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 Alexandre Carvalho
Karine Ferreira Gilberto Queiroz Victor Maus

2021-04-08

Contents

Preface **7**
Who this book is for . . . 7
How to use this book . . . 7
Publications using sits . . . 8

Setup **11**
Docker images . . . 11

Acknowledgements **13**

I Overview **15**

1 Introduction **17**
1.1 Workflow and API . . . 19
1.2 Handling Data Cubes in sits . . . 21
1.3 Handling satellite image time series in **sits** . . . 22
1.4 Sample quality control using clustering . . . 24
1.5 Classification using machine learning . . . 24
1.6 Cube classification . . . 25
1.7 Smoothing and Labelling of raster data after classification . . . 26
1.8 Validation techniques . . . 27
1.9 Final remarks . . . 28

2 Earth observation data cubes **29**
2.1 Image data cubes as the basis for big Earth observation data analysis . . . 29
2.2 Using STAC to Access Image Collection . . . 32
2.3 Accessing data cubes in Amazon Web Services . . . 32
2.4 Accessing the Brazil Data Cube . . . 33
2.5 Defining a data cube using files . . . 35
2.6 Regularizing data cubes . . . 35

3 Working with time series **37**

3.1 Data structures for satellite time series 37
3.2 Utilities for handling time series 38
3.3 Time series visualisation 39
3.4 Obtaining time series data from data cubes 41
3.5 Filtering techniques for time series 44

II Clustering 49

4 Time Series Clustering to Improve the Quality of Training Samples 51
4.1 Clustering for sample quality control 51
4.2 Hierachical clustering for Sample Quality Control 52
4.3 Using Self-organizing Maps for Sample Quality 55
4.4 Conclusion 61

III Classification 63

5 Machine Learning for Data Cubes using the SITS package 65
5.1 Machine learning classification 65
5.2 Visualizing Samples 67
5.3 Common interface to machine learning and deeplearning models 69
5.4 Random forests 69
5.5 Support Vector Machines 71
5.6 Extreme Gradient Boosting 73
5.7 Deep learning using multi-layer perceptrons 74
5.8 Combined 1D CNN and multi-layer perceptron networks 77
5.9 LSTM Convolutional Networks for Time Series Classification . . 80

6 Classification of Images in Data Cubes using Satellite Image Time Series 83
6.1 Image data cubes as the basis for big Earth observation data analysis 83
6.2 Defining a data cube using files organised as raster bricks 84
6.3 Classification using machine learning 85
6.4 Cube classification 86
6.5 Final remarks 88

IV Post classification 91

7 Post classification smoothing using Bayesian techniques in SITS 93
7.1 Introduction 93
7.2 Overview of Bayesian estimattion 94
7.3 Use of Bayesian smoothing in SITS 97

8 Validation and accuracy measurements in SITS **101**
8.1 Validation techniques 101
8.2 Comparing different validation methods 102

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 satellite image time series analysis. The package supports the application of machine learning techniques for classification image time series obtained from data cubes. Methods available include linear and quadratic discrimination analysis, support vector machines, random forests, boosting, deep learning, and convolution neural. The package also provides functions to post-classification and sample quality assessment.

Who this book is for

This book is recommended for researchers and experts that want to use the full potential of Earth observation data cubes for time series analysis and classification. Ideally, users should have basic knowledge of data science methods using R. The examples of the book are largely self-explanatory. Background information for the data science methods used in the book is available in 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.

How to use this book

This book describes sits version 0.11.0. Download and install sits 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 with.

Chapter	Description
Chr 1	Provides an overview of **sits** package.
Chr 2	Describes how to work with Earth observation data cubes in sits.
Chr 3	Describes how to access information from time series in sits.
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 available in 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.

Setup

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

```
devtools::install_github("e-sensing/sits", dependencies = TRUE)
```

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.

```
docker run --detach \
           --publish 127.0.0.1:8787:8787 \
           --name my-sits-rstudio \
           --volume ${PWD}/data:/data \
           brazildatacube/sits-rstudio:1.4.1103
```

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 Fauze Karim and Charlotte Pelletier for sharing the python code that has been reused for the "LSTM-FCN" and "TempCNN" machine learning models.

Part I

Overview

Chapter 1

Introduction

This chapter present an overview of sits. 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-classification.

Satellite image time series are obtained by taking calibrated and comparable measures of the same location in Earth at different times. These measures can come from a single sensor (e.g., MODIS) or by combining various sensors (e.g., Landsat 8 and Sentinel-2). If acquired by frequent revisits, these data set can capture significant land use changes. As argued by (Woodcock et al. 2020), *"dense time series analysis is providing new information on the timing of landscape changes, as well as improving the quality and accuracy of information being derived from remote sensing. The result is a paradigm shift away from change detection, typically using two points in time, to monitoring, or an attempt to track change continuously in time."*

Time series of remote sensing data show that land cover can occur not only

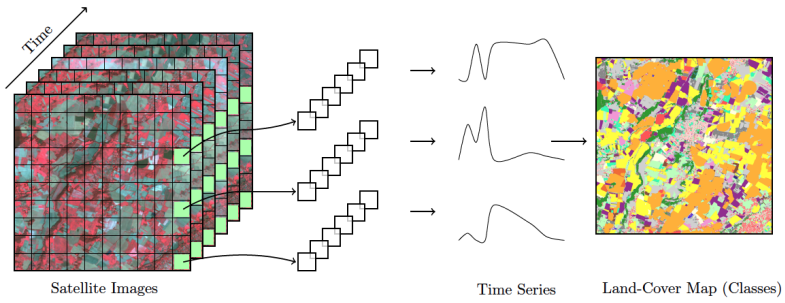


Figure 1.1: Using time series for land classification (source: Tan et al., 2017)

progressively and gradually, but they may also show discontinuities with abrupt changes (Lambin, Geist, and Lepers 2003). Analyses of multiyear time series of land surface attributes, their fine-scale spatial pattern, and their seasonal evolution lead to a broader view of land-cover change. Satellite image time series have already been used in applications such as mapping for detecting forest disturbance (Kennedy, Yang, and Cohen 2010), land disturbance (Zhu et al. 2020), ecological dynamics (Pasquarella et al. 2016), agricultural intensification (Galford et al. 2008), and deforestation monitoring (Arvor et al. 2012). Algorithms for processing image time series include BFAST for detecting breaks (Verbesselt et al. 2010), TIMESAT for extraction phenological attributes (Jonsson and Eklundh 2004), CCDC for continuous change detection (Arévalo et al. 2020), and methods based on Dynamic Time Warping (DTW) for land use and land cover classification (Petitjean, Inglada, and Gancarski 2012, @Maus2019).

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.

Designing software to do big Earth observation data analysis requires balancing between flexibility, interoperability, efficiency, and ease of use. The approach taken in *sits* is different from software such as Google Earth Engine (GEE) (Gorelick et al. 2017) and Open Data Cube (ODC) (Lewis et al. 2017). Similar to GEE, *sits* provides a compact API. However, GEE is restricted to the Google environment and does not provide direct support for deep learning using image

time series. Since *sits* runs in different cloud providers, users are flexible to choose the working environment that better fits their needs and resources.

As for the ODC, the approach taken by its designers is that of maximum flexibility. ODC provides users with data organized into a python *xarray* structure. Since ODC does not provide an API to work with xarrays, specialists have to develop their applications by themselves. This choice by ODC designers aims at allowing experts to develop many different types of applications. Such flexibility comes a cost for studies involving large areas that require parallel processing. In these cases, experts have to learn how to parallelize their scripts. By contrast, the *sits* API focus on providing a simple and powerful environment for land classification. Parellel processing is done internally and is user-transparent. We consider that the combination of user-centered design and interoperability is *sits* is well suited for experts that want to use satellite data to land-related data analysis.

1.1 Workflow and API

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

1. Create a data cube based on analysis-ready data image collections available in the cloud or local machines using `sits_cube`.
2. Provide a description of the training data as a CSV or SHP file.
3. Obtain the time series for the training samples using `sits_get_data`.
4. Validate the samples using `sits_kfold_validate`.
5. Improve the quality of the samples using `sits_som_map` and `sits_som_clean_samples`.
6. Train a machine learning model with `sits_train`.
7. Classify the data cubes with `sits_classify`, producing a cube with class probabilities.
8. Smoothen the probability data to reduce classification noise with `sits_smooth`.
9. Label the probability cube with `sits_label_classification`.
10. Assess the quality of the classification with `sits_accuracy`.

Figure 1.2 shows a generic vision of the workflow for land classification.

Figure 1.3 shows how the *sits* API is used to perform the main steps of the 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_label_classication` that produces the final labelled image. These five functions encapsulate the core of the *sits* package.

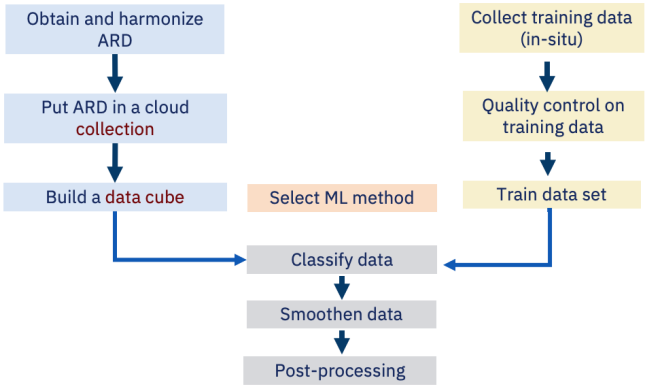


Figure 1.2: Workflow of using satellite image time series for classification

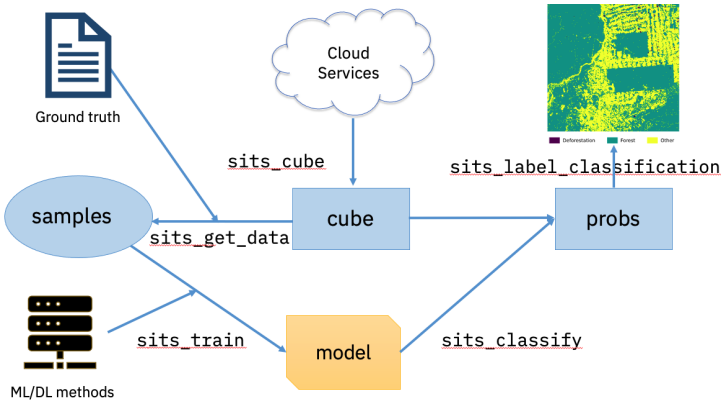


Figure 1.3: Main functions of the SITS API

1.2 Handling Data Cubes in sits

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

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

The user defines a data cube by selecting a cloud service collection and determining a space-time extent. The code below shows the definition of a data cube using AWS Sentinel-2/2A images to exemplify how it is used. In the example, the user selects the "Sentinel-2 Level 2" collection in the AWS cloud services. The data cube's geographical area is defined by the tile "20LKP" and the temporal extent by a start and end date. Access to other cloud services works in similar ways (See Chapter 2 for more details).

```
s2_cube <- sits_cube(
    source        = "AWS",
    name          = "T20LKP_2018_2019",
    collection    = "sentinel-s2-l2a",
    bands         = c("B08", "SCL"),
    tiles         = "20LKP",
    start_date    = as.Date("2018-07-18"),
    end_date      = as.Date("2018-08-18"),
    s2_resolution = 60
)
```

To define a data cube using plain files, all image files should have the same spatial resolution and same projection and be in the same directory. Each file should contain a single image band and cover single date. Since timeline and band information is deduced from filenames, users should provide parsing information to allow *sits* to extract the band and the date. As an example, for the file `CBERS-4_AWFI_B13_2018-02-02.tif`, the parsing info is `c("X1", "X2", "band", "date")`.

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

# files are named using the convention
# "CBERS-4_AWFI_B13_2018-02-02.tif"
cbers_cube <- sits_cube(
      source = "LOCAL",
      name = "022024",
      satellite = "CBERS-4",
```

```
      sensor = "AWFI",
      data_dir = data_dir,
      parse_info = c("X1", "X2", "band", "date")
)

# print the timeline of the cube
sits_timeline(cbers_cube)
```

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

1.3 Handling satellite image time series in sits

1.3.1 Data structure

Training a machine learning model in **sits** requires a set of time series describing properties in spatio-temporal locations of interest. This set consists of samples provided by experts that take in-situ field observations or recognize land classes using high-resolution images for land use classification.

The package uses a `sits tibble` to organize time series data with associated spatial information for handling time series. As an example of how the sits tibble works, the following code shows the first three lines of a tibble containing 1,218 labeled samples of land cover in the Mato Grosso state of Brazil, with four classes: "Forest", "Cerrado", "Pasture", "Soybean-Corn". Training samples organized as a tibble can be used to training a machine learning model with `sits_train`.

```
# data set of samples
data("samples_modis_4bands")
samples_modis_4bands[1:3,]
```

```
#> # A tibble: 3 x 7
#>   longitude latitude start_date end_date   label   cube    time_series
#>       <dbl>    <dbl> <date>     <date>     <chr>   <chr>   <list>
#> 1     -55.2   -10.8  2013-09-14 2014-08-29 Pasture MOD13Q1 <tibble [23 x 5]>
#> 2     -57.8    -9.76 2006-09-14 2007-08-29 Pasture MOD13Q1 <tibble [23 x 5]>
#> 3     -51.9   -13.4  2014-09-14 2015-08-29 Pasture MOD13Q1 <tibble [23 x 5]>
```

A `sits tibble` contains data and metadata. The first six columns contain the metadata: spatial and temporal information, 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 "Pasture" at location (−55.1852, −10.8378) and is valid for the period (2013-09-14, 2014-08-29). The `time_series` column contains the actual data.

1.3.2 Obtaining time series data

To get a time series in *sits*, user should create a data cube and then extract one or more time series from the cube using `sits_get_data()`. Users should provide a CSV or SHP file that includes latitude and longitude of the desired location, bands, and start date and end date of the time series.

```
library(sits)
data_dir <- system.file("extdata/raster/mod13q1", package = "sits")
modis_cube <- sits_cube(
    source = "LOCAL",
    name = "sinop-2014",
    satellite = "TERRA",
    sensor = "MODIS",
    data_dir = data_dir,
    delim = "_",
    parse_info = c("X1", "X2", "band", "date")
)
# obtain a set of locations defined by a CSV file
csv_raster_file <- system.file("extdata/samples/samples_sinop_crop.csv",
                               package = "sits")
# retrieve the points from the data cube
points <- sits_get_data(modis_cube, file = csv_raster_file)
```

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

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

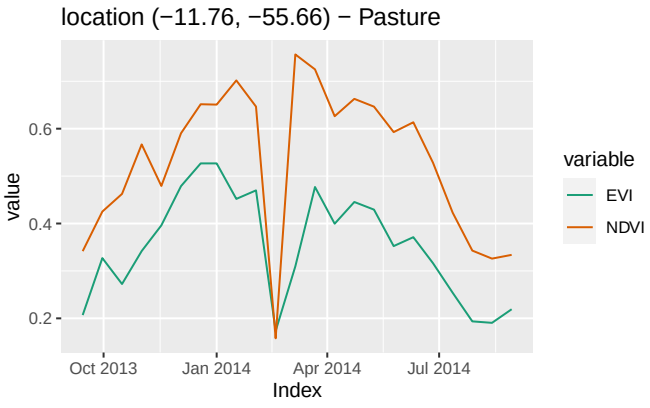


Figure 1.4: A one year time series of MOD13Q1 data for bands NDVI and EVI

1.4 Sample quality control using clustering

One of the key challenges of machine learning is improving the quality of the training samples. Good samples lead to good classification maps. It is recommended that users apply pre-processing methods to identify samples that might have been wrongly labeled or have low discriminatory power. *sits* provides support for two clustering methods to evaluate sample quality: (a) Agglomerative Hierarchical Clustering (AHC); (b) Self-organizing Maps (SOM). Full details of the clustering methods are available in Chapter 4.

1.5 Classification using machine learning

The package provides functionality to explore the full depth of satellite image time series data, treating time series as a feature vector. It uses as many temporal attributes as possible, increasing the classification space's dimension. The `sits_train` function provides a common interface to all machine learning (ML) models, with two parameters: the input samples and the ML method (`ml_method`). After the model is estimated, users can classify individual time series or full data cubes with `sits_classify`. In the example below, we train a ML model to classify a MODIS time series (described above). The classification results can be shown in text format using `sits_show_prediction` or graphically `plot`.

```
#select the data for classification
# Train a machine learning model using Random Forest
rfor_model <- sits_train(data = samples_modis_4bands,
                    ml_method = sits_rfor(num_trees = 1000))
# get a point to be classified (select 4 bands to match the model)
point_4bands <- sits_select(point_mt_6bands,
                            bands = c("NDVI", "EVI", "NIR", "MIR"))
# Classify using random forest model
class <- sits_classify(point_4bands, rfor_model)
# plot the results of the prediction
plot(class)
```

The following machine learning methods are available:

- Linear discriminant analysis (`sits_lda`)
- Quadratic discriminant analysis (`sits_qda`)
- Multinomial logit and its variants 'lasso' and 'ridge' (`sits_mlr`)
- Support vector machines (`sits_svm`)
- Random forests (`sits_rfor`)
- Extreme gradient boosting (`sits_xgboost`)
- Deep learning (DL) using multi-layer perceptrons (`sits_deeplearning`)
- DL with 1D convolutional neural networks (`sits_CNN`),
- DL combining 1D CNN and multi-layer perceptron networks

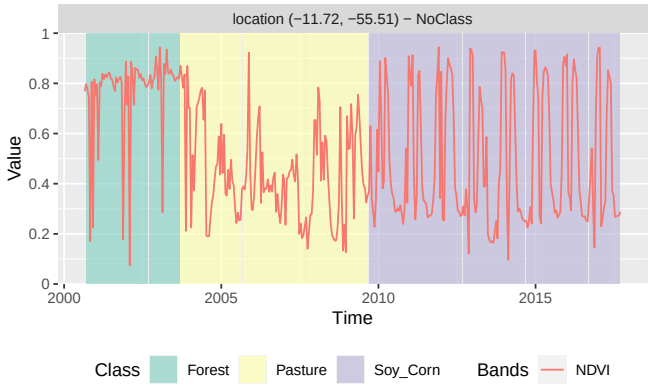


Figure 1.5: Random forest classification of a 16 years time series. The location (latitude, longitude) shown at the top of the graph is in geographic coordinate system (WGS84 *datum*).

(`sits_tempCNN`)

- DL using 1D version of ResNet (`sits_ResNet`).
- DL using a combination of long-short term memory (LSTM) and 1D CNN (`sits_LSTM_FCN`)

For more details, please see Chapter 5.

1.6 Cube classification

Given a data cube and a trained ML model, the function `sits_classify()` assigns class probability to each time series. To optimize processing time, users can adjust the function according to the server's capabilities, configuring the available memory (`memsize`) and the number of cores to be used (`multicores`).

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

# Retrieve the set of samples for the Mato Grosso region
# these samples have already been quality controlled
```

```
samples_2bands  <- sits_select(samples_modis_4bands,
                               bands = c("NDVI", "EVI"))
# build a machine learning model for this area
svm_model <- sits_train(data = samples_2bands, ml_method = sits_svm())

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

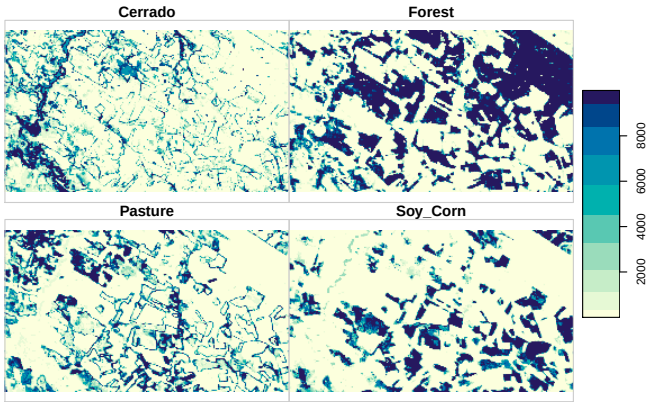


Figure 1.6: Class probabilities for each pixel

When applied to a data cube, `sits_classify` produces one probability image for each land class. These images are grouped in a probability cube. The value of each pixel of each class is proportional to the class probability estimated by the machine learning classifier, as shown in the Figure above.

1.7 Smoothing and Labelling of raster data after classification

Post-processing is a desirable step in any classification process. Most statistical classifiers use training samples derived from single pixels. However, images contain many mixed pixels irrespective of the resolution. Also, there is a considerable degree of data variability in each class. Smoothing methods available in `sits_smooth` use the neighborhood information to remove outliers and enhance consistency in the resulting product. By default, this function selects the most

likely class for each pixel using a Bayesian estimator that considers the neighbors. Alternatives are gaussian and bilinear smoothing. The resulting cube can be labeled using `sits_label_classification`. This function select the most likely class in each pixel and assigns it to the final classified image.

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

Figure 1.7: Classified image post-processed with Bayesian smoothing

1.8 Validation techniques

Validation is a process undertaken on models to estimate their prediction errors. **sits** supports the *k-fold cross-validation* approach, probably the most used validation technique (Hastie, Tibshirani, and J 2009) using the function `sits_kfold_validate`. The following code performs a five-fold validation using the SVM classification model as a default classifier. By default, `sits_kfold_validate` provides the overall accuracy metrics. The detailed metrics

```
# perform a five fold validation for the "cerrado_2classes" data set
# Random Forest machine learning method using default parameters
acc <- sits_kfold_validate(cerrado_2classes,
                           folds = 5,
                           ml_method = sits_rfor(num_trees = 1000))
```

```
# print detailed validation metrics (not shown here to reduce space)
sits_accuracy_summary(acc)
```

More details about validation are available in Chapter 8

1.9 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

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

Generalizing (Appel and Pebesma 2019), 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.

Data cubes are particularly amenable for machine learning; their data can be transformed into arrays in memory, fed to training and classification algorithms. Conceptually, a data cube defines a compact space. For all positions inside its

spatial temporal extent, it is possible to obtain a valid set of values from every band of the images which are included in it (see Figure 2.1).

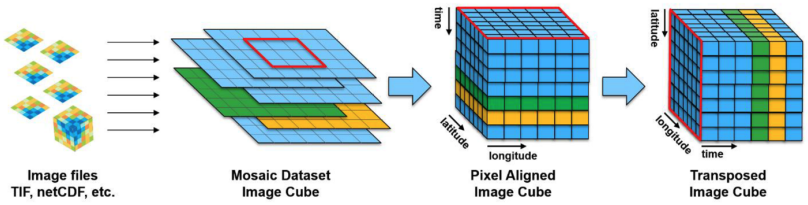


Figure 2.1: Conceptual view of data cubes (source: Kopp et al., 2019)

This idealized vision of data cubes does not correspond to the reality. Data available in cloud services such as AWS, Microsoft and Digital Earth Africa is better described as collections of analysis-ready data (ARD). Such data is defined by the Committee on Earth Observation Satellites (CEOS) as "satellite data that have been processed to a minimum set of requirements and organized into a form that allows immediate analysis with a minimum of additional user effort and interoperability both through time and with other datasets" (Potapov et al. 2020). In practice, ARD label is applied to images that have been processed by space agencies to improve multidate comparability. This processing includes conversion from radiance measures at the top of the athmosphere 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. These data sets are better characterized as "ARD image collections", defined as:

1. An ARD image collection is set of files from a given sensor (or combined set of sensors) that has been corrected to ensure comparability of measurements between different dates.
2. All images are reprojected to a cartographical projection following well-established standards.
3. Image collections are usually cropped into a tiling system, where each tile is associated to a different timeline than the other tiles.

The use of tiling systems for ARD data has been adopted by space agencies for many land satellites. For example, the Sentinel-2 tiling system uses the Military Grid Reference System (MGRS) and its naming convention derived from the UTM (Universal Transverse Mercator). Use of tiling system enables users to more easily choose an area of interest. Figure 2.2 with the location of two Sentinel-2 tiles (20LLP and 20LKP) covering part of the state of Rondonia, Brazil.

There are important distinctions between ARD image collections and data cubes. Image collections do not guarantee that every pixel of an image has a valid value,

Figure 2.2: Illustration of Sentinel-2 tiling system (source: www.remotepixel.ca).

since ARD still contains cloudy pixels. Also, each tile of an image collection has a unique timeline. Using the Sentinel-2 level 2A image collection as an example, consider neighboring tiles "20LLP" and "20LKP". Figure 2.3 shows images of tile "20LKP" for different dates. Some of images have a significant number of clouds. Ideally, these clouds should be replaced by a valid value before being ingested into a data cube.

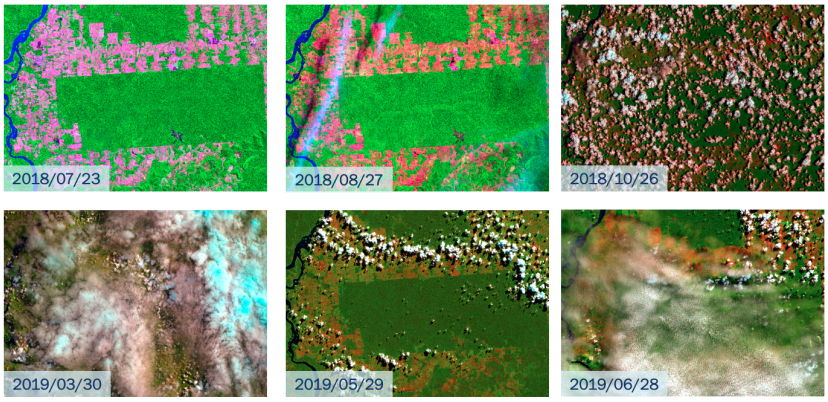


Figure 2.3: Sentinel-2 for tile 20LKP in different dates (source: authors).

A further point concerns the timeline of different tiles. For example, consider the Sentinel-2 tiles "20LLP" and "20LKP" for the period 2018-07-13 to 2019-07-28. Tile 20LKP has 71 temporal instances and tile 20 LLP has 144 instances. Users that want to process large areas cannot thus rely only on ARD image collections being compatible with data cube properties. There are three options available for users aiming to classify areas that span more than one tile:

1. Use cloud providers whose data fully adheres to a data cube definition.

The Brazil Data Cube (BDC) is an example.
2. Use tools that generate regularly spaced cubes in time from image collection, as does the `sits_regularize` funcion in *sits*.
3. Provide additional information to *sits* to allow the software to deal with multi-tile image collections.

Since option (1) is not generally available to users other than those of the BDC, and option (2) requires the creation of additional files to compose a proper data, the developers of *sits* are working to provide full support to multi-tile data cubes, in a user-transparent way. This facility is not yet available in *sits* 0.11.0, but it is planned for version 1.00.00.

2.2 Using STAC to Access Image Collection

With the coming of age of big Earth observation data, it is not always practical to transfer large data sets. Users have to rely on web services to provide access to these data sets. To obtain information on cloud image collection, *sits* uses information provided by STAC (SpatioTemporal Asset Catalogue), by means of the rstac package. [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.

2.3 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 only 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.

```
Sys.setenv(
    "AWS_ACCESS_KEY_ID"     = <your_access_key>,
    "AWS_SECRET_ACCESS_KEY" = <your_secret_access_key>,
    "AWS_DEFAULT_REGION"    = <your AWS region>,
    "AWS_ENDPOINT"          = <your AWS endpoint>,
    "AWS_REQUEST_PAYER"     = "requester"
)
```

Sentinel-2/2A 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. Thus, to create data cubes in AWS using Sentinel-2/2A, users need to specify the `s2_resolution` parameter. In the example below, the user selects two Sentinel-2A tiles following the S2A

tiling system. Each S2A tile is an 100x100 km2 orthoimage in UTM/WGS84 projection.

```
# creating a data cube in AWS
s2_cube <- sits_cube(source = "AWS",
                     name = "T20LKP_2018_2019",
                     collection = "sentinel-s2-l2a",
                     tiles = c("20LKP","20LLP"),
                     start_date = as.Date("2018-07-18"),
                     end_date = as.Date("2018-07-23"),
                     s2_resolution = 20
)
```

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 *sfc* or *sf* object from the **sf** package; or (c) a GeoJSON geometry (RFC 7946). When selecting images that compose a data cube based on a `bbox`, sits does does not crop them, and selects the images that intersect with it. When doing the classification using `sits_classify`, only the pixels inside the bounding box will be processed.

In short, the output of the `sits_cube` function is composed of metadata about the images that satisfy the requirements stated in its parameters (spatio-temporal 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.

2.4 Accessing the Brazil Data Cube

The Brazil Data Cube [1] (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 from medium-resolution EO images for all Brazil.

The organization of data cubes in BDC is based on a spatial partition of the territory. This partition is represented by a grid of tiles where every pixel can be efficiently located within a tile. Considering the spatial resolution of the images and aimed at maintaining files that can be easily manageable, three hierarchical grids were defined using an 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 approximately 672×440 km^2 and is used to organize CBERS-4 AWFI collections, each tile represents an image of $10,504 \times 6,865$ pixels. The medium grid is used in the Landsat-8 OLI collections,

[1] http://brazildatacube.org/

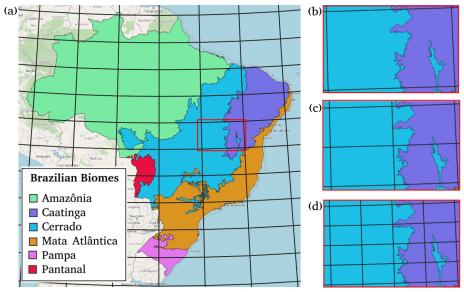


Figure 2.4: Hierarchical BDC tiling system showing overlayed on Brazilian Biomes (a), illustrating that one large tile (b) contains four medium tiles (c) and that medium tile contains four small tiles

the tiles have an extension of 336 x 220 kilometers and images of 11,204 x 7,324 pixels. The small grid present tiles of approximately 168 x 110 kilometers and is used on Sentinel-2 MSI collections (10m), with images of 16,806 x 10,986 pixels.

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

```
Sys.setenv(
    "BDC_ACCESS_KEY" = <your_bdc_access_key>
)
```

Creating a data cube using the BDC is similar to what is required for AWS. The user needs to specific an image collection, a spatio-temporal extent, bands, and optionally a bounding box. In the example, we have selected the "CB4_64_16D_STK-1" collection (CBERS AWFI images at 16 days) that complies with the BDC large tile specification. 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).

```
cbers_tile <- sits_cube(
    source = "BDC",
    collection = "CB4_64_16D_STK-1",
    name = "cbers_022024",
    bands = c("NDVI", "EVI"),
    tiles = "022024",
    start_date = "2018-09-01",
    end_date = "2019-08-28"
)
```

2.5 Defining a data cube using files

To define a data cube using plain files (without STAC information), users should organise files in a single directory. All files should have the same spatial resolution and same 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 time information, each filename needs to include date and band information. For example, `CBERS-4_AWFI_B13_2018-02-02.tif` 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 `"_"`.

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

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

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

For handling time series, the package uses a `sits tibble` to organize time series data with associated spatial information. A `tibble` is a generalization of a `data.frame`, the usual way in R to organize data in tables. Tibbles are part of the `tidyverse`, a collection of R packages designed to work together in data manipulation (Wickham and Grolemund 2017). As an example of how the sits tibble works, the following code shows the first three lines of a tibble containing $1,882$ labelled 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".

```
# data set of samples
data("samples_matogrosso_mod13q1")
samples_matogrosso_mod13q1[1:3,]
```

```
#> # A tibble: 3 x 7
#>   longitude latitude start_date end_date   label   cube    time_series
#>       <dbl>    <dbl> <date>     <date>     <chr>   <chr>   <list>
#> 1     -55.2   -10.8  2013-09-14 2014-08-29 Pasture MOD13Q1 <tibble [23 x 5]>
#> 2     -57.8    -9.76 2006-09-14 2007-08-29 Pasture MOD13Q1 <tibble [23 x 5]>
#> 3     -51.9   -13.4  2014-09-14 2015-08-29 Pasture MOD13Q1 <tibble [23 x 5]>
```

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

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.

```
sits_labels_summary(samples_matogrosso_mod13q1)
```

```
#> # 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` (Wickham and Grolemund 2017) 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".

```
# select NDVI band
samples_ndvi <- sits_select(samples_matogrosso_mod13q1,
                            bands = "NDVI")

# select only samples with Cerrado label
samples_cerrado <- dplyr::filter(samples_ndvi, label == "Cerrado")
```

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 15 samples of "Cerrado" class refer to the same spatial location in consecutive time periods. For this reason, these samples are plotted together.

```
# plot the first 15 samples
plot(samples_cerrado[1:15,])
```

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.

```
# plot all cerrado samples together
plot(samples_cerrado)
```

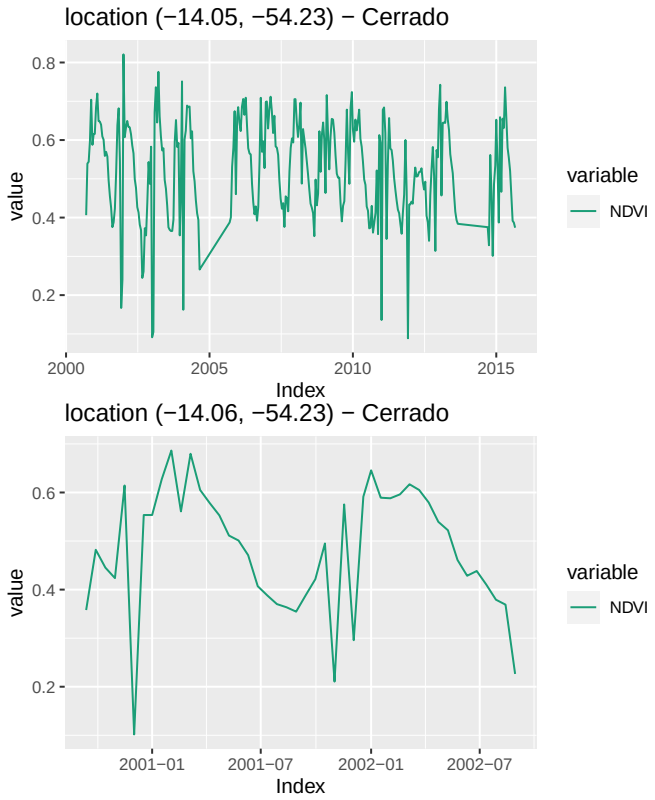


Figure 3.1: Plot of the first 'Cerrado' sample from data set

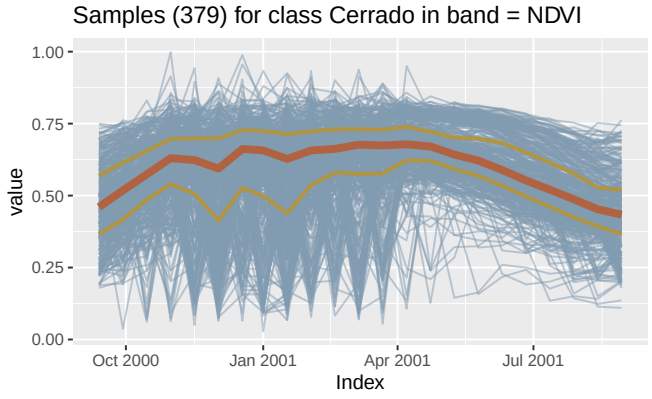


Figure 3.2: Plot of all Cerrado samples from data set

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

```
# Obtain a raster cube with 23 instances for one year
# Select the band "ndvi", "evi" from images available in the "sitsdata" package
data_dir <- system.file("extdata/sinop", package = "sitsdata")

# create a raster metadata file based on the information about the files
raster_cube <- sits_cube(
    source     = "LOCAL",
    satellite  = "TERRA",
    sensor     = "MODIS",
    name       = "Sinop",
    data_dir   = data_dir,
    parse_info = c("X1", "X2", "band", "date"),
)
# a point in the transition forest to pasture in Northern MT
# obtain a time series from the raster cube for this point
series.tb <- sits_get_data(cube      = raster_cube,
                           longitude = -55.57320,
                           latitude  = -11.50566,
                           bands     = c("NDVI", "EVI"))
plot(series.tb)
```

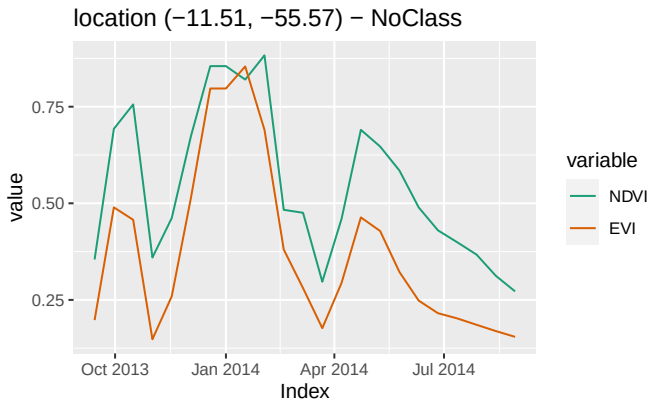


Figure 3.3: NDVI and EVI time series fetched from local raster cube.

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.

```
# retrieve a list of samples described by a CSV file
samples_csv_file <- system.file("extdata/samples/samples_sinop_crop.csv",
                                package = "sits")
# for demonstration, read the CSV file into an R object
samples_csv <- read.csv(samples_csv_file)
# print the first three lines
samples_csv[1:3,]
```

```
#>   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.

```
samples_modis_4bands[1:5,]
```

```
#> # 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.

```
# get the points from a data cube in raster brick format
points <- sits_get_data(cube = raster_cube, file = samples_csv_file)
# show the tibble with the first three points
points[1:3,]
```

```
#> # A tibble: 3 x 7
#>   longitude latitude start_date end_date   label   cube  time_series
#>       <dbl>    <dbl> <date>     <date>     <chr>   <chr> <list>
#> 1     -55.7    -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.

```
# define the input shapefile (consisting of POLYGONS)
shp_file <- system.file("extdata/shapes/agriculture/parcel_agriculture.shp",
                        package = "sitsdata")

# set the start and end dates
start_date <- "2013-09-14"
end_date   <- "2014-08-29"

# define the name of attribute of the shapefile that contains the label
shp_attr <- "ext_na"

# define the number of samples to extract from each polygon
.n_shp_pol <- 10

# read the points in the shapefile and produce a CSV file
samples <- sits_get_data(cube       = raster_cube,
                         file       = shp_file,
                         start_date = start_date,
                         end_date   = end_date,
                         shp_attr   = shp_attr,
                         .n_shp_pol = .n_shp_pol)
samples[1:3,]
```

```
#> # A tibble: 3 x 7
```

```
#>   longitude latitude start_date end_date   label cube  time_series
#>       <dbl>    <dbl> <date>     <date>     <chr> <chr> <list>
#> 1     -55.6    -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 (Lambin and Linderman 2006). 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 (Atkinson et al. 2012). 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, Atkinson et al. (2012) found that the Whittaker and Fourier transform approaches were preferable to the double logistic and asymmetric Gaussian models. They applied the filters to preprocess MERIS NDVI time series for estimating phenological parameters in India. Comparing the same filters as in the previous work, Shao et al. (2016) found that only Fourier transform and Whittaker techniques improved interclass separability for crop classes and significantly improved overall classification accuracy. The authors used MODIS NDVI time series from the Great Lakes region in North America. Zhou et al. (2016) found that 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 (Bradley et al. 2007), wavelet decomposition (Sakamoto et al. 2005), mean-value iteration, ARMD3-ARMA5, and 4253H (Hird and McDermid 2009). Therefore,

any comparative analysis of smoothing algorithms depends on the adopted performance measurement.

One of the main uses of time series filtering is to reduce the noise and miss data produced by clouds in tropical areas. The following examples use data produced by the PRODES project (INPE 2019), 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 Madden (1978), for $d \in [2, 3]$ each C_j smoothing coefficients can be obtained by

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

In general, the Savitzky-Golay filter produces smoother results for a larger value of n and/or a smaller value of d (Chen et al. 2004). The optimal value for these two parameters can vary from case to case. 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.

```
# Take NDVI band of the first sample data set
point_ndvi <- sits_select(point_mt_6bands, band = "NDVI")
# apply Savitzky-Golay filter
point_sg <- sits_sgolay(point_ndvi, length = 15)
# merge the point and plot the series
sits_merge(point_sg, point_ndvi) %>% plot()
```

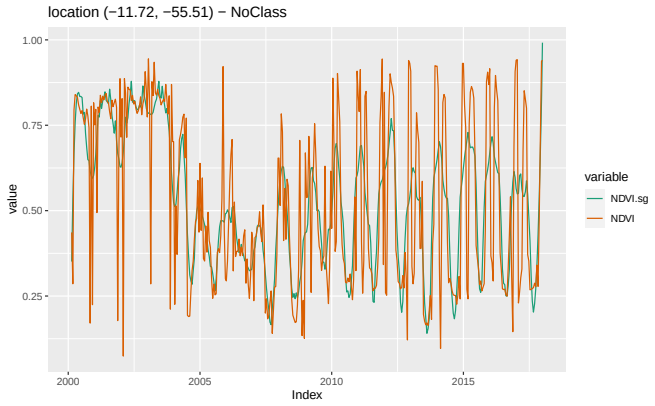


Figure 3.4: Savitzky-Golay filter applied on a multi-year NDVI time series.

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.

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 (Atzberger and Eilers 2011). The Whittaker filter is a balancing between the residual to the original data and the "smoothness" of the fitted curve. The residual, as measured by the sum of squares of all n time series points deviations, is given by

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

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

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

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

```
# Take NDVI band of the first sample data set
point_ndvi <- sits_select(point_mt_6bands, band = "NDVI")
# apply Whitakker filter
point_whit <- sits_whittaker(point_ndvi, lambda = 8)
# merge the point and plot the series
sits_merge(point_whit, point_ndvi) %>% plot()
```

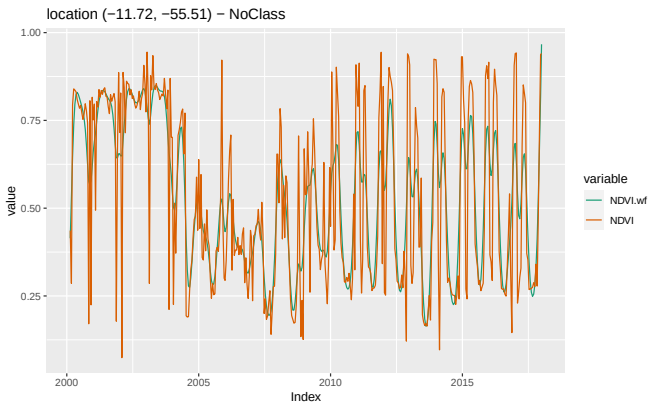


Figure 3.5: Whittaker filter applied on a one-year NDVI time series.

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 (Picoli et al. 2018; Simoes et al. 2020; Parente et al. 2019; Griffiths et al. 2019). 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 (Maxwell, Warner, and Fang 2018); increasing the training sample size results in better classification accuracy (Thanh Noi and Kappas 2018). 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 (Frenay and Verleysen 2014). There are two main sources of noise and errors in satellite image time series. One effect is *feature noise*, caused by clouds and inconsistencies in data calibration. The second effect is *class noise*, 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. Our work thus addresses the question: *How to reduce class noise in large training sets of satellite image time series?*

Many factors lead to *class noise* in *sits*. 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 idealised 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 has been used for many purposes in satellite image time series literature ranging from unsupervised classification and pattern detection (Petitjean et al. 2011). Here, we are interested in the second use of clustering, using it as a way to improve training data to feed machine learning classification models. In this regard, cluster analysis can assist the identification of structural time series patterns and anomalous samples (Frenay and Verleysen 2014).

Agglomerative hierarchical clustering (AHC) is a family of methods that groups elements using a distance function to associate a real value to a pair of elements. From this distance measure, we can compute the dissimilarity between any two elements from a data set. Depending on the distance functions and linkage criteria, the algorithm decides which two clusters are merged at each iteration. AHC approach is suitable for the purposes of samples data exploration due to its visualization power and ease of use (Keogh, Lin, and Truppel 2003). Moreover, AHC does not require a predefined number of clusters as an initial parameter. This is an important feature in satellite image time series clustering since defining the number of clusters present in a set of multi-attribute time series is not straightforward (Aghabozorgi, Shirkhorshidi, and Wah 2015).

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* (Ward 1963). Ward's intuition is that clusters of multivariate observations, such as time series, should be approximately elliptical in shape (Hennig 2015). In `sits`, a dendrogram can be generated by `sits_dendrogram()`. The following codes illustrate how to create, visualize, and cut a dendrogram (for details, see `?sits_dendrogram()`).

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:

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

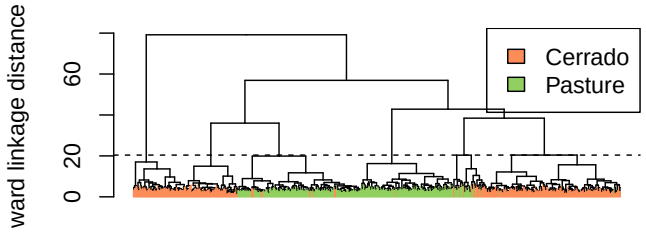
where E is the expected agreement, a random chance correction calculated by

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

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 (Hubert and Arabie 1985), which is usually the case.

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

# show clusters samples frequency
sits::sits_cluster_frequency(clusters)
```

```
#>
#>              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.

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

# show new clusters samples frequency
sits::sits_cluster_frequency(clusters_new)
```

```
#>
#>             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.

```
# clear clusters, leaving only the majority class in each cluster
clean <- sits::sits_cluster_clean(clusters)
# show clusters samples frequency
sits_cluster_frequency(clean)
```

```
#>
#>             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 (Kohonen 1990), 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 (Kohonen 2013). 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.

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 (**???**) 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.

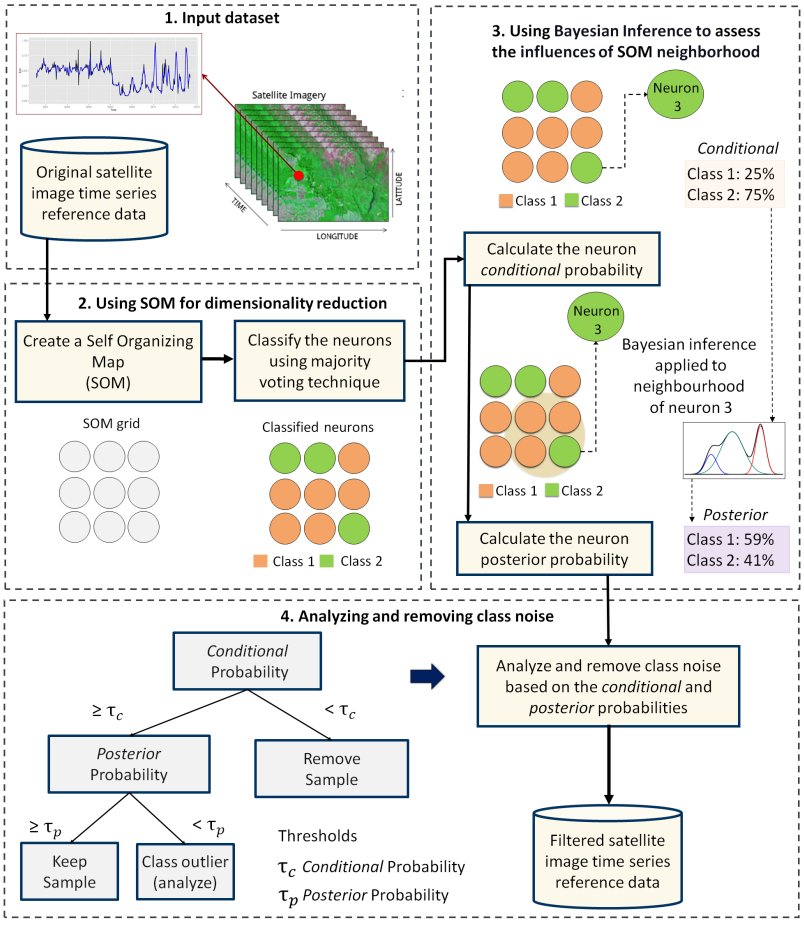


Figure 4.1: Using SOM for class noise reduction

- 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].

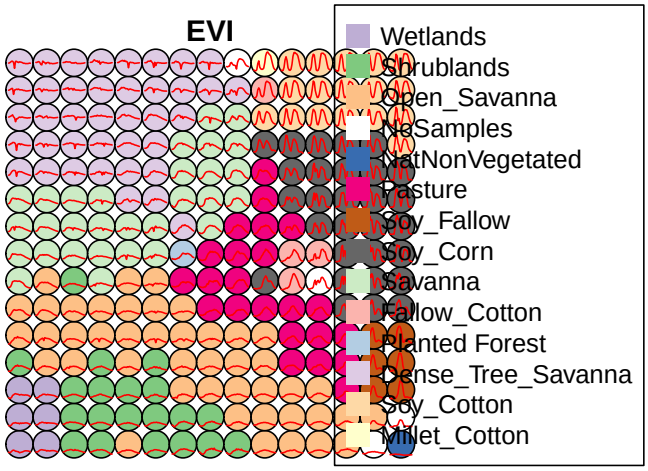
As an example of the use of SOM clustering for quality control of samples, we take a dataset containing a tibble with time series samples for the Cerrado region of Brazil, the second largest biome in South America with an area of more than 2 million km2. The training samples were collected by ground 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 (in Portuguese *Cerrado*, Dense Tree Savanna (in Portuguese *Cerradao*), Open Savanna (in Portuguese *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.

```
# take only 10% of the samples
samples_cerrado_mod13q1_reduced <- sits_sample(samples_cerrado_mod13q1, frac = 0.1)
# clustering time series using SOM
som_cluster <-
    sits_som_map(
        samples_cerrado_mod13q1_reduced,
        grid_xdim = 15,
        grid_ydim = 15,
        alpha = 1.0,
        distance = "euclidean",
        rlen = 100
    )
```

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.

```
plot(som_cluster)
```



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.

```
new_samples <- sits_som_clean_samples(som_cluster,
                                      prior_threshold = 0.6,
                                      posterior_threshold = 0.6,
                                      keep = c("clean", "analyze"))
# find out how many samples are evaluated as "clean" or "analyze"
new_samples %>%
  dplyr::group_by(eval) %>%
  dplyr::summarise(count = dplyr::n(), .groups = "drop")
```

```
#> # 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

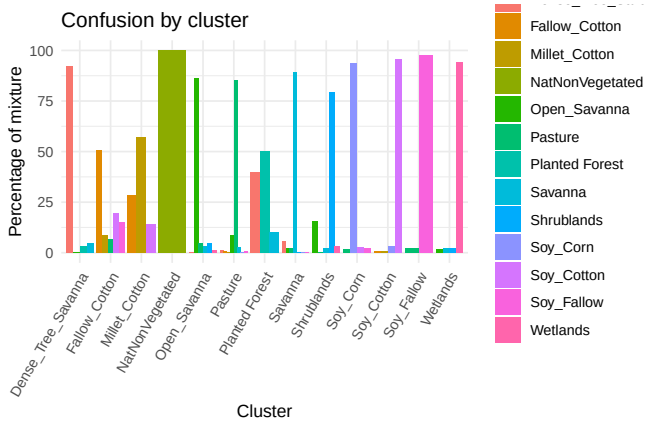
To compare the accuracy of the original and clean samples, we run a 5-fold validation on the original and on the cleaned sample. We use the function `sits_kfold_validate`. As the results show, the SOM procedure is useful, since the global accuracy improves from 91% to 95%.

```
assess_orig <- sits_kfold_validate(samples_cerrado_mod13q1_reduced,
                                   ml_method = sits_svm())
```

```
assess_new <- sits_kfold_validate(new_samples,
                                   ml_method = sits_svm())
```

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.

```
# evaluate the misture in the SOM clusters
cluster_mixture <- sits_som_evaluate_cluster(som_cluster)
# plot the mixture information.
plot(cluster_mixture)
```



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 (Mountrakis, Im, and Ogole 2011) and random forests (Belgiu and Dragut 2016) 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 (Brown et al. 2013; Kastens et al. 2017). Using software such as TIMESAT (Jonsson and Eklundh 2004), they derive a small set of phenological parameters from vegetation indexes, like the beginning, peak, and length of the growing season (Estel et al. 2015; Pelletier et al. 2016).

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 (Gomez, White, and Wulder 2016). 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 (Brown et al. 2013), (Kastens et al. 2017), (Estel et al. 2015), (Pelletier et al. 2016). These phenological parameters are then fed in specialized classifiers such as TIMESAT (Jonsson and Eklundh 2004). These approaches do not use the power of advanced statistical learning techniques to work on high-dimensional spaces with big training data sets (James et al. 2013). Package *sits** uses the full depth of satellite image time series to create larger dimensional spaces, an approach we consider to be more appropriate to use with machine learning.

`sits` 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 (Pelletier, Webb, and Petitjean 2019) . In a recent review of machine learning methods to classify remote sensing data (Maxwell, Warner, and Fang 2018), the authors note that many factors influence the performance of these classifiers, including the size and quality of the training dataset, the dimension of the feature space, and the choice of the parameters. We support both *space-first, time-later* and *time-first, space-later* approaches. Therefore, the `sits` package provides functionality to explore the full depth of satellite image time series data.

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

SITS provides support for the classification of both individual time series as well as data cubes. The following machine learning methods are available in SITS:

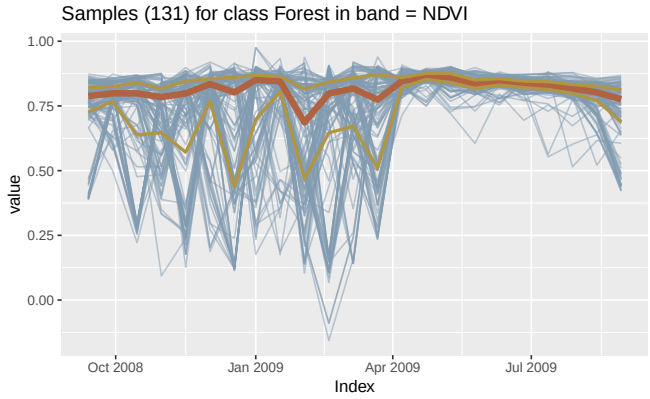
- Linear discriminant analysis (`sits_lda`)
- Quadratic discriminant analysis (`sits_qda`)
- Multinomial logit and its variants 'lasso' and 'ridge' (`sits_mlr`)
- Support vector machines (`sits_svm`)
- Random forests (`sits_rfor`)
- Extreme gradient boosting (`sits_xgboost`)
- Deep learning (DL) using multi-layer perceptrons (`sits_deeplearning`)
- DL with 1D convolutional neural networks (`sits_FCN`)
- DL combining 1D CNN and multi-layer perceptron networks (`sits_TempCNN`)
- DL using a combination of long-short term memory (LSTM) and 1D CNN (`sits_LSTM-FCN`)

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" (Picoli et al. 2018), 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.

```
data("samples_matogrosso_mod13q1")
# Select a subset of the samples to be plotted
# Retrieve the set of samples for the Mato Grosso region
samples_matogrosso_mod13q1 %>%
    sits_select(bands = "NDVI") %>%
    dplyr::filter(label == "Forest") %>%
    plot()
```

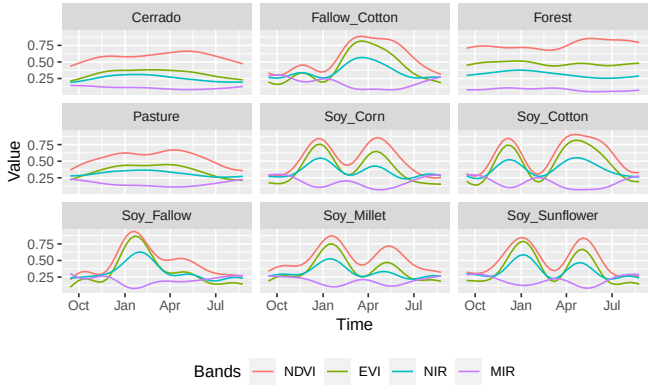


An alternative to visualise the samples is to estimate a statistical approximation to an idealized pattern based on a generalised additive model (GAM). A GAM is a linear model in which the linear predictor depends linearly on a smooth function of the predictor variables

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

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`(Maus et al. 2019), which implements the TWDTW time series matching method described in Maus et al. (2016). The resulting patterns can be viewed using `plot`.

```
# Select a subset of the samples to be plotted
samples_matogrosso_mod13q1 %>%
    sits_patterns() %>%
    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`.

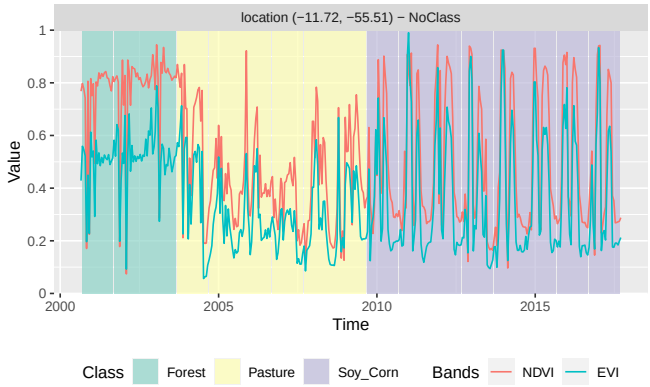
5.4 Random forests

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(Wright et al. 2017); its main parameters is `num_trees` (number of trees to grow).

```
# Retrieve the set of samples (provided by EMBRAPA) from the
# Mato Grosso region for train the Random Forest model.
rfor_model <- sits_train(data = samples_matogrosso_mod13q1,
                         ml_method = sits_rfor(num_trees = 1000))
# retrieve a point to be classified
point_mt_4bands <- sits_select(point_mt_6bands,
                               bands = c("NDVI", "EVI", "NIR", "MIR"))
# Classify using Random Forest model and plot the result
point_class <- sits_classify(point_mt_4bands, rfor_model)
plot(point_class, bands = c("NDVI", "EVI"))
```




```
# show the results of the prediction
sits_show_prediction(point_class)
```

```
#> # 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 (Hastie, Tibshirani, and J 2009). 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 (Hastie, Tibshirani, and J 2009). Random forest classifiers can be quite useful to provide a baseline to compare with more sophisticated methods.

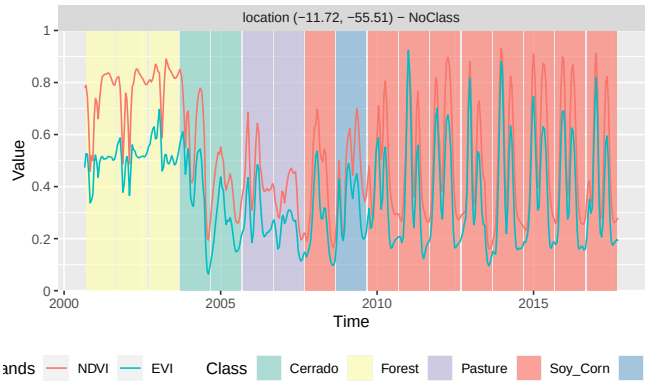
5.5 Support Vector Machines

Given a multidimensional data set, the Support Vector Machine (SVM) method finds an optimal separation hyperplane that minimizes misclassifications (Cortes and Vapnik 1995). 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 (Mountrakis, Im, and Ogole 2011).

In `sits`, SVM is implemented as a wrapper of `e1071` R package that uses the `LIBSVM` implementation (Chang and Lin 2011), `sits` adopts the *one-against-one* method for multiclass classification. For a q class problem, this method creates $q(q-1)/2$ SVM binary models, one for each class pair combination and tests any unknown input vectors throughout all those models. The overall result is computed by a voting scheme. 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.

```
# Filter the data slightly to reduce noise without reducing variability
samples_filtered <- sits_whittaker(samples_matogrosso_mod13q1, lambda = 0.5,
                                   bands_suffix = "")
# Train a machine learning model for the mato grosso dataset using SVM
# The parameters are those of the "e1071:svm" method
svm_model <- sits_train(samples_matogrosso_mod13q1,
                        ml_method = sits_svm(kernel = "radial",
                                             cost = 100))
# Classify using SVM model and plot the result
class <- point_mt_4bands %>%
    sits_whittaker(lambda = 0.5, bands_suffix = "") %>%
    sits_classify(svm_model) %>%
    plot(bands = c("NDVI", "EVI"))
```



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 contraints violation) from 100 to 1, the results can be strinkgly different. Such sensitity to the input parameters points to a limitation when using the SVM method for classifying time series.

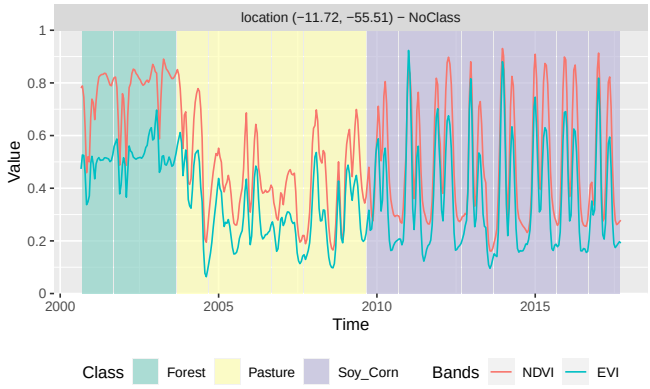
5.6 Extreme Gradient Boosting

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 (Hastie, Tibshirani, and J 2009). 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 agradient descent algorithm. Extreme gradient boosting (**???**), 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.

```
# Train a machine learning model for the mato grosso dataset using XGBOOST
# The parameters are those of the "xgboost" package
xgb_model <- sits_train(samples_filtered, sits_xgboost(verbose = 0))
# Classify using SVM model and plot the result
point_mt_4bands %>%
    sits_whittaker(lambda = 0.50, bands_suffix = "") %>%
    sits_classify(xgb_model) %>%
    plot(bands = c("NDVI", "EVI"))
```



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 (Chollet and Allaire 2018) 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 (Goodfellow, Bengio, and Courville 2016).

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

Three other important parameters for an MLP are: (a) the activation function;

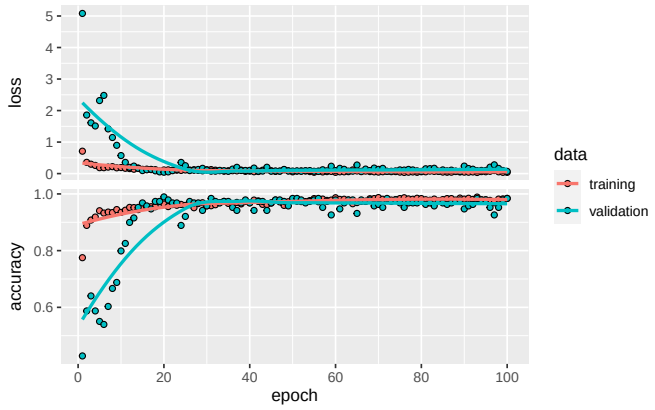
(b) the optimization method; (c) the dropout rate. The activation function the activation function of a node defines the output of that node given an input or set of inputs. Following standard practices (Goodfellow, Bengio, and Courville 2016), we recommend the use of the "relu" and "elu" functions.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

The dropout rates have a huge impact on the performance of MLP classifiers. Dropout is a technique for randomly dropping units from the neural network during training (Srivastava et al. 2014). By randomly discarding some neurons, dropout reduces overfitting. It is a counter-intuitive idea that works well. Since the purpose of a cascade of neural nets is to improve learning as more data is acquired, discarding some of these neurons may seem a waste of resources. In fact, as experience has shown (Goodfellow, Bengio, and Courville 2016), 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 (Wang, Yan, and Oates 2017) 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.

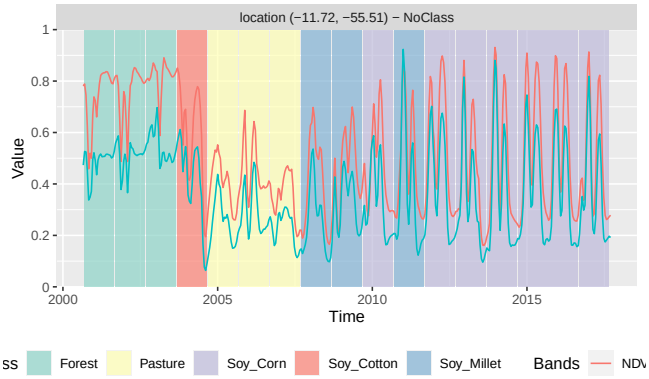
In this and in the following examples of using deep learning classifiers, both the training samples and the point to be classified are filtered with `sits_whittaker` with a small smoothing parameter (lambda = 0.5). In our experiments, we found that deep learning classifiers are not as robust to noise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

```
# train a machine learning model for the Mato Grosso data using an MLP model

mlp_model <- sits_train(samples_filtered,
                        sits_deeplearning(
                        layers           = c(512, 512, 512, 512, 512, 512),
                        activation       = "elu",
                        dropout_rates    = c(0.10, 0.20, 0.25, 0.30, 0.35, 0.40),
                        epochs           = 100,
                        batch_size       = 64,
                        verbose          = 0,
                        validation_split = 0.1) )

# show training evolution
plot(environment(mlp_model)$history)
```



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

```
# Classify using DL model and plot the result
point_mt_4bands <- sits_select(point_mt_6bands,
                               bands = c("NDVI", "EVI", "NIR", "MIR"))
point_mt_4bands %>%
    sits_whittaker(lambda = 0.5, bands_suffix = "") %>%
    sits_classify(mlp_model) %>%
    plot(bands = c("ndvi", "evi"))
```



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. Rußwurm and Korner (2017) 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 Pelletier, Webb, and Petitjean (2019). 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 (Pelletier, Webb, and Petitjean 2019), the authors compare favourably the tempCNN model with the Recurrent Neural Network proposed by Russwurm and Korner (2018) for land use classification. The figure below shows the architecture of the tempCNN model.

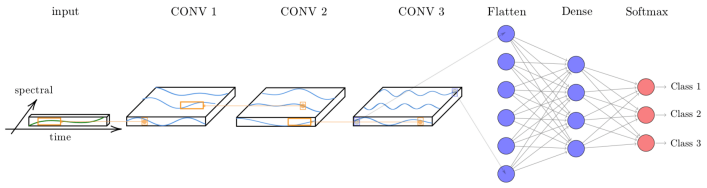
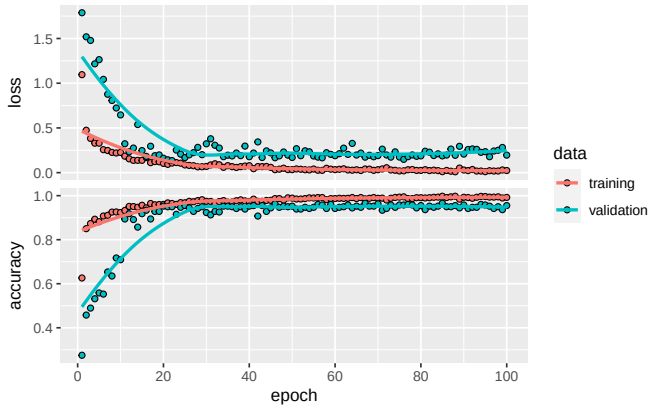


Figure 5.1: Structure of tempCNN architecture (source: Pelletier et al.(2019))

The function `sits_tempCNN` implements the model. The code has been derived from the Python source provided by the authors (https://github.com/charlotte-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 Pelletier, Webb, and Petitjean (2019). 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.

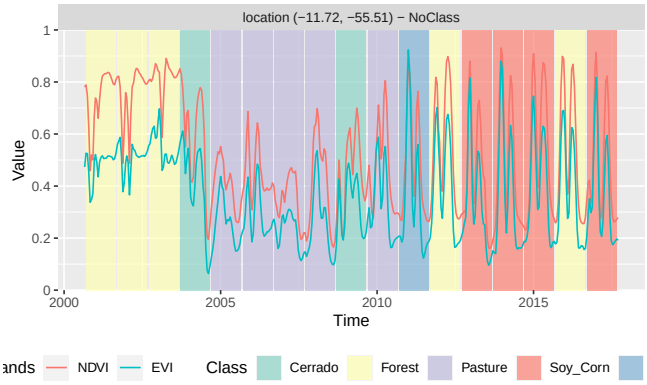
```
# train a machine learning model using tempCNN
tCNN_model <- sits_train(samples_filtered,
                         sits_TempCNN(
                            cnn_layers           = c(96, 96, 96),
                            cnn_kernels          = c(3, 3, 3),
```

```
                          cnn_activation      = 'relu',
                          cnn_L2_rate         = 1e-06,
                          cnn_dropout_rates   = c(0.20, 0.20, 0.20),
                          mlp_layers          = c(512, 512, 512),
                          mlp_activation      = 'relu',
                          mlp_dropout_rates   = c(0.10, 0.20, 0.30),
                          epochs              = 100,
                          batch_size          = 64,
                          validation_split    = 0.2,
                          verbose             = 0) )

# show training evolution
plot(environment(tCNN_model)$history)
```



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

```
# Classify using DL model and plot the result
point_mt_4bands <- sits_select(point_mt_6bands,
                               bands = c("NDVI", "EVI", "NIR", "MIR"))
class <- point_mt_4bands %>%
    sits_whittaker(lambda = 0.5, bands_suffix = "") %>%
    sits_classify(tCNN_model) %>%
    plot(bands = c("ndvi", "evi"))
```



The tempCNN model 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 LSTM Convolutional Networks for Time Series Classification

Given the success of 1D-CNN networks for time series classification, there have been a number of variants proposed in the literature. One of these variants is the LSTM-CNN network (Karim et al. 2018), where a fullCNN is combined with long short term memory (LSTM) recurrent neural network. LSTMs are an improved version of recurrent neural networks (RNN). An RNN is a neural network that includes a state vector, which is updated every time step. In this way, a RNN combines an input vector with information that is kept from all previous inputs. One can conceive of RNN as networks that have loops, allowing information to be passed from one step to to the next. In theory, a RNN would be able to handle long-term dependencies between elements of the input vectors. In practice, they are prone to the exhibit the "vanishing gradient" effect. As discussed above, in deep neural networks architectures with gradient descent optimization the gradient function can approach zero, thus impeding training to be done efficiently. LSTM improve on RNN architecture by including the additional feature of being able to regulate whether or not new information should be included on the cell state. LSTM unit also include a forget gate, which is able to discard the previous information stored in the cell state. Thus, a LSTM unit is able to remember values over arbitrary time intervals.

Karim et al. (2019) consider that LSTM networks are "well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series". The authors proposed a mixed LSTM-CNN architecture, composed of two paralel

data streams: a 3-step CNN such as the one implemented in `sits_FCN` (see above) combined with a data stream consisting of an LSTM unit, as shown in the figure below. In Karim et al. (2018), the authors argue the LSTM-CNN model is capable of a better performance in the UCR/UEA time series test set than architectures such as ResNet and 1DCNN.

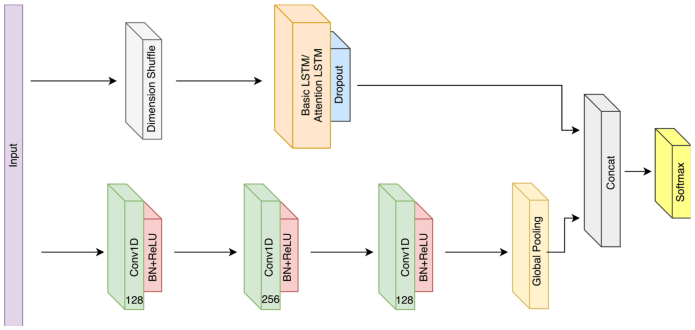
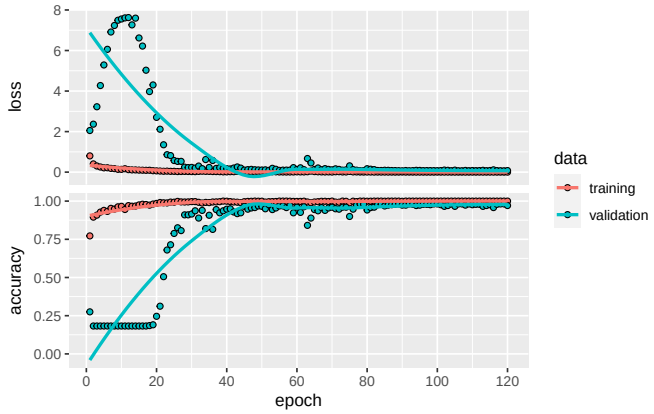


Figure 5.2: LSTM Fully Convolutional Networks for Time Series Classification (source: Karim et al.(2019))

In the SITS package, the combined LSTM-CNN architecture is implemented by the `sits_LSTM_CNN` function. The default values are similar those proposed by Karim et al. (2019). The parameter `lstm_units` controls the number of units in the LSTM cell at every time step of the network. Karim et al. (2019) proposes an LSTM with 8 units, each with a dropout rate of 80%, which are controlled by parameters `lstm_units` and `lstm_dropout`. In initial experiments, we got a better performance with an LSTM with 16 units. The CNN layers have filter sizes of {128, 256, 128} and kernel convolution sizes of {8, 5, 3}, controlled by the parameters `cnn_layers` and `cnn_kernels`. One should experiment with these parameters, and consider the simulations carried out by Pelletier, Webb, and Petitjean (2019) (see above), where the authors found that an FCN network of sizes {64, 64, 64} with kernels sizes of {5, 5, 5} had best performance in their case study. In this example, the estimated accuracy of the model was 94.7%.

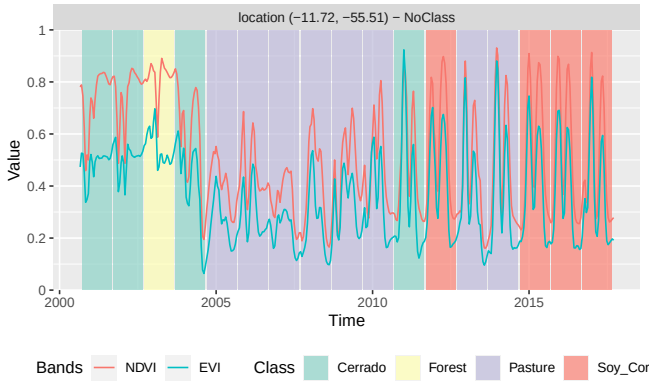
```
lstm_fcn_model <- sits_train(samples_filtered,
                        sits_LSTM_FCN(
                           lstm_units          = 16,
                           lstm_dropout        = 0.30,
                           cnn_layers          = c(128, 256, 128),
                           cnn_kernels         = c(3, 3, 3),
                           activation          = 'relu',
                           epochs              = 120,
                           batch_size          = 64,
                           validation_split    = 0.2,
                           verbose             = 0) )
```

```
# show training evolution
plot(environment(lstm_fcn_model)$history)
```



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

```
# Classify using DL model and plot the result
point_mt_4bands <- sits_select(point_mt_6bands,
                               bands = c("NDVI", "EVI", "NIR", "MIR"))
class <- point_mt_4bands %>%
    sits_whittaker(lambda = 0.5, bands_suffix = "") %>%
    sits_classify(lstm_fcn_model) %>%
    plot(bands = c("ndvi", "evi"))
```



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

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

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

classification algorithms. Given the widespread availability of large data sets of Earth observation data, there is a growing interest in organising large sets of data into "data cubes".

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

6.2 Defining a data cube using files organised as raster bricks

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

Since GeoTiff files do not contain information about satellites and sensors, it is best practice to provide information on satellite and sensor.

```
# Obtain a raster cube with 23 instances for one year
# Select the band "ndvi", "evi" from images available in the "sitsdata" package
data_dir <- system.file("extdata/sinop", package = "sitsdata")

# create a raster metadata file based on the information about the files
raster_cube <- sits_cube(source = "LOCAL",
                         satellite = "TERRA",
                         sensor  = "MODIS",
                         name = "Sinop",
                         data_dir = data_dir,
                         parse_info = c("X1", "X2", "band", "date"),
)

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

```
#> # A tibble: 1 x 3
#>   source satellite sensor
#>   <chr>  <chr>     <chr>
#> 1 LOCAL  TERRA     MODIS
```

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

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

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

6.3 Classification using machine learning

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

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

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

The SITS package provides a common interface to all machine learning models, using the `sits_train` function. this function takes two parameters: the input

data samples and the ML method (`ml_method`), as shown below. After the model is estimated, it can be used to classify individual time series or full data cubes using the `sits_classify` function. In the examples that follow, we show how to apply each method for the classification of a single time series. Then, we disscuss how to classify full data cubes.

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

- Linear discriminant analysis (`sits_lda`)
- Quadratic discriminant analysis (`sits_qda`)
- Multinomial logit and its variants 'lasso' and 'ridge' (`sits_mlr`)
- Support vector machines (`sits_svm`)
- Random forests (`sits_rfor`)
- Extreme gradient boosting (`sits_xgboost`)
- Deep learning (DL) using multi-layer perceptrons (`sits_deeplearning`)
- DL combining 1D CNN and multi-layer perceptron networks (`sits_tempCNN`)
- DL using a combination of long-short term memory (LSTM) and 1D CNN (`sits_LSTM_FCN`)

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

6.4 Cube classification

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

To classify a data cube, use the function `sits_classify()` as described below. This function works with cubes built from raster bricks. 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.

6.4.1 Steps for cube classification

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

1. Select a set of training samples.
2. Train a machine learning model

3. Classify the data cubes using the model, producing a data cube with class probabilities.
4. Label the cube with probabilities, including data smoothing if desired.

6.4.2 Adjustments for improved performance

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

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

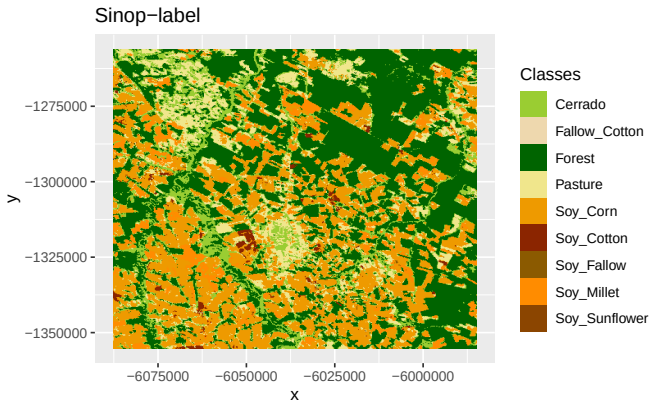
Based on current experience, the classification of a MODIS tile (4800 x 4800) with four bands and 400 time instances, covering 15 years of data, using SVM with a training data set of about 10,000 samples, takes about 24 hours using 20 cores and a memory size of 60 GB, in a server with 2.4GHz Xeon CPU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

```
# select the bands "ndvi", "evi"
samples_2bands <- sits_select(samples_matogrosso_mod13q1, bands = c("NDVI", "EVI"))

#select a rfor model
xgb_model <- sits_train(samples_2bands, ml_method = sits_xgboost())
```

```
#> [1]  train-mlogloss:2.009944+0.000891    test-mlogloss:2.022905+0.001490
#> Multiple eval metrics are present. Will use test_mlogloss for early stopping.
#> Will train until test_mlogloss hasn't improved in 20 rounds.
#>
#> [11] train-mlogloss:0.620737+0.002764    test-mlogloss:0.755200+0.005152
#> [21] train-mlogloss:0.207436+0.001456    test-mlogloss:0.364395+0.009251
#> [31] train-mlogloss:0.094411+0.000752    test-mlogloss:0.251361+0.008440
#> [41] train-mlogloss:0.059334+0.000473    test-mlogloss:0.211024+0.009399
#> [51] train-mlogloss:0.048914+0.000651    test-mlogloss:0.197786+0.009918
#> [61] train-mlogloss:0.045721+0.000552    test-mlogloss:0.194586+0.010300
#> [71] train-mlogloss:0.044009+0.000838    test-mlogloss:0.192748+0.010751
#> [81] train-mlogloss:0.043003+0.000582    test-mlogloss:0.191353+0.011066
#> [91] train-mlogloss:0.042301+0.000387    test-mlogloss:0.190409+0.011007
```

```
#> [100]    train-mlogloss:0.042005+0.000349    test-mlogloss:0.189919+0.011470
```

```
data_dir <- system.file("extdata/sinop", package = "sitsdata")

# create a raster metadata file based on the information about the files
sinop <- sits_cube(source = "LOCAL",
                   satellite = "TERRA",
                   sensor  = "MODIS",
                   name = "Sinop",
                   data_dir = data_dir,
                   parse_info = c("X1", "X2", "band", "date"))

# classify the raster image
sinop_probs <- sits_classify(sinop, ml_model = xgb_model,
                             memsize = 4, multicores = 1,
                             output_dir = tempdir())

# label the probability file
# (by default selecting the class with higher probability)
sinop_label <- sits_label_classification(sinop_probs, output_dir = tempdir())
plot(sinop_label, title = "Sinop-label")
```



6.5 Final remarks

Current approaches to image time series analysis still use limited number of attributes. A common approach is deriving a small set of phenological parameters from vegetation indices, like beginning, peak, and length of growing season (Brown et al. 2013), (Kastens et al. 2017), (Estel et al. 2015), (Pelletier et al. 2016). These phenological parameters are then fed in specialized classifiers such as TIMESAT (Jonsson and Eklundh 2004). These approaches do not use the power of advanced statistical learning techniques to work on high-dimensional spaces with big training data sets (James et al. 2013).

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

Part IV

Post classification

Chapter 7

Post classification smoothing using Bayesian techniques in SITS

This chapter describes a Bayesian smoothing method to reclassify the pixels, based on the machine learning probabilities. We consider that the output of the machine learning algorithm provides, for each pixel, the information on the probability of such pixel belonging to each of the target classes. Usually, we label a pixel as being of a given class if the associated class probability is higher than the probability of it belonging to any of the other classes. 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.

7.1 Introduction

Image classification post-processing has been defined as "a refinement of the labelling in a classified image in order to enhance its classification accuracy" (Huang et al. 2014). 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" (Schindler 2012): nearby pixels tend to have the same label. To put this assumption in practice, smoothing methods use the neighbourhood information to remove outliers and enhance consistency in the resulting product.

Smoothing methods are an important complement to machine learning algorithms for image classification. Since these methods are mostly pixel-based, it is useful to complement them with post-processing smoothing to include spatial information in the result. A traditional choice for smoothing classified images is the majority filter, where the class of the central pixel is replaced by the most frequent class of the neighbourhood. This technique is rather simplistic; more sophisticated methods use class probabilities. 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.

In this vignette, we introduce a Bayesian smoothing method, which provides the means to incorporate prior knowledge in data analysis. 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. As stated by (**???**): "In the Bayesian paradigm, degrees of belief in states of nature are specified. Bayesian statistical methods start with existing 'prior' beliefs, and update these using data to give 'posterior' beliefs, which may be used as the basis for inferential decisions". Bayesian inference has now been established as a major method for assessing probability.

7.2 Overview of Bayesian estimattion

Most applications of machine learning methods for image classification use only the categorical result of the classifier which is the most probable class. The proposed method uses all class probabilities to compute our confidence in the result. 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 Smmothing using Bayes' rule

Given the general principles of Bayesian inference, smoothing of classified images requires estimating the *likelihood* and the *prior* probability of each pixel belonging to each class. In order to express our problem in a more tractable form, we perform data transformations. More formally, consider a set of K classes that are candidates for labelling each pixel. Let $p_{i,k}$ be the probability of pixel i belonging to class k, $k = 1, \ldots, K$. We have

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

We label a pixel p_i as being of class k if

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

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

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

where $p_{i,k}$ is the probability of class k. 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 neighbours and reduced the variance of the estimated class for each pixel.

We further make the transformation

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

which measures the *logit* (log of the odds) associated to classifying the pixel i as being of class k. The support of $x_{i,k}$ is $\mathbb{R}$. Let V_i be a spatial neighbourhood for pixel i. We use Bayes' rule to update the value $x_{i,k}$ based on the neighbourhood, assuming independence between the classes. In this way, the update is performed for each class k at a time.

For each pixel, the random variable that describes the class probability is denoted by $\theta_{i,k}$. Therefore, we can express Bayes' rule for each combination of pixel and class as

$$\pi(\theta_{i,k}|x_{i,k}) \propto \pi(x_{i,k}|\theta_{i,k})\pi(\theta_{i,k}).$$

We assume the prior distribution $\pi(\theta_{i,k})$ and the likelihood $\pi(x_{i,k}|\theta_{i,k})$ are modelled by Gaussian distributions. In this case, the posterior will also be a Gaussian distribution. To estimate the prior distribution for a pixel, we consider that all pixels in the spatial neighbourhood V_i of pixel i follow the same Gaussian distribution with parameters $m_{i,k}$ and $s^2_{i,k}$. Thus, the prior is expressed as

$$\theta_{i,k} \sim N(m_{i,k}, s^2_{i,k}).$$

In the above equation, the parameter $m_{i,k}$ is the local mean of the probability distribution of values for class k and $s^2_{i,k}$ is the local variance for class k. We estimate the local mean and variance by considering the neighbouring pixels in space. Let $\#(V_i)$ be the number of elements in the spatial neighbourhood V_i. The local mean is calculated by:

$$m_{i,k} = \frac{\sum\limits_{j \in V_i} x_{j,k}}{\#(V_i)}$$

and the local variance by

$$s^2_{i,k} = \frac{\sum\limits_{j \in V_i} [x_{j,k} - m_{i,k}]^2}{\#(V_i) - 1}.$$

We also consider that the likelihood follows a normal distribution. We take the likelihood as being the distribution of $x_{i,k}$, conditioned by the local variable $\theta_{i,k}$. This conditional distribution is also taken as normal with parameters $\theta_{i,k}$ and σ^2_k, expressed as

$$x_{i,k}|\theta_{i,k} \sim N(\theta_{i,k}, \sigma^2_k)$$

In the likelihood equation above, σ^2_k is a hyper-parameter that controls the level of smoothness.The Bayesian smoothing estimates the value of $\theta_{i,k}$ conditioned by the data $x_{i,k}$. This is the updated value of the logit of class probability for class k of pixel i. Since both the prior and the likelihood are assumed as Gaussian distribution, based on Bayesian statistics the value of conditional mean for a normal distribution is given by:

$$E[\theta_{i,k}|x_{i,k}] = \frac{m_{i,t} \times \sigma^2_k + x_{i,k} \times s^2_{i,k}}{\sigma^2_k + s^2_{i,k}}$$

which can also be expressed as

$$E[\theta_{i,k}|x_{i,k}] = \left[\frac{s_{i,k}^2}{\sigma_k^2 + s_{i,k}^2}\right] \times x_{i,k} + \left[\frac{\sigma_k^2}{\sigma_k^2 + s_{i,k}^2}\right] \times m_{i,k}$$

The updated value for the class probability of the pixel is a weighted average between the original logit value $x_{i,k}$ and the mean of the class logits $m_{i,k}$ for the neighboring pixels. When the local class variance of the neighbors $s_{i,k}^2$ is high relative to the smoothing factor σ_k^2, 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 variance $s_{i,k}^2$ decreases relative to the smoothness factor σ_k^2, 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.

The hyperparameter σ_k^2 sets the level of smoothness. If σ_k^2 is zero, the smoothed value $E[\mu_{i,,k}|l_{i,k}]$ is equal to the pixel value $l_{i,k}$. Higher values of σ_k^2 will cause the assignment of the local mean to the pixel updated probability. In practice, σ_k^2 is a user-controlled parameter that will be set by users based on their knowledge of the region to be classified. In our case, after some classification tests, we decided to set the parameters V as the Moore neighborhood where each pixel is connected to all those pixels with Chebyshev distance of 1, and $\sigma_k^2 = 20$ for all k. This level of smoothness showed the best performance in the technical validation.

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 more probability images, one per requested clasification interval. The next step is to apply the `sits_label_classification` function. By default, this function selects the most likely class for each pixel considering only the probabilities of each class for each pixel. To allow for Bayesian smooting, it suffices to include the `smoothing = bayesian` parameter. If desired, the `variance` parameter (associated to the hyperparameter σ_k^2 described above) can control the degree of smoothness.

```
# Retrieve the data for the Mato Grosso state
data("samples_matogrosso_mod13q1")

# select the bands "ndvi", "evi"
samples_2bands <- sits_select(samples_matogrosso_mod13q1, bands = c("NDVI", "EVI"))
```

```
#select a rfor model
xgb_model <- sits_train(samples_2bands, ml_method = sits_xgboost())
```

```
#> [1]  train-mlogloss:2.009944+0.000891    test-mlogloss:2.022905+0.001490
#> Multiple eval metrics are present. Will use test_mlogloss for early stopping.
#> Will train until test_mlogloss hasn't improved in 20 rounds.
#>
#> [11] train-mlogloss:0.620737+0.002764    test-mlogloss:0.755200+0.005152
#> [21] train-mlogloss:0.207436+0.001456    test-mlogloss:0.364395+0.009251
#> [31] train-mlogloss:0.094411+0.000752    test-mlogloss:0.251361+0.008440
#> [41] train-mlogloss:0.059334+0.000473    test-mlogloss:0.211024+0.009399
#> [51] train-mlogloss:0.048914+0.000651    test-mlogloss:0.197786+0.009918
#> [61] train-mlogloss:0.045721+0.000552    test-mlogloss:0.194586+0.010300
#> [71] train-mlogloss:0.044009+0.000838    test-mlogloss:0.192748+0.010751
#> [81] train-mlogloss:0.043003+0.000582    test-mlogloss:0.191353+0.011066
#> [91] train-mlogloss:0.042301+0.000387    test-mlogloss:0.190409+0.011007
#> [100]    train-mlogloss:0.042005+0.000349    test-mlogloss:0.189919+0.011470
```

```
data_dir <- system.file("extdata/sinop", package = "sitsdata")

# create a raster metadata file based on the information about the files
raster_cube <- sits_cube(source = "LOCAL",
                  satellite = "TERRA",
                  sensor  = "MODIS",
                  name = "Sinop",
                  data_dir = data_dir,
                  parse_info = c("X1", "X2", "band", "date"),
)

# classify the raster image and generate a probability file
raster_probs <- sits_classify(raster_cube, ml_model = xgb_model,
                              memsize = 4, multicores = 2,
                              output_dir = tempdir())

# smooth the result with a bayesian filter
raster_probs_bayes <- sits_smooth(raster_probs, output_dir = tempdir())

# label the smoothed probability images
raster_class <- sits_label_classification(raster_probs_bayes, output_dir = tempdir())
```

The result is shown below.

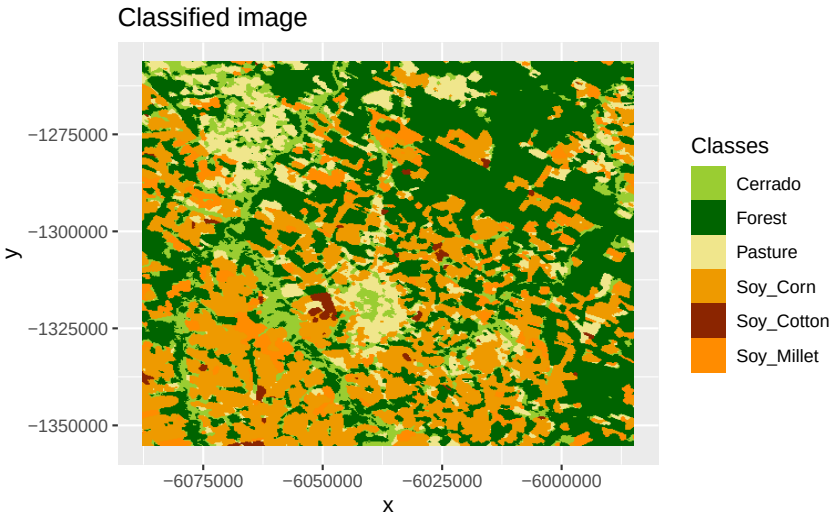


Figure 7.1: Classified image post-processed with Bayesian smoothing. The image coordinates (*meters*) shown at vertical and horizontal axis are in MODIS sinusoidal projection.

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 (Hastie, Tibshirani, and J 2009).

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

A natural question that arises is: *how good is this estimation?* According to Hastie, Tibshirani, and J (2009), there is a bias-variance trade-off in choice of k. If k is set to the number of samples, we obtain the so-called *leave-one-out* validation, the estimator gives a low bias for the true expected error, but

produces a high variance expectation. This can be computational expensive as it requires the same number of fitting process as the number of samples. On the other hand, if we choose $k = 2$, we get a high biased expected prediction error estimation that overestimates the true prediction error, but has a low variance. The recommended choices of k are 5 or 10 (Hastie, Tibshirani, and J 2009), which somewhat overestimates the true prediction error.

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

```
# perform a five fold validation for the "cerrado_2classes" data set
# Random Forest machine learning method using default parameters
prediction.mx <- sits_kfold_validate(cerrado_2classes,
                                     folds = 5,
                                     ml_method = sits_rfor())
```

8.2 Comparing different validation methods

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.

```
# Retrieve the set of samples for the Mato Grosso region (provided by EMBRAPA)
data("samples_matogrosso_mod13q1")

# create a list to store the results
results <- list()

# adjust the multicores parameters to suit your machine

## SVM model
conf_svm.tb <- sits_kfold_validate(
    samples_matogrosso_mod13q1,
    folds = 5,
    multicores = 2,
    ml_method = sits_svm(kernel = "radial", cost = 10))

# Give a name to the SVM model
conf_svm.tb$name <- "svm_10"

# store the result
results[[length(results) + 1]] <- conf_svm.tb
```

```
conf_rfor.tb <- sits_kfold_validate(
    samples_matogrosso_mod13q1,
    folds = 5,
    multicores = 1,
    ml_method = sits_rfor(num_trees = 500))

# Give a name to the model
conf_rfor.tb$name <- "rfor_500"

# store the results in a list
results[[length(results) + 1]] <- conf_rfor.tb

# Save to an XLS file
sits_to_xlsx(results, file = tempfile(fileext = ".xlsx"))
```

```
#> Saved Excel file /tmp/Rtmpd7Tr58/file12240817fd6.xlsx
```

Aghabozorgi, Saeed, Ali Seyed Shirkhorshidi, and Teh Ying Wah. 2015. "Time-Series Clustering: A Decade Review." *Information Systems* 53: 16–38. https://doi.org/10.1016/j.is.2015.04.007.

Appel, Marius, and Edzer Pebesma. 2019. "On-Demand Processing of Data Cubes from Satellite Image Collections with the Gdalcubes Library." *Data* 4 (3): 1–16. https://doi.org/10.3390/data4030092.

Arévalo, Paulo, Eric L. Bullock, Curtis E. Woodcock, and Pontus Olofsson. 2020. "A Suite of Tools for Continuous Land Change Monitoring in Google Earth Engine." *Frontiers in Climate* 2. https://doi.org/10.3389/fclim.2020.576740.

Arvor, D., M. Meirelles, V. Dubreuil, and Y. E. Shimabukuro. 2012. "Analyzing the Agricultural Transition in Mato Grosso, Brazil, Using Satellite-Derived Indices." *Applied Geography* 32 (2): 702–13.

Atkinson, Peter M, C Jeganathan, Jadu Dash, and Clement Atzberger. 2012. "Inter-Comparison of Four Models for Smoothing Satellite Sensor Time-Series Data to Estimate Vegetation Phenology." *Remote Sensing of Environment* 123: 400–417.

Atzberger, Clement, and Paul HC Eilers. 2011. "Evaluating the Effectiveness of Smoothing Algorithms in the Absence of Ground Reference Measurements." *International Journal of Remote Sensing* 32 (13): 3689–3709.

Belgiu, Mariana, and Lucian Dragut. 2016. "Random Forest in Remote Sensing: A Review of Applications and Future Directions." *ISPRS Journal of Photogrammetry and Remote Sensing* 114: 24–31.

Bradley, Bethany A, Robert W Jacob, John F Hermance, and John F Mustard. 2007. "A Curve Fitting Procedure to Derive Inter-Annual Phenologies from

Time Series of Noisy Satellite NDVI Data." *Remote Sensing of Environment* 106 (2): 137–45.

Brown, J. Christopher, Jude H. Kastens, Alexandre Camargo Coutinho, Daniel de Castro Victoria, and Christopher R. Bishop. 2013. "Classifying Multiyear Agricultural Land Use Data from Mato Grosso Using Time-Series MODIS Vegetation Index Data." *Remote Sensing of Environment* 130: 39–50.

Camara, Gilberto, Gilberto Queiroz, Lubia Vinhas, Karine Ferreira, Ricardo Cartaxo, Rolf Simoes, Eduardo Llapa, Luiz Assis, and Alber Sanchez. 2017. "The E-Sensing Architecture for Big Earth Observation Data Analysis." In *Big Data from Space*, 48–51. Toulouse: ESA.

Chang, Chih-Chung, and Chih-Jen Lin. 2011. "LIBSVM: A Library for Support Vector Machines." *ACM Transactions on Intelligent Systems and Technology (TIST)* 2 (3): 27.

Chen, Jin, Per. Jönsson, Masayuki Tamura, Zhihui Gu, Bunkei Matsushita, and Lars Eklundh. 2004. "A Simple Method for Reconstructing a High-Quality NDVI Time-Series Data Set Based on the Savitzky–Golay Filter." *Remote Sensing of Environment* 91 (3): 332–44. https://doi.org/10.1016/j.rse.2004.03.014.

Chollet, François, and J. J Allaire. 2018. *Deep Learning with R*. Manning Publications Co. https://www.safaribooksonline.com/library/view//9781617295546/?ar.

Cortes, Corinna, and Vladimir Vapnik. 1995. "Support-Vector Networks." *Machine Learning* 20 (3): 273–97.

Estel, Stephan, Tobias Kuemmerle, Camilo Alcantara, Christian Levers, Alexander Prishchepov, and Patrick Hostert. 2015. "Mapping Farmland Abandonment and Recultivation Across Europe Using MODIS NDVI Time Series." *Remote Sensing of Environment* 163: 312–25.

Frenay, B., and M. Verleysen. 2014. "Classification in the Presence of Label Noise: A Survey." *IEEE Transactions on Neural Networks and Learning Systems* 25 (5): 845–69. https://doi.org/10.1109/TNNLS.2013.2292894.

Galford, Gillian L, John F Mustard, Jerry Melillo, Aline Gendrin, Carlos C Cerri, and Carlos E Cerri. 2008. "Wavelet Analysis of MODIS Time Series to Detect Expansion and Intensification of Row-Crop Agriculture in Brazil." *Remote Sensing of Environment* 112 (2): 576–87.

Gomez, Cristina, Joanne C. White, and Michael A. Wulder. 2016. "Optical Remotely Sensed Time Series Data for Land Cover Classification: A Review." *{ISPRS} Journal of Photogrammetry and Remote Sensing* 116: 55–72. https://doi.org/10.1016/j.isprsjprs.2016.03.008.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.

Gorelick, Noel, Matt Hancher, Mike Dixon, Simon Ilyushchenko, David Thau, and Rebecca Moore. 2017. "Google Earth Engine: Planetary-Scale Geospatial Analysis for Everyone." *Remote Sensing of Environment* 202: 18–27.

Griffiths, Patrick, Claas Nendel, Jürgen Pickert, and Patrick Hostert. 2019. "Towards National-Scale Characterization of Grassland Use Intensity from Integrated Sentinel-2 and Landsat Time Series." *Remote Sensing of Environment*, April, 111124. https://doi.org/10.1016/j.rse.2019.03.017.

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

Hennig, Christian. 2015. "Clustering Strategy and Method Selection." In *Handbook of Cluster Analysis*, edited by Christian Hennig, Marina Meila, Fionn Murtagh, and Roberto Rocci. CRC Press.

Hird, Jennifer N, and Gregory J McDermid. 2009. "Noise Reduction of NDVI Time Series: An Empirical Comparison of Selected Techniques." *Remote Sensing of Environment* 113 (1): 248–58.

Huang, Xin, Qikai Lu, Liangpei Zhang, and Antonio Plaza. 2014. "New Postprocessing Methods for Remote Sensing Image Classification: A Systematic Study." *IEEE Transactions on Geoscience and Remote Sensing* 52 (11): 7140–59.

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

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

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

Jonsson, Per, and Lars Eklundh. 2004. "TIMESAT: A Program for Analyzing Time-Series of Satellite Sensor Data." *Computers and Geosciences* 30 (8): 833–45.

Karim, Fazle, Somshubra Majumdar, Houshang Darabi, and Shun Chen. 2018. "LSTM Fully Convolutional Networks for Time Series Classification." *IEEE Access* 6: 1662–9.

Karim, Fazle, Somshubra Majumdar, Houshang Darabi, and Samuel Harford. 2019. "Multivariate LSTM-FCNS for Time Series Classification." *Neural Networks* 116: 237–45.

Kastens, J., J. Brown, A. Coutinho, C. Bishop, and J. Esquerdo. 2017. "Soy Moratorium Impacts on Soybean and Deforestation Dynamics in Mato Grosso, Brazil." *PLOS ONE* 12 (4): e0176168.

Kennedy, Robert E., Zhiqiang Yang, and Warren B. Cohen. 2010. "Detecting Trends in Forest Disturbance and Recovery Using Yearly Landsat Time Series: 1. LandTrendr — Temporal Segmentation Algorithms." *Remote Sensing of Environment* 114 (12): 2897–2910. https://doi.org/10.1016/j.rse.2010.07.008.

Keogh, Eamonn, Jessica Lin, and Wagner Truppel. 2003. "Clustering of Time Series Subsequences Is Meaningless: Implications for Previous and Future Research." In *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, 115–22.

Kohonen, T. 1990. "The Self-Organizing Map." *Proceedings of the IEEE* 78 (9): 1464–80. https://doi.org/10.1109/5.58325.

Kohonen, Teuvo. 2013. "Essentials of the Self-Organizing Map." *Neural Networks*, Twenty-fifth Anniversay Commemorative Issue, 37 (January): 52–65. https://doi.org/10.1016/j.neunet.2012.09.018.

Lambin, E. F., and M. Linderman. 2006. "Time Series of Remote Sensing Data for Land Change Science." *IEEE Transactions on Geoscience and Remote Sensing* 44 (7): 1926–8.

Lambin, Eric F, Helmut J Geist, and Erika Lepers. 2003. "Dynamics of Land-Use and Land-Cover Change in Tropical Regions." *Annual Review of Environment and Resources* 28 (1): 205–41.

Lewis, Adam, Simon Oliver, Leo Lymburner, Ben Evans, Lesley Wyborn, Norman Mueller, Gregory Raevksi, et al. 2017. "The Australian Geoscience Data Cube — Foundations and Lessons Learned." *Remote Sensing of Environment* 202 (December): 276–92. https://doi.org/10.1016/j.rse.2017.03.015.

Madden, Hannibal H. 1978. "Comments on the Savitzky-Golay Convolution Method for Least-Squares-Fit Smoothing and Differentiation of Digital Data." *Analytical Chemistry* 50 (9): 1383–6.

Maus, Victor, Gilberto Camara, Ricardo Cartaxo, Alber Sanchez, Fernando M Ramos, and Gilberto R Queiroz. 2016. "A Time-Weighted Dynamic Time Warping Method for Land-Use and Land-Cover Mapping." *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 9 (8): 3729–39. https://doi.org/10.1109/JSTARS.2016.2517118.

Maus, Victor, Gilberto Câmara, Marius Appel, and Edzer Pebesma. 2019. "dtwSat: Time-Weighted Dynamic Time Warping for Satellite Image Time Series Analysis in R." *Journal of Statistical Software* 88 (5): 1–31. https://doi.org/10.18637/jss.v088.i05.

Maxwell, Aaron E., Timothy A. Warner, and Fang Fang. 2018. "Implementation of Machine-Learning Classification in Remote Sensing: An Applied Review." *International Journal of Remote Sensing* 39 (9): 2784–2817.

Mountrakis, G., J. Im, and C. Ogole. 2011. "Support Vector Machines in Remote Sensing: A Review." *ISPRS Journal of Photogrammetry and Remote Sensing* 66 (3): 247–59.

Parente, Leandro, Evandro Taquary, Ana Paula Silva, Carlos Souza, and Laerte Ferreira. 2019. "Next Generation Mapping: Combining Deep Learning, Cloud

Computing, and Big Remote Sensing Data." *Remote Sensing* 11 (23, 23): 2881. https://doi.org/10.3390/rs11232881.

Pasquarella, Valerie J., Christopher E. Holden, Les Kaufman, and Curtis E. Woodcock. 2016. "From Imagery to Ecology: Leveraging Time Series of All Available LANDSAT Observations to Map and Monitor Ecosystem State and Dynamics." *Remote Sensing in Ecology and Conservation* 2 (3): 152–70. https://doi.org/10.1002/rse2.24.

Pelletier, Charlotte, Silvia Valero, Jordi Inglada, Nicolas Champion, and Gerard Dedieu. 2016. "Assessing the Robustness of Random Forests to Map Land Cover with High Resolution Satellite Image Time Series over Large Areas." *Remote Sensing of Environment* 187: 156–68. https://doi.org/10.1016/j.rse.2016.10.010.

Pelletier, Charlotte, Geoffrey I. Webb, and Francois Petitjean. 2019. "Temporal Convolutional Neural Network for the Classification of Satellite Image Time Series." *Remote Sensing* 11 (5).

Petitjean, F., J. Inglada, and P. Gancarski. 2012. "Satellite Image Time Series Analysis Under Time Warping." *IEEE Transactions on Geoscience and Remote Sensing* 50 (8): 3081–95. https://doi.org/10.1109/TGRS.2011.2179050.

Petitjean, François, Florent Masseglia, Pierre Gançarski, and Germain Forestier. 2011. "Discovering Significant Evolution Patterns from Satellite Image Time Series." *International Journal of Neural Systems* 21 (06): 475–89. https://doi.org/10.1142/S0129065711003024.

Picoli, Michelle, Gilberto Camara, Ieda Sanches, Rolf Simoes, Alexandre Carvalho, Adeline Maciel, Alexandre Coutinho, et al. 2018. "Big Earth Observation Time Series Analysis for Monitoring Brazilian Agriculture." *ISPRS Journal of Photogrammetry and Remote Sensing* 145: 328–39. https://doi.org/10.1016/j.isprsjprs.2018.08.007.

Potapov, Peter, Matthew C. Hansen, Indrani Kommareddy, Anil Kommareddy, Svetlana Turubanova, Amy Pickens, Bernard Adusei, Alexandra Tyukavina, and Qing Ying. 2020. "Landsat Analysis Ready Data for Global Land Cover and Land Cover Change Mapping." *Remote Sensing* 12 (3, 3): 426. https://doi.org/10.3390/rs12030426.

Ruder, Sebastian. 2016. "An Overview of Gradient Descent Optimization Algorithms." *CoRR* abs/1609.04747. http://arxiv.org/abs/1609.04747.

Russwurm, Marc, and Marco Korner. 2018. "Multi-Temporal Land Cover Classification with Sequential Recurrent Encoders." *ISPRS International Journal of Geo-Information* 7 (4): 129.

Rußwurm, Marc, and Marco Korner. 2017. "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*, 11–19.

Sakamoto, Toshihiro, Masayuki Yokozawa, Hitoshi Toritani, Michio Shibayama, Naoki Ishitsuka, and Hiroyuki Ohno. 2005. "A Crop Phenology Detection Method Using Time-Series MODIS Data." *Remote Sensing of Environment* 96 (3-4): 366–74.

Schindler, Konrad. 2012. "An Overview and Comparison of Smooth Labeling Methods for Land-Cover Classification." *IEEE Transactions on Geoscience and Remote Sensing* 50 (11): 4534–45.

Shao, Yang, Ross S. Lunetta, Brandon Wheeler, John S. Iiames, and James B. Campbell. 2016. "An Evaluation of Time-Series Smoothing Algorithms for Land-Cover Classifications Using MODIS-NDVI Multi-Temporal Data." *Remote Sensing of Environment* 174: 258–65.

Simoes, Rolf, Michelle C. A. Picoli, Gilberto Camara, Adeline Maciel, Lorena Santos, Pedro R. Andrade, Alber Sánchez, Karine Ferreira, and Alexandre Carvalho. 2020. "Land Use and Cover Maps for Mato Grosso State in Brazil from 2001 to 2017." *Scientific Data* 7 (1): 34. https://doi.org/10.1038/s41597-020-0371-4.

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." *The Journal of Machine Learning Research* 15 (1): 1929–58.

Thanh Noi, Phan, and Martin Kappas. 2018. "Comparison of Random Forest, K-Nearest Neighbor, and Support Vector Machine Classifiers for Land Cover Classification Using Sentinel-2 Imagery." *Sensors* 18 (1, 1): 18. https://doi.org/10.3390/s18010018.

Verbesselt, Jan, Rob Hyndman, Glenn Newnham, and Darius Culvenor. 2010. "Detecting Trend and Seasonal Changes in Satellite Image Time Series." *Remote Sensing of Environment* 114 (1): 106–15. https://doi.org/10.1016/j.rse.2009.08.014.

Wang, Zhiguang, Weizhong Yan, and Tim Oates. 2017. "Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline." In *2017 International Joint Conference on Neural Networks (IJCNN)*.

Ward, Joe H. 1963. "Hierarchical Grouping to Optimize an Objective Function." *Journal of the American Statistical Association* 58 (301): 236–44.

Whittaker, Edmund T. 1922. "On a New Method of Graduation." *Proceedings of the Edinburgh Mathematical Society* 41: 63–75.

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

Woodcock, Curtis E., Thomas R. Loveland, Martin Herold, and Marvin E. Bauer. 2020. "Transitioning from Change Detection to Monitoring with Remote Sensing: A Paradigm Shift." *Remote Sensing of Environment* 238 (March): 111558. https://doi.org/10.1016/j.rse.2019.111558.

Wright, Jonathon S., Rong Fu, John R. Worden, Sudip Chakraborty, Nicholas E. Clinton, Camille Risi, Ying Sun, and Lei Yin. 2017. "Rainforest-Initiated Wet Season Onset over the Southern Amazon." *Proceedings of the National Academy of Sciences*, July. https://doi.org/10.1073/pnas.1621516114.

Zhou, Jie, Li Jia, Massimo Menenti, and Ben Gorte. 2016. "On the Performance of Remote Sensing Time Series Reconstruction Methods: A Spatial Comparison." *Remote Sensing of Environment* 187: 367–84.

Zhu, Zhe, Junxue Zhang, Zhiqiang Yang, Amal H Aljaddani, Warren B Cohen, Shi Qiu, and Congliang Zhou. 2020. "Continuous Monitoring of Land Disturbance Based on Landsat Time Series." *Remote Sensing of Environment* 238: 111116.