-CNN layer are controlled individually by the parameter `cnn_dropout_rates`. The parameters `mlp_layers` and `mlp_dropout_rates` allow the user to set the number and size of the desired MLP layers, as well as their dropout_rates. The activation of the MLP layers is controlled by `mlp_activation`. By default, the function uses the ADAM optimizer, but any of the optimizers available in the `keras` package can be used. The `validation_split` controls the size of the test set, relative to the full data set. We recommend to set aside at least 20% of the samples for validation. Based on our experiments, in the example below we propose a different setup of parameters than 2. The parameters are: (a) filters of size 3, which aim for a local convolution that reduces noise but do not decrease natural variability; (b) smaller dropout rates ranging from 10% to 35%; (d) an increased number of perceptron layers of size 512, also with smaller dropout rates.



Then, we classify a 16-year time series using the TempCNN model



While the result of the TempCNN model using default parameters is similar to that of the MLP model, it has has the potential to better explore the time series data than the the MLP model. Given that it has more parameters, it requires more effort to calibrate them. Users interested in working with this models are encouraged to compare different settings to define what is the best configuration for their problems.

5.9 Residual 1D CNN Networks (ResNet)

The Residual Network (ResNet) is a 1D convolution neural network (CNN) proposed by 53. ResNet is composed of 11 layers (see figure below). By default, ResNet is divided in three blocks of three 1D CNN layers each. Each block corresponds to a 1D CNN architecture. The output of each block is combined with a shortcut that links its output to its input. The idea is avoid the so-called "vanishing gradient", which occurs when a deep learning network is trained based gradient optimization methods[56]. As the networks get deeper, optimizing them becomes more difficult. Including the input layer of the block at its end is a heuristic that has shown to be effective.

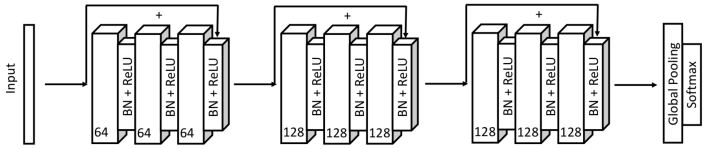
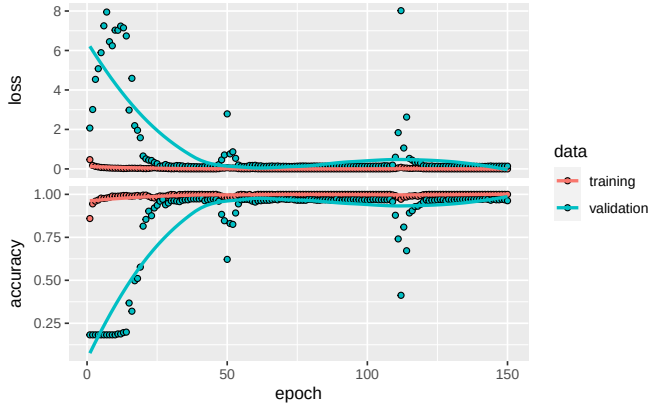


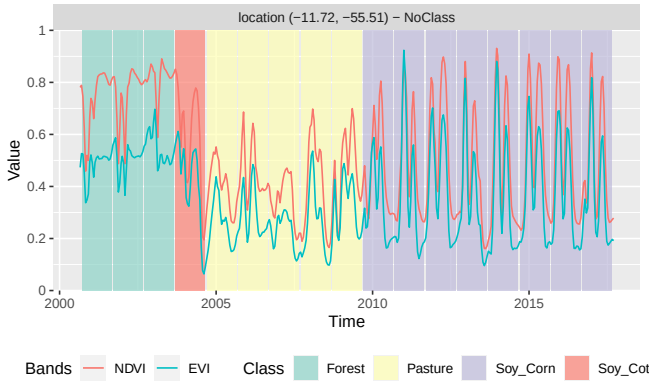
Figure 5.2: Structure of ResNet architecture (source: Wang et al.(2017))

In SITS, the ResNet is implemented using the `sits_resnet` function. The default parameters are those proposed by 53, and we also benefited from the code provided by 57 (https://github.com/hfawaz/dl-4-tsc). The first parameter is `blocks`, which controls the number of blocks and the size of filters in each block. By default, the model implements three blocks, the first with 64 filters

and the others with 128 filters. Users can control the number of blocks and filter size by changing this parameter. The parameter `kernels` controls the size the of kernels of the three layers inside each block. We have found out that it is useful to experiment a bit with these kernel sizes in the case of satellite image time series. The default activation is "relu", which is recommended in the literature to reduce the problem of vanishing gradients. The default optimizer is the same as proposed in 53 and 57.



Then, we classify a 16-year time series using the ResNet model.



In a similar way as other deep learning models, ResNet finds an outlier in year 2004. As explained above, the time series for 2004 is likely to come from a situation where part of the natural forest had been removed, but not all of the area had been replaced by pasture. Since the training set does not include time series for degraded forests, the classifier assigns it to a crop class. Although incorrect, such labeling is useful to point out to missing classes in the training set. Also similarly to the MLP model, the RestNet model considers that the agricultural production in the area started in 2008.

Chapter 6

Classification of Images in Data Cubes using Satellite Image Time Series

---

This chapter shows the use of the SITS package for classification of satellite images that are associated to Earth observation data cubes.

6.1 Data cube classification

To classify a data cube, use the function `sits_classify()`. This function works in the same for all types of data cubes, regardless of origin. 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. 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.

To reduce processing time, it is necessary to adjust `sits_classify()` according to the capabilities of the host environment. There is a 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 `memsize` parameter controls the size of the main memory (in GBytes) to be used for classification. A practical approach is to set `memsize` to about 30% of the total memory available.

When using the `sits_rfor` and `sits_svm` model, users can also specify the number of cores to be used for parallel processing by setting the parameter `multicores`. In this case, the classification task it is split into different cores.

6.2 Processing time estimates

Processing time depends on the data size and the model used. Some estimates derived from experiments made the authors show that:

1. Classification of one year of the entire Cerrado region of Brazil (2,5 million $km\hat{}2$) using 18 tiles of CBERS-4 AWFI images (64 meter resolution), each tile consisting of 10,504 x 6,865 pixels with 24 time instances, using 4 spectral bands, 2 vegetation indexes and a cloud mask, resulting in 1,7 TB, took 16 hours using 100 GB of memory and 20 cores of a virtual machine. The classification was done with a random forest model with 2000 trees.

2. Classification of one year in one tile of LANDSAT-8 images (30 meter resolution), each tile consisting of 11,204 x 7,324 pixels with 24 time instances, using 7 spectral bands, 2 vegetation indexes and a cloud mask, resulting in 157 GB, took 90 minutes using 100 GB of memory and 20 cores of a virtual machine. The classification was done with a random forest model with 2000 trees.

Part IV

Post classification

Chapter 7

Post classification smoothing

This chapter describes the methods available for spatial smoothing of the results of machine learning classifications.

7.1 Introduction

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.

Image classification post-processing has been defined as "a refinement of the labelling in a classified image in order to enhance its classification accuracy" [58]. In remote sensing image analysis, these procedures are used to combine pixel-based classification methods with a spatial post-processing method to remove outliers and misclassified pixels. For pixel-based classifiers, post-processing methods enable the inclusion of spatial information in the final results.

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" [59]: 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.

The following spatial smoothing methods are available in **sits**: bayesian smoothing, gaussian smoothing and bilinear smoothing. These methods are called using the `sits_smooth` function, as shown in the examples below.

7.2 Bayesian smoothing

Bayesian inference can be thought of as way of coherently updating our uncertainty in the light of new evidence. It allows the inclusion of expert knowledge on the derivation of probabilities. In a Bayesian context, probability is taken as a subjective belief. The observation of the class probabilities of each pixel is taken as our initial belief on what the actual class of the pixel is. We then use Bayes' rule to consider how much the class probabilities of the neighbouring pixels affect our original belief. In the case of continuous probability distributions, Bayesian inference is expressed by the rule:

$$\pi(\theta|x) \propto \pi(x|\theta)\pi(\theta)$$

Bayesian inference involves the estimation of an unknown parameter θ, which is the random variable that describe what we are trying to measure. In the case of smoothing of image classification, θ is the class probability for a given pixel. We model our initial belief about this value by a probability distribution, $\pi(\theta)$, called the *prior* distribution. It represents what we know about θ *before* observing the data. The distribution $\pi(x|\theta)$, called the *likelihood*, is estimated based on the observed data. It represents the added information provided by our observations. The *posterior* distribution $\pi(\theta|x)$ is our improved belief of θ *after* seeing the data. Bayes's rule states that the *posterior* probability is proportional to the product of the *likelihood* and the *prior* probability.

7.2.1 Derivation of bayesian parameters for spatiotemporal smoothing

In our post-classification smoothing model, we consider the output of a machine learning algorithm that provides the probabilities of each pixel in the image to belong to target classes. More formally, consider a set of K classes that are candidates for labelling each pixel. Let $p_{i,t,k}$ be the probability of pixel i

belonging to class k, $k = 1, \ldots, K$ at a time t, $t = 1, \ldots, T$. We have

$$\sum_{k=1}^{K} p_{i,t,k} = 1, p_{i,t,k} > 0$$

We label a pixel p_i as being of class k if

$$p_{i,t,k} > p_{i,t,m}, \forall m = 1, \ldots, K, m \neq k$$

For each pixel i, we take the odds of the classification for class k, expressed as

$$O_{i,t,k} = p_{i,t,k}/(1 - p_{i,t,k})$$

where $p_{i,t,k}$ is the probability of class k at time t. We have more confidence in pixels with higher odds since their class assignment is stronger. There are situations, such as border pixels or mixed ones, where the odds of different classes are similar in magnitude. We take them as cases of low confidence in the classification result. To assess and correct these cases, Bayesian smoothing methods borrow strength from the neighbors and reduces the variance of the estimated class for each pixel.

We further make the transformation

$$x_{i,t,k} = \log[O_{i,t,k}]$$

which measures the *logit* (log of the odds) associated to classifying the pixel i as being of class k at time t. The support of $x_{i,t,k}$ is $\mathbb{R}$. We can express the pixel data as a K-dimensional multivariate logit vector

$$\mathbf{x}_{i,t} = (x_{i,t,k_0}, x_{i,t,k_1}, \ldots, x_{i,t,k_K})$$

For each pixel, the random variable that describes the class probability k at time t is denoted by $\theta_{i,t,k}$. This formulation allows uses to use the class covariance matrix in our formulations. We can express Bayes' rule for all combinations of pixel and classes for a time interval as

$$\pi(\boldsymbol{\theta}_{i,t}|\mathbf{x}_{i,t}) \propto \pi(\mathbf{x}_{i,t}|\boldsymbol{\theta}_{i,t})\pi(\boldsymbol{\theta}_{i,t}).$$

We assume the conditional distribution $\mathbf{x}_{i,t}|\boldsymbol{\theta}_{i,t}$ follows a multivariate normal distribution

$$[\mathbf{x}_{i,t}|\boldsymbol{\theta}_{i,t}] \sim \mathcal{N}_K(\boldsymbol{\theta}_{i,t}, \boldsymbol{\Sigma}_{i,t}),$$

where $\boldsymbol{\theta}_{i,t}$ is the mean parameter vector for the pixel i at time t, and $\boldsymbol{\Sigma}_{i,t}$ is a known $k \times k$ covariance matrix that we will use as a parameter to control the level of smoothness effect. We will discuss later on how to estimate $\boldsymbol{\Sigma}_{i,t}$. To

model our uncertainty about the parameter $\boldsymbol{\theta}_{i,t}$, we will assume it also follows a multivariate normal distribution with hyper-parameters $\mathbf{m}_{i,t}$ for the mean vector, and $\mathbf{S}_{i,t}$ for the covariance matrix.

$$[\boldsymbol{\theta}_{i,t}] \sim \mathcal{N}_K(\mathbf{m}_{i,t}, \mathbf{S}_{i,t}).$$

The above equation defines our prior distribution. The hyper-parameters $\mathbf{m}_{i,t}$ and $\mathbf{S}_{i,t}$ are obtained by considering the neighboring pixels of pixel i. The neighborhood can be defined as any graph scheme (e.g. a given Chebyshev distance on the time-space lattice) and can include the referencing pixel i as a neighbor. Also, it can make no reference to time steps other than t defining a space-only neighborhood. More formally, let

$$\mathbf{V}_{i,t} = \{\mathbf{x}_{i_j,t_j}\}_{j=1}^{N},$$

denote the N logit vectors of a spatiotemporal neighborhood N of pixel i at time t. Then the prior mean is calculated by

$$\mathbf{m}_{i,t} = \mathrm{E}[\mathbf{V}_{i,t}],$$

and the prior covariance matrix by

$$\mathbf{S}_{i,t} = \mathrm{E}\left[(\mathbf{V}_{i,t} - \mathbf{m}_{i,t})\,(\mathbf{V}_{i,t} - \mathbf{m}_{i,t})^{\intercal}\right].$$

Since the likelihood and prior are multivariate normal distributions, the posterior will also be a multivariate normal distribution, whose updated parameters can be derived by applying the density functions associated to the above equations. The posterior distribution is given by

$$[\boldsymbol{\theta}_{i,t}|\mathbf{x}_{i,t}] \sim \mathcal{N}_K\left((\mathbf{S}_{i,t}^{-1} + \boldsymbol{\Sigma}^{-1})^{-1}(\mathbf{S}_{i,t}^{-1}\mathbf{m}_{i,t} + \boldsymbol{\Sigma}^{-1}\mathbf{x}_{i,t}), (\mathbf{S}_{i,t}^{-1} + \boldsymbol{\Sigma}^{-1})^{-1}\right).$$

The $\boldsymbol{\theta}_{i,t}$ parameter model is our initial belief about a pixel vector using the neighborhood information in the prior distribution. It represents what we know about the probable value of $\mathbf{x}_{i,t}$ (and hence, about the class probabilities as the logit function is a monotonically increasing function) *before* observing it. The *likelihood* function $P[\mathbf{x}_{i,t}|\boldsymbol{\theta}_{i,t}]$ represents the added information provided by our observation of $\mathbf{x}_{i,t}$. The *posterior* probability density function $P[\boldsymbol{\theta}_{i,t}|\mathbf{x}_{i,t}]$ is our improved belief of the pixel vector *after* seeing $\mathbf{x}_{i,t}$.

At this point, we are able to infer a point estimator $\hat{\boldsymbol{\theta}}_{i,t}$ for the $\boldsymbol{\theta}_{i,t}|\mathbf{x}_{i,t}$ parameter. For the multivariate normal distribution, the posterior mean minimises not only the quadratic loss but the absolute and zero-one loss functions. It can be taken

from the updated mean parameter of the posterior distribution (Eq.**??**) which, after some algebra, can be expressed as

$$\hat{\boldsymbol{\theta}}_{i,t} = \mathrm{E}[\boldsymbol{\theta}_{i,t}|\mathbf{x}_{i,t}] = \boldsymbol{\Sigma}_{i,t}\left(\boldsymbol{\Sigma}_{i,t} + \mathbf{S}_{i,t}\right)^{-1}\mathbf{m}_{i,t} + \mathbf{S}_{i,t}\left(\boldsymbol{\Sigma}_{i,t} + \mathbf{S}_{i,t}\right)^{-1}\mathbf{x}_{i,t}.$$

The estimator value for the logit vector $\hat{\boldsymbol{\theta}}_{i,t}$ is a weighted combination of the original logit vector $\mathbf{x}_{i,t}$ and the neighborhood mean vector $\mathbf{m}_{i,t}$. The weights are given by the covariance matrix $\mathbf{S}_{i,t}$ of the prior distribution and the covariance matrix of the conditional distribution. The matrix $\mathbf{S}_{i,t}$ is calculated considering the spatiotemporal neighbors and the matrix $\boldsymbol{\Sigma}_{i,t}$ corresponds to the smoothing factor provided as prior belief by the user.

When the values of local class covariance $\mathbf{S}_{i,t}$ are relative to the conditional covariance $\boldsymbol{\Sigma}_{i,t}$, our confidence on the influence of the neighbors is low, and the smoothing algorithm gives more weight to the original pixel value $x_{i,k}$. When the local class covariance $\mathbf{S}_{i,t}$ decreases relative to the smoothness factor $\boldsymbol{\Sigma}_{i,t}$, then our confidence on the influence of the neighborhood increases. The smoothing procedure will be most relevant in situations where the original classification odds ratio is low, showing a low level of separability between classes. In these cases, the updated values of the classes will be influenced by the local class variances.

In practice, $\boldsymbol{\Sigma}_{i,t}$ is a user-controlled covariance matrix parameter that will be set by users based on their knowledge of the region to be classified. In the simplest case, users can associate the conditional covariance $\boldsymbol{\Sigma}_{i,t}$ to a diagonal matrix, using only one hyperparameter σ_k^2 to set the level of smoothness. Higher values of σ_k^2 will cause the assignment of the local mean to the pixel updated probability. In our case, after some classification tests, we decided to $\sigma_k^2 = 20$ by default for all k.

7.3 Use of Bayesian smoothing 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 probability images, one per class. The next step is to apply the `sits_smooth` function. By default, this function selects the most likely class for each pixel considering only the probabilities of each class for each pixel. To allow for Bayesian smoothing, it suffices to include the `type = bayesian` parameter (which is also the default). If desired, the `smoothness` parameter (associated to the hyperparameter σ_k^2 described above) can control the degree of smoothness. If so desired, the `smoothness` parameter can also be expressed as a matrix

The plots show the class probabilities, which can then be smoothed by a bayesian smoother.

The bayesian smoothing has removed some of local variability associated to misclassified pixels which are different from their neighbors. The impact of

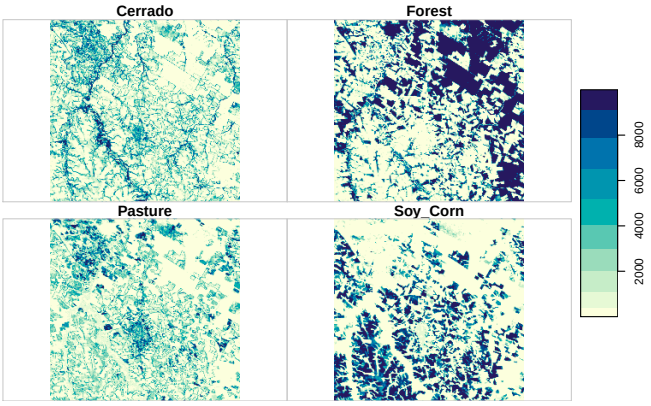


Figure 7.1: Probability values for classified image

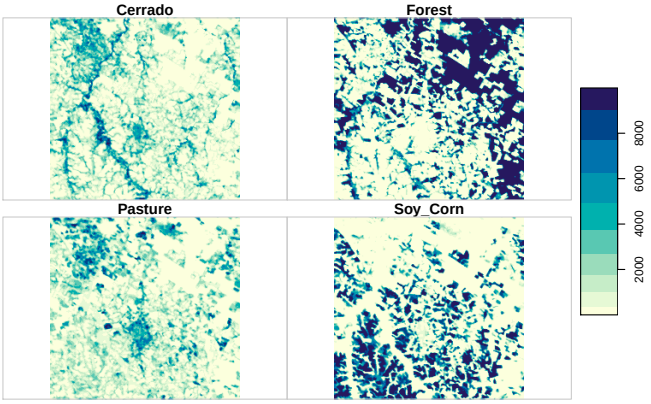


Figure 7.2: Probability values smoothed by bayesian method

smoothing is best appreciated comparing the labelled map produced without smoothing to the one that follows the procedure, as shown below.

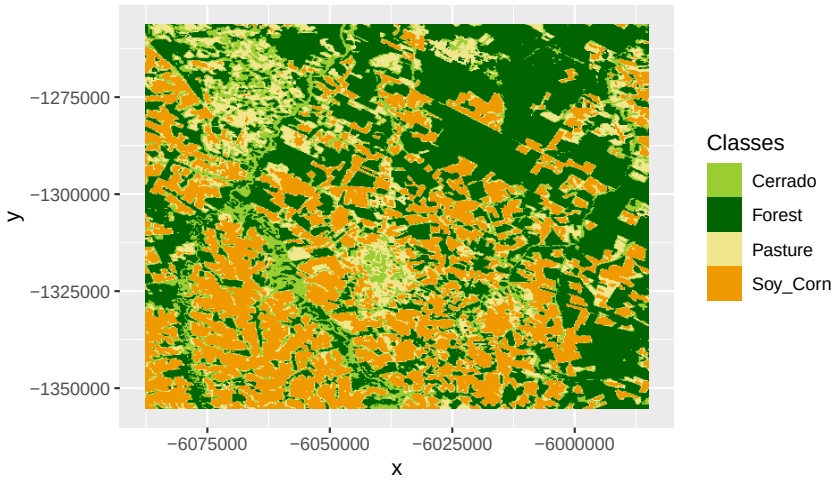


Figure 7.3: Classified image without smoothing

The resulting labelled map shows a number of likely misclassified pixels which can be removed using the bayesian smoother.

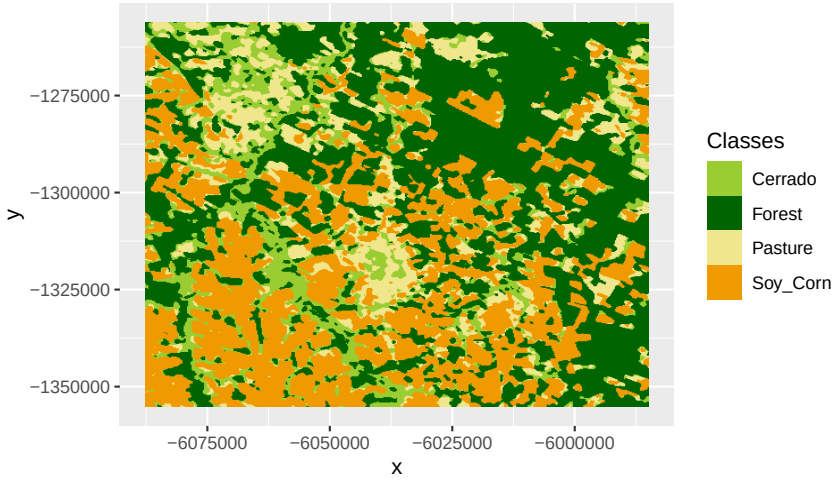


Figure 7.4: Classified image with Bayesian smoothing

Comparing the two plots, it is apparent that the smoothing procedure has reduced a lot of the noise in the original classification and produced a more homogeneous result.

7.4 Bilateral smoothing

One of the problems with post-classification smoothing is that we would like to remove noisy pixels (e.g., a pixel with high probability of being labeled "Forest" in the midst of pixels likely to be labeled "Cerrado"), but would like to preserve the edges between areas. Because of its design, bilateral filter has proven to be a useful method for post-classification processing since it preserves edges while removing noisy pixels [59].

Bilateral smoothing combines proximity (combining pixels which are close) and similarity (comparing the values of the pixels) [60]. If most of the pixels in a neighborhood have similar values, it is easy to identify outliers and noisy pixels. In contrast, there is a strong difference between the values of pixels in a if neighborhood, it is possible that the pixel is located in a class boundary. Bilateral filtering combines domain filtering with range filtering. In domain filtering, the weights used to combine pixels decrease with distance. In range filtering, the weights are computed considering value similarity.

The combination of domain and range filtering is mathematically expressed as:

$$S(x_i) = \frac{1}{W_i} \sum_{x_k \in \theta} I(x_k) \mathcal{N}_\tau(\|I(x_k) - I(x_i)\|) \mathcal{N}_\sigma(\|x_k - x_i\|),$$

where

- $S(x_i)$ is the smoothed value of pixel i;
- I is the original probability image to be filtered;
- $I(x_i)$ is the value of pixel i;
- θ is the neighborhood centered in x_i;
- x_k is a pixel k which belongs to neighborhood θ;
- $I(x_k)$ is the value of a pixel k in the neighborhood of pixel i;
- $\|I(x_k) - I(x_i)\|$ is the absolute difference between the values of the pixel k and pixel i;
- $\|x_k - x_i\|$ is the distance between pixel k and pixel i;
- $\mathcal{N}_\tau$ is the Gaussian range kernel for smoothing differences in intensities;
- $\mathcal{N}_\sigma$is the Gaussian spatial kernel for smoothing differences based on proximity.
- τ is the variance of the Gaussian range kernel;
- σ is the variance of the Gaussian spatial kernel.

The normalization term to be applied to compute the smoothed values of pixel i is defined as

$$W_i = \sum_{x_k \in \theta} \mathcal{N}_\tau(\|I(x_k) - I(x_i)\|) \mathcal{N}_\sigma(\|x_k - x_i\|)$$

For every pixel, the method takes a considers two factors: the distance between the pixel and its neighbors, and the difference in value between them. Each of the

values contributes according to a Gaussian kernel. These factors are calculated independently. Big difference between pixel values reduce the influence of the neighbor in the smoothed pixel. Big distance between pixels also reduce the impact of neighbors. The achieve a satisfactory result, we need to balance the σ and τ. As a general rule, the values of τ should range from 0.05 to 0.50, while the values of σ should vary between 4 and 16[61]. The default values adopted in *sits* are `tau = 0.1` and `sigma = 8`. As the best values of τ and σ depend on the variance of the noisy pixels, users are encouraged to experiment and find parameter values that best fit their requirements.

The following example shows the behavior of the bilateral smoother.

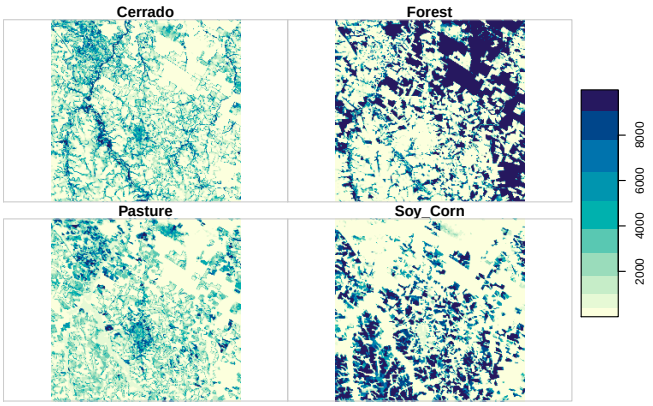


Figure 7.5: Probability values for classified image smoothed by bilateral filter

The impact on the classified image can be seen in the following example.

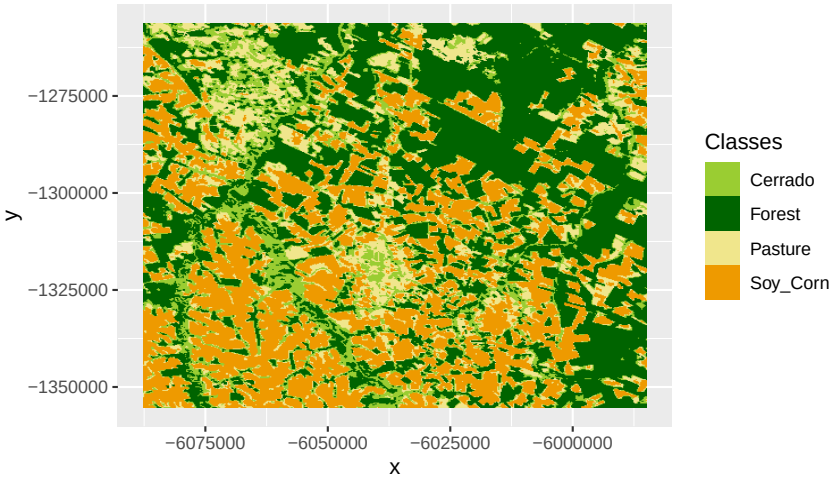


Figure 7.6: Classified image with bilateral smoothing

Bayesian smoothing tends to produce more homogeneous labeled images than bilateral smoothing. However, some spatial details and some edges are better preserved by the bilateral method. Choosing between the methods depends on user needs and requirements. In any case, as stated by 59, smoothing improves the quality of classified images and thus should be applied in most situations.

Chapter 8

Validation and accuracy measurements in SITS

This chapter presents the validation and accuracy measures available in the SITS package.

8.1 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 [45].

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 45, 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, 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).

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction Cerrado Pasture
#>    Cerrado     394      15
#>    Pasture       6     331
#>
#>           Accuracy : 0.9718
#>             95% CI : (0.9573, 0.9825)
#>
#>              Kappa : 0.9433
#>
#>  Prod Acc  Cerrado : 0.9850
#>  Prod Acc  Pasture : 0.9566
#>  User Acc  Cerrado : 0.9633
#>  User Acc  Pasture : 0.9822
#>
```

8.2 Comparing different machine learning methods using k-fold validation

One useful function in SITS is the capacity to compare different validation methods and store them in an XLS file for further analysis. The following example shows how to do this, using the Mato Grosso data set. We take five models: random forests(`sits_rfor`), support vector machines (`sits_svm`), extreme gradient boosting (`sits_xgboost`), multi-layer perceptron (`sits_mlp`) and temporal convolutional neural network (`sits_TempCNN`). For simplicity, we use the default parameters provided by sits. After computing the confusion matrix and the statistics for each model, we store the result in a list. When the calculation is finished, the function `sits_to_xlsx` writes all of the results in an Excel-compatible spreadsheet.

The resulting Excel file can be opened with R or using spreadsheet programs. The figure below shows a printout of what is read by Excel. As shown below,

each sheet corresponds to the output of one model. For simplicity, we show only the result of TempCNN, that has an overall accuracy of 97% and is the best-performing model.

	A	B	C	D	E	F	G	H	I	J
1		Pasture	Soy_Corn	Soy_Millet	Soy_Cottor	Fallow_Cot	Soy_Sunflow	Cerrado	Forest	Soy_Fallow
2	Pasture	340	3	1	2	1	0	2	0	0
3	Soy_Corn	0	340	6	8	0	3	0	0	0
4	Soy_Millet	0	13	171	0	0	0	0	0	1
5	Soy_Cotton	0	5	0	341	1	0	0	0	0
6	Fallow_Cotton	0	0	0	1	27	0	0	0	0
7	Soy_Sunflower	0	3	0	0	0	23	0	0	0
8	Cerrado	4	0	1	0	0	0	377	0	0
9	Forest	0	0	0	0	0	0	0	131	0
10	Soy_Fallow	0	0	1	0	0	0	0	0	86
11										
12		V1								
13	Accuracy	0.97								
14	Kappa	0.96								
15										
16										
17		Pasture	Soy_Corn	Soy_Millet	Soy_Cottor	Fallow_Cot	Soy_Sunflow	Cerrado	Forest	Soy_Fallow
18	Sensitivity (PA)	0.99	0.93	0.95	0.97	0.93	0.88	0.99	1.00	0.99
19	Specificity	0.99	0.99	0.99	1.00	1.00	1.00	1.00	1.00	1.00
20	PosPredValue (UA)	0.97	0.95	0.92	0.98	0.96	0.88	0.99	1.00	0.99
21	NegPredValue	1.00	0.98	0.99	0.99	1.00	1.00	1.00	1.00	1.00
22										

svm rfor xgboost MLP TempCNN +

Figure 8.1: Result of 5-fold cross validation of Mato Grosso dataset using TempCNN

8.3 Accuracy assessment

8.3.1 Time series

Users can perform accuracy assessment in *sits* both in time series datasets or in classified images using the `sits_accuracy` function. In the case of time series, the input is a sits tibble which has been classified by a sits model. The input tibble needs to contain valid labels in its "label" column. These labels are compared to the results of the prediction to the reference values. This function calculates the confusion matrix and then the resulting statistics using the R package "caret".

```
#>
#> Overall Statistics
#>
#>  Accuracy : 1
#>    95% CI : (0.9951, 1)
#>
#>     Kappa : 1
```

The detailed accuracy measures can be obtained by printing the accuracy object.

```
#> Confusion Matrix and Statistics
#>
```

```
#>           Reference
#> Prediction Cerrado Pasture
#>    Cerrado     400       0
#>    Pasture       0     346
#>
#>           Accuracy : 1
#>             95% CI : (0.9951, 1)
#>
#>              Kappa : 1
#>
#>  Prod Acc  Cerrado : 1
#>  Prod Acc  Pasture : 1
#>  User Acc  Cerrado : 1
#>  User Acc  Pasture : 1
#>
```

8.3.2 Classified images

To measure the accuracy of classified images, the `sits_accuracy` function uses an area-weighted technique, following the best practices proposed by 62. The need for area-weighted estimates arises from the fact the land use and land cover classes are not evenly distributed in space. In some applications (e.g., deforestation) where the interest lies in assessing how much of the image has changed, the area mapped as deforested is likely to be a small fraction of the total area. If users disregard the relative importance of small areas where change is taking place, the overall accuracy estimate will be inflated and unrealistic. For this reason, 62 argue that *"mapped areas should be adjusted to eliminate bias attributable to map classification error and these error-adjusted area estimates should be accompanied by confidence intervals to quantify the sampling variability of the estimated area"*.

With this motivation, when measuring accuracy of classified images, the function `sits_accuracy` follows 62 and 63. The following explanation is extracted from the paper of 62, and users should refer to this paper for further explanation.

Given a classified image and a validation file, the first step is to calculate the confusion matrix in the traditional way, i.e., by identifying the commission and omission errors. Then we calculate the unbiased estimator of the proportion of area in cell i, j of the error matrix

$$\hat{p_{i,j}} = W_i \frac{n_{i,j}}{n_i}$$

where the total area of the map is A_{tot}, the mapping area of class i is $A_{m,i}$ and the proportion of area mapped as class i is $W_i = A_{m,i}/A_{tot}$. Adjusting for area size allows producing an unbiased estimation of the total area of class j, defined

as a stratified estimator

$$\hat{A}_j = A_{tot} \sum_{i=1}^{K} W_i \frac{n_{i,j}}{n_i}$$

This unbiased area estimator includes the effect of false negatives (omission error) while not considering the effect of false positives (commission error). The area estimates also allow producing an unbiased estimate of the user's and producer's accuracy for each class. Following 62, we can also estimate the 95% confidence interval for $\hat{A}_j$.

To use the `sits_accuracy` function to produce the adjusted area estimates, users have to provide the classified image together with a csv file containing a set of labeled points. The csv file should have the same format as the one used to obtain samples, as discussed earlier.

In what follows, we show a simple example of using the accuracy function to estimate the quality of the classification

```
#> Area Weigthed Statistics
#> Overall Accuracy = 0.764\begin{table}
#>
#> \caption{\label{tab:unnamed-chunk-72}Area-Weighted Users and Producers Accuracy}
#> \centering
#> \begin{tabular}[t]{l|r|r}
#> \hline
#>   & User & Producer\\
#> \hline
#> Cerrado & 1.00 & 0.67\\
#> \hline
#> Forest & 0.60 & 1.00\\
#> \hline
#> Pasture & 0.75 & 0.61\\
#> \hline
#> Soy\_Corn & 0.86 & 0.72\\
#> \hline
#> \end{tabular}
#> \end{table}
#> \begin{table}
#>
#> \caption{\label{tab:unnamed-chunk-72}Mapped Area x Estimated Area (ha)}
#> \centering
#> \begin{tabular}[t]{l|r|r|r}
#> \hline
#>   & Mapped Area (ha) & Error-Adjusted Area (ha) & Conf Interval (ha)\\
#> \hline
#> Cerrado & 32821.2 & 49092.3 & 31891.3\\
#> \hline
```

```
#> Forest & 81355.4 & 48813.2 & 39058.7\\
#> \hline
#> Pasture & 18884.5 & 23195.1 & 19974.8\\
#> \hline
#> Soy\_Corn & 63222.2 & 75182.6 & 37630.4\\
#> \hline
#> \end{tabular}
#> \end{table}
```

This is an illustrative example to express the situation where there is a limited number of ground truth points. As a result of a limited validation sample, the estimated confidence interval in area estimation is large. This indicates a questionable result. We recommend that users follow the procedures recommended by 63 to estimate the number of ground truth measures per class that are required to get a reliable estimate.

Chapter 9

Case studies

This chapter presents case studies using sits

Chapter 10

Design and extensibility considerations

This chapter presents design decision for the **sits** package and shows how users can add their own machine learning algorithms to work with sits.

10.1 Design decisions

Compared with existing tools, sits has distinctive features:

1. A consistent API that encapsulates the entire land classification workflow in a few commands.
2. Integration with data cubes and Earth observation image collections available in cloud services such as AWS and Microsoft.
3. A single interface for different machine learning and deep learning algorithms.
4. Internal support for parallel processing, without requiring users to learn how to improve the performance of their scripts.
5. Support for efficient processing of large areas in a user-transparent way.
6. Innovative methods for sample quality control and post-processing.
7. Capacity to run on virtual machines in cloud environments.

Considering the aims and design of **sits**, it is relevant to discuss how its design and implementation choices differ from other software for big EO data analytics, such as Google Earth Engine [**?**], Open Data Cube [**?**] and openEO [**?**]. In what follows, we compare **sits** to each of these solutions.

Google Earth Engine (GEE) [**?**] uses the Google distributed file system [**?**] and its map-reduce paradigm [**?**]. By combining a flexible API with an efficient back-end processing, GEE has become a widely used platform [**?**]. However, GEE is restricted to the Google environment and does not provide direct support for deep learning. By contrast, **sits** aims to support different cloud environments and to allow advances in data analysis by providing a user-extensible interface to include new machine learning algorithms.

The Open Data Cube (ODC) is an important contribution to the EO community and has proven its usefulness in many domains [**?**, **?**]. It reads subsets of image collections and makes them available to users as a Python `xarray` structure. ODC does not provide an API to work with `xarrays`, relying on the tools available in Python. This choice allows much flexibility at the cost of increasing the learning curve. It also means that temporal continuity is restricted to the `xarray` memory data structure; cases where tiles from an image collection have different timelines are not handled by ODC. The design of **sits** takes a different approach, favouring a simple API with few commands to reduce the learning curve. Processing and handling large image collections in **sits** does not require knowledge of parallel programming tools. Thus, **sits** and ODC have different aims and will appeal to different classes of users.

Designers of the openEO API [**?**] aim to support applications that are both language-independent and server-independent. To achieve their goals, openEO designers use microservices based on REST protocols. The main abstraction of openEO is a *process*, defined as an operation that performs a specific task. Processes are described in JSON and can be chained in process graphs. The software relies on server-specific implementations that translate an openEO process graph into an executable script. Arguably, openEO is the most ambitious solution for reproducibility across different EO data cubes. To achieve its goals, openEO needs to overcome some challenges. Most data analysis functions are not self-contained. For example, machine learning algorithms depend on libraries such as TensorFlow and Torch. If these libraries are not available in the target environment, the user-requested process may not be executable. Thus, while the authors expect openEO to evolve into a widely-used API, it is not yet feasible to base an user-driven operational software such as **sits** in openEO.

Designing software for big Earth observation data analysis requires making compromises between flexibility, interoperability, efficiency, and ease of use. GEE is constrained by the Google environment and excels at certain tasks (e.g., pixel-based processing) while being limited at others such as deep learning. ODC allows users complete flexibility in the Python ecosystem, at the cost of limitations when working with large areas and requiring programming skills. The openEO API achieves platform independence but needs additional effort in designing drivers for specific languages and cloud services. While the **sits** API provides a simple and powerful environment for land classification, it has currently no support for other kinds of EO applications. Therefore, each of these solutions has benefits and drawbacks. Potential users need to understand the

design choices and constraints to decide which software best meets their needs.

[1] K. Didan, "MOD13Q1 MODIS/Terra Vegetation Indices 16-Day L3 Global 250m SIN Grid V006," NASA EOSDIS Land Processes DAAC, 2015.

[2] C. Pelletier, G. I. Webb, and F. Petitjean, "Temporal Convolutional Neural Network for the Classification of Satellite Image Time Series," *Remote Sensing*, vol. 11, no. 5, 2019.

[3] M. Appel and E. Pebesma, "On-Demand Processing of Data Cubes from Satellite Image Collections with the gdalcubes Library," *Data*, vol. 4, no. 3, pp. 1–16, 2019, doi: 10.3390/data4030092.

[4] K. R. Ferreira *et al.*, "Earth Observation Data Cubes for Brazil: Requirements, Methodology and Products," *Remote Sensing*, vol. 12, nos. 24, 24, p. 4033, Jan. 2020, doi: 10.3390/rs12244033.

[5] H. Wickham and G. Grolemund, *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. O'Reilly Media, Inc., 2017.

[6] E. F. Lambin and M. Linderman, "Time series of remote sensing data for land change science," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 44, no. 7, pp. 1926–1928, 2006.

[7] P. M. Atkinson, C. Jeganathan, J. Dash, and C. Atzberger, "Inter-comparison of four models for smoothing satellite sensor time-series data to estimate vegetation phenology," *Remote Sensing of Environment*, vol. 123, pp. 400–417, 2012.

[8] Y. Shao, R. S. Lunetta, B. Wheeler, J. S. Iiames, and J. B. Campbell, "An evaluation of time-series smoothing algorithms for land-cover classifications using MODIS-NDVI multi-temporal data," *Remote Sensing of Environment*, vol. 174, pp. 258–265, 2016.

[9] J. Zhou, L. Jia, M. Menenti, and B. Gorte, "On the performance of remote sensing time series reconstruction methods: A spatial comparison," *Remote Sensing of Environment*, vol. 187, pp. 367–384, 2016.

[10] B. A. Bradley, R. W. Jacob, J. F. Hermance, and J. F. Mustard, "A curve fitting procedure to derive inter-annual phenologies from time series of noisy satellite NDVI data," *Remote Sensing of Environment*, vol. 106, no. 2, pp. 137–145, 2007.

[11] T. Sakamoto, M. Yokozawa, H. Toritani, M. Shibayama, N. Ishitsuka, and H. Ohno, "A crop phenology detection method using time-series MODIS data," *Remote Sensing of Environment*, vol. 96, nos. 3-4, pp. 366–374, 2005.

[12] J. N. Hird and G. J. McDermid, "Noise reduction of NDVI time series: An empirical comparison of selected techniques," *Remote Sensing of Environment*, vol. 113, no. 1, pp. 248–258, 2009.

[13] INPE, "Amazon Deforestation Monitoring Project (PRODES)," National Institute for Space Research, Brazil, 2019. [Online]. Available: www.obt.inpe.br/prodes.

[14] H. H. Madden, "Comments on the Savitzky-Golay convolution method for least-squares-fit smoothing and differentiation of digital data," *Analytical chemistry*, vol. 50, no. 9, pp. 1383–1386, 1978.

[15] J. Chen, P. Jönsson, M. Tamura, Z. Gu, B. Matsushita, and L. Eklundh, "A simple method for reconstructing a high-quality NDVI time-series data set based on the Savitzky–Golay filter," *Remote Sensing of Environment*, vol. 91, no. 3, pp. 332–344, Jun. 2004, doi: 10.1016/j.rse.2004.03.014.

[16] C. Atzberger and P. H. Eilers, "Evaluating the effectiveness of smoothing algorithms in the absence of ground reference measurements," *International Journal of Remote Sensing*, vol. 32, no. 13, pp. 3689–3709, 2011.

[17] E. T. Whittaker, "On a new method of graduation," *Proceedings of the Edinburgh Mathematical Society*, vol. 41, pp. 63–75, 1922.

[18] M. Picoli *et al.*, "Big earth observation time series analysis for monitoring Brazilian agriculture," *ISPRS journal of photogrammetry and remote sensing*, vol. 145, pp. 328–339, 2018, doi: 10.1016/j.isprsjprs.2018.08.007.

[19] R. Simoes *et al.*, "Land use and cover maps for Mato Grosso State in Brazil from 2001 to 2017," *Scientific Data*, vol. 7, no. 1, p. 34, Dec. 2020, doi: 10.1038/s41597-020-0371-4.

[20] L. Parente, E. Taquary, A. P. Silva, C. Souza, and L. Ferreira, "Next Generation Mapping: Combining Deep Learning, Cloud Computing, and Big Remote Sensing Data," *Remote Sensing*, vol. 11, nos. 23, 23, p. 2881, Jan. 2019, doi: 10.3390/rs11232881.

[21] P. Griffiths, C. Nendel, J. Pickert, and P. Hostert, "Towards national-scale characterization of grassland use intensity from integrated Sentinel-2 and Landsat time series," *Remote Sensing of Environment*, p. 111124, Apr. 2019, doi: 10.1016/j.rse.2019.03.017.

[22] A. E. Maxwell, T. A. Warner, and F. Fang, "Implementation of machine-learning classification in remote sensing: An applied review," *International Journal of Remote Sensing*, vol. 39, no. 9, pp. 2784–2817, 2018.

[23] P. Thanh Noi and M. Kappas, "Comparison of Random Forest, k-Nearest Neighbor, and Support Vector Machine Classifiers for Land Cover Classification Using Sentinel-2 Imagery," *Sensors*, vol. 18, nos. 1, 1, p. 18, Jan. 2018, doi: 10.3390/s18010018.

[24] B. Frenay and M. Verleysen, "Classification in the Presence of Label Noise: A Survey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 5, pp. 845–869, May 2014, doi: 10.1109/TNNLS.2013.2292894.

[25] E. Keogh, J. Lin, and W. 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*, 2003, pp. 115–122.

[26] S. Aghabozorgi, A. S. Shirkhorshidi, and T. Y. Wah, "Time-series clustering: A decade review," *Information Systems*, vol. 53, pp. 16–38, 2015, doi: 10.1016/j.is.2015.04.007.

[27] J. H. Ward, "Hierarchical grouping to optimize an objective function," *Journal of the American statistical association*, vol. 58, no. 301, pp. 236–244, 1963.

[28] C. Hennig, "Clustering strategy and method selection," in *Handbook of cluster analysis*, C. Hennig, M. Meila, F. Murtagh, and R. Rocci, Eds. CRC Press, 2015.

[29] L. Hubert and P. Arabie, "Comparing partitions," *Journal of classification*, vol. 2, no. 1, pp. 193–218, 1985.

[30] T. Kohonen, "The self-organizing map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, Sep. 1990, doi: 10.1109/5.58325.

[31] T. Kohonen, "Essentials of the self-organizing map," *Neural Networks*, vol. 37, pp. 52–65, Jan. 2013, doi: 10.1016/j.neunet.2012.09.018.

[32] R. Wehrens and J. Kruisselbrink, "Flexible Self-Organizing Maps in kohonen 3.0," *Journal of Statistical Software*, vol. 87, nos. 1, 1, pp. 1–18, Nov. 2018, doi: 10.18637/jss.v087.i07.

[33] G. Mountrakis, J. Im, and C. Ogole, "Support vector machines in remote sensing: A review," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 66, no. 3, pp. 247–259, 2011.

[34] M. Belgiu and L. Dragut, "Random Forest in remote sensing: A review of applications and future directions," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 114, pp. 24–31, 2016.

[35] J. C. Brown, J. H. Kastens, A. C. Coutinho, D. de C. Victoria, and C. R. Bishop, "Classifying multiyear agricultural land use data from Mato Grosso using time-series MODIS vegetation index data," *Remote Sensing of Environment*, vol. 130, pp. 39–50, 2013.

[36] 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*, vol. 12, no. 4, p. e0176168, 2017.

[37] P. Jonsson and L. Eklundh, "TIMESAT: A program for analyzing time-series of satellite sensor data," *Computers and Geosciences*, vol. 30, no. 8, pp. 833–845, 2004.

[38] S. Estel, T. Kuemmerle, C. Alcantara, C. Levers, A. Prishchepov, and P. Hostert, "Mapping farmland abandonment and recultivation across Europe

using MODIS NDVI time series," *Remote Sensing of Environment*, vol. 163, pp. 312–325, 2015.

[39] C. Pelletier, S. Valero, J. Inglada, N. Champion, and G. 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*, vol. 187, pp. 156–168, 2016, doi: 10.1016/j.rse.2016.10.010.

[40] C. Gomez, J. C. White, and M. A. Wulder, "Optical remotely sensed time series data for land cover classification: A review," *{ISPRS} Journal of Photogrammetry and Remote Sensing*, vol. 116, pp. 55–72, 2016, doi: 10.1016/j.isprsjprs.2016.03.008.

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

[42] V. Maus, G. Câmara, M. Appel, and E. Pebesma, "dtwSat: Time-Weighted Dynamic Time Warping for Satellite Image Time Series Analysis in R," *Journal of Statistical Software*, vol. 88, no. 5, pp. 1–31, 2019, doi: 10.18637/jss.v088.i05.

[43] V. Maus, G. Camara, R. Cartaxo, A. Sanchez, F. M. Ramos, and G. R. 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*, vol. 9, no. 8, pp. 3729–3739, 2016, doi: 10.1109/JSTARS.2016.2517118.

[44] J. S. Wright *et al.*, "Rainforest-initiated wet season onset over the southern Amazon," *Proceedings of the National Academy of Sciences*, Jul. 2017, doi: 10.1073/pnas.1621516114.

[45] T. Hastie, R. Tibshirani, and F. J, *The Elements of Statistical Learning. Data Mining, Inference, and Prediction.* New York: Springer, 2009.

[46] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[47] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM transactions on intelligent systems and technology (TIST)*, vol. 2, no. 3, p. 27, 2011.

[48] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug. 2016, pp. 785–794, doi: 10.1145/2939672.2939785.

[49] F. Chollet and J. J. Allaire, *Deep Learning with R*. Manning Publications Co., 2018.

[50] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[51] S. Ruder, "An overview of gradient descent optimization algorithms," *CoRR*, vol. abs/1609.04747, 2016, [Online]. Available: http://arxiv.org/abs/1609.04747.

[52] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[53] Z. Wang, W. Yan, and T. Oates, "Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline," 2017.

[54] M. Rußwurm and M. Korner, "Temporal vegetation modelling using long short-term memory networks for crop identification from medium-resolution multi-spectral satellite images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 11–19.

[55] M. Russwurm and M. Korner, "Multi-temporal land cover classification with sequential recurrent encoders," *ISPRS International Journal of Geo-Information*, vol. 7, no. 4, p. 129, 2018.

[56] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 2, pp. 107–116, Apr. 1998, doi: 10.1142/S0218488598000094.

[57] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: A review," *Data Mining and Knowledge Discovery*, vol. 33, no. 4, pp. 917–963, 2019.

[58] X. Huang, Q. Lu, L. Zhang, and A. Plaza, "New postprocessing methods for remote sensing image classification: A systematic study," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 52, no. 11, pp. 7140–7159, 2014.

[59] K. Schindler, "An overview and comparison of smooth labeling methods for land-cover classification," *IEEE transactions on geoscience and remote sensing*, vol. 50, no. 11, pp. 4534–4545, 2012.

[60] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, 1998, pp. 839–846, doi: 10.1109/ICCV.1998.710815.

[61] S. Paris, P. Kornprobst, J. Tumblin, and F. Durand, "A gentle introduction to bilateral filtering and its applications," in *ACM SIGGRAPH 2007 courses*, Aug. 2007, pp. 1–es, doi: 10.1145/1281500.1281602.

[62] P. Olofsson, G. M. Foody, S. V. Stehman, and C. E. Woodcock, "Making better use of accuracy data in land change studies: Estimating accuracy and area and quantifying uncertainty using stratified estimation," *Remote Sensing of Environment*, vol. 129, pp. 122–131, Feb. 2013, doi: 10.1016/j.rse.2012.10.031.

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