Classification of Images in Data Cubes using Satellite Image Time Series

Rolf Simoes *National Institute for Space Research (INPE), Brazil*
Gilberto Camara *National Institute for Space Research (INPE), Brazil*
Pedro R. Andrade *National Institute for Space Research (INPE), Brazil*
Alexandre Carvalho *Institute for Applied Economics Research (IPEA), Brazil*
Gilberto Queiroz *National Institute for Space Research (INPE), Brazil*

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

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.

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 brick with 23 instances for one year
# Select the band "ndvi", "evi" from bricks  available in the "sits" package
ndvi_file <- system.file("extdata/raster/mod13q1/sinop-crop-ndvi.tif",
                         package = "sits")

# Obtain the associated timeline
timeline <- data("timeline_2000_2017")

# create a raster metadata file based on the information about the files
raster_cube <- sits_cube(type = "BRICK",
                         name = "Sinop",
                         satellite = "TERRA",
                         sensor = "MODIS",
                         timeline = timeline,
                         bands = c("ndvi"),
                         files = c(ndvi_file))
```

```
## Warning: All formats failed to parse. No formats found.
```

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

```
## # A tibble: 1 x 4
##   type  URL              satellite sensor
##   <chr> <chr>            <chr>     <chr>
## 1 BRICK http://127.0.0.1 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 -6054361. -6051117. -1282895. -1280347.
```

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`.

Classification using machine learning

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

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

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

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

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 with 1D convolutional neural networks (`sits_CNN`),
- DL combining 1D CNN and multi-layer perceptron networks (`sits_tempCNN`)
- DL using 1D version of ResNet (`sits_ResNet`).
- DL using a combination of long-short term memory (LSTM) and 1D CNN (`sits_LSTM_FCN`)

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

Cube classification

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

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

Steps for cube classification

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

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

Adjustments for improved performance

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

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

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

```
# install the inSitu package
#
devtools::install_github("e-sensing/inSitu")
```

```
## Skipping install of 'inSitu' from a github remote, the SHA1 (c279dcd4) has not changed since
##   Use `force = TRUE` to force installation
```

```
# load the inSitu package

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

# build the classification model
rfor_model <- sits_train(mato_grosso_2bands, ml_method = sits_rfor(num_trees = 2000))

# select the bands "ndvi", "evi" from the "inSitu" package
evi_file  <- system.file("extdata/Sinop", "Sinop_evi_2014.tif", package = "inSitu")
ndvi_file <- system.file("extdata/Sinop", "Sinop_ndvi_2014.tif", package = "inSitu")

files <- c(ndvi_file, evi_file)
# define the timeline
time_file <- system.file("extdata/Sinop", "timeline_2014.txt", package = "inSitu")
timeline_2013_2014 <- scan(time_file, character())
```

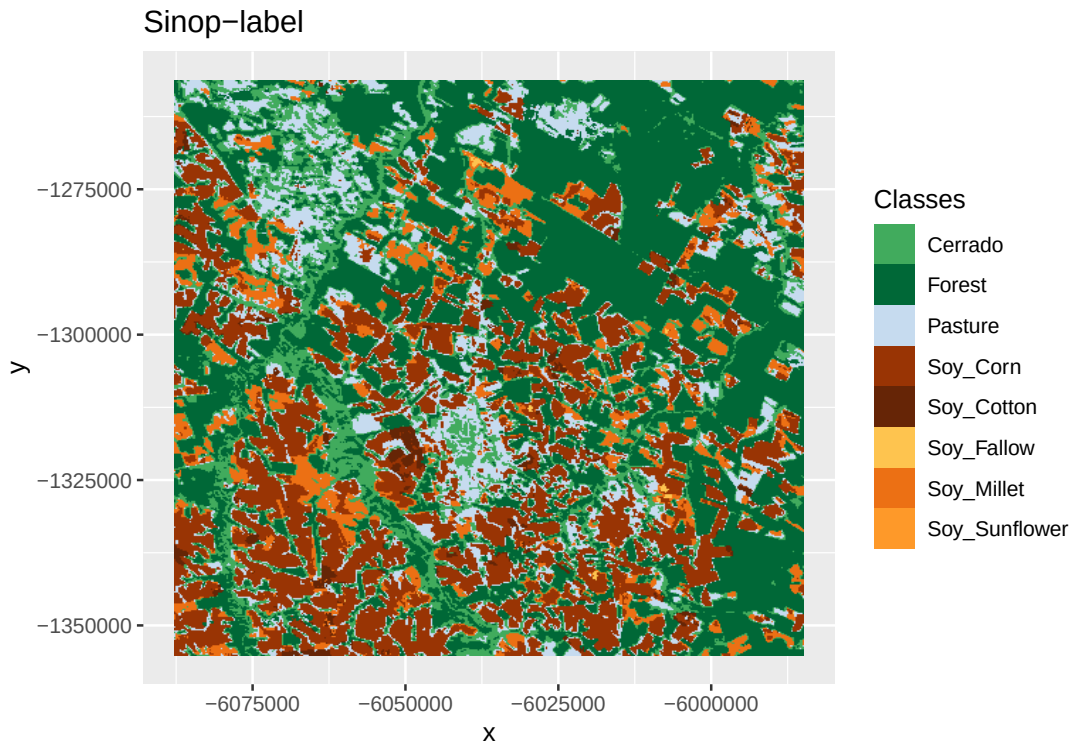
```
# create a raster metadata file based on the information about the files
sinop <- sits_cube(type = "BRICK",
                   satellite = "TERRA",
                   sensor    = "MODIS",
                   name = "Sinop",
                   timeline = timeline_2013_2014,
                   bands = c("ndvi", "evi"),
                   files = files)

# classify the raster image
sinop_probs <- sits_classify(sinop, ml_model = rfor_model, memsize = 2, multicores = 1)
```

```
## Starting classification at 2020-08-13 14:20:41
```

```
## Classification finished at 2020-08-13 14:21:12. Total elapsed time: 0.5 minute(s).
```

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



Smoothing of raster data after classification

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

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

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. The following example takes the previously produced classification output and applies a Bayesian smoothing.

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

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

Final remarks

Current approaches to image time series analysis still use limited number of attributes. A common approach is deriving a small set of phenological parameters from vegetation indices, like beginning, peak, and length of growing season [Brown et al., 2013], [Kastens et al., 2017], [Estel et al., 2015], [Pelletier et al., 2016]. These phenological parameters are

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

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

References

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

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

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

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

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

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

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

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

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

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

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

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