**Sentinel-2 Agriculture**

Design Definition File

Algorithm Theoretical Basis Document for L4 dynamic crop mask product



|  |  |
| --- | --- |
| Milestone | Milestone 2 |
| Version | 1.4 |
| Authors | CESBIO - Silvia Valero, David Morin  UCL - Guadalupe Sepulcre, Nicolas Matton, Sophie Bontemps, Nicolas Bellemans |

***This page is intentionally blank***

***Table of recorded changes***

**Issue record sheet**

|  |  |  |
| --- | --- | --- |
| Issue/Rev. | Date | Reason |
| 1.0 | 19/04/2015 | 1st version for ESA |
| 1.1 | 01/07/2015 | 2nd version to ESA |
| 1.2 | 13/01/2016 | Modifications applied in the gap filling and smoothing step of the crop mask without in situ data imply an adjustment of the ATBD. |
| 1.3 | 08/04/2016 | Addition of quality flags |
| 1.4 | 14/06/2017 | Modifications to specify how including the S2 spectral bands |

**Detailed record sheet**

*From version 1.1 to 1.2*

|  |  |  |
| --- | --- | --- |
| RID | Section | Change |
| / | 1. | Modification of the Flowchart (Figure 1-1.) and his text description. Both crop mask (with and without in situ data) have now a different gap filling step. |
| / | 2.1 | In the previous release, the gap filling was a common step. Previous section 2. is now incorporated in the former section 3. as section 2.1. |
| / | 3.1. | The smoothing of the data is now joined to the gap filling step using the Whitaker filter. Processing details added. |

*From version 1.2 to 1.3*

|  |  |  |
| --- | --- | --- |
| RID | Section | Change |
| / | 5. | Quality flags added |

*From version 1.3 to 1.4*

|  |  |  |
| --- | --- | --- |
| RID | Section | Change |
| / | 2.3.4. | Specification of how incorporating the S2 red-edge features in the in-situ mode |
| / |  | Specification of how incorporating the S2 red-edge features in the non in-situ mode |

***Table of contents***

[1 Logical model 9](#_Toc485395197)

[2 Mode based on in-situ data 11](#_Toc485395198)

[2.1 Preliminary step : gap filling 11](#_Toc485395199)

[2.2 Pre-processing: spectral indices computation 13](#_Toc485395200)

[2.3 Feature extraction step 13](#_Toc485395201)

[2.3.1 Temporal features 14](#_Toc485395202)

[2.3.2 Statistic features 18](#_Toc485395203)

[2.3.3 Feature image concatenation 19](#_Toc485395204)

[2.3.4 Incorporating S2 red-edge features 20](#_Toc485395205)

[2.4 Classification step 20](#_Toc485395206)

[2.4.1 Constructing the classification model 20](#_Toc485395207)

[2.4.2 Classification of the image times series : Random Forest 26](#_Toc485395208)

[2.5 Processing chain working in the iterative operational model 27](#_Toc485395209)

[2.5.1 The first year of Sentinel-2 images 28](#_Toc485395210)

[2.5.2 The rest of the years 28](#_Toc485395211)

[3 Mode without in-situ data 30](#_Toc485395212)

[3.1 Preliminary step: data smoothing and gap filling 30](#_Toc485395213)

[3.2 Features extraction 32](#_Toc485395214)

[3.2.1 Temporal features definition 32](#_Toc485395215)

[3.2.2 Spectral features extraction 34](#_Toc485395216)

[3.3 Reference map preparation 37](#_Toc485395217)

[3.4 Trimming 38](#_Toc485395218)

[3.5 Random forest classifier 42](#_Toc485395219)

[3.5.1 RF model estimation 42](#_Toc485395220)

[3.5.2 Classification 43](#_Toc485395221)

[4 Common final steps 44](#_Toc485395222)

[4.1 Classifier Evaluation: Computing evaluation quality measures 44](#_Toc485395223)

[4.2 Crop mask post-filtering 45](#_Toc485395224)

[4.2.1 Principal Component Analysis 46](#_Toc485395225)

[4.2.2 Mean-Shift segmentation algorithm 46](#_Toc485395226)

[4.2.3 Majority vote 47](#_Toc485395227)

[4.2.4 Operational context 47](#_Toc485395228)

[5 Processing chain model working for a large area covering several satellite footprints 49](#_Toc485395229)

[5.1 Large region classification 49](#_Toc485395230)

[5.2 Stratification 49](#_Toc485395231)

[6 Quality flags 51](#_Toc485395232)

[7 Appendix - Additional information on the codes of the « in-situ » processing chain 52](#_Toc485395233)

***List of figures***

[Figure 1‑1. Flowchart of the processing chain developed to generate the cropland mask 9](#_Toc485395234)

[Figure 2‑1. Flowchart describing the processing chain when field data exists 11](#_Toc485395235)

[Figure 2‑2. Gap filling workflow 11](#_Toc485395236)

[Figure 2‑3. Pre-processing workflow, computing NDVI, NDWI and Brightness indices 13](#_Toc485395237)

[Figure 2‑4. Temporal features generation workflow 14](#_Toc485395238)

[Figure 2‑5. Maximum NDVI values identification from a full NDVI time series 15](#_Toc485395239)

[Figure 2‑6. Identification of the largest increasing period in a full NDVI time series 16](#_Toc485395240)

[Figure 2‑7. Identification of bare soil transitions in a full NDVI time series 18](#_Toc485395241)

[Figure 2‑8. Statistic features generation workflow 18](#_Toc485395242)

[Figure 2‑9. Workflow of the temporal and statistic features concatenation 19](#_Toc485395243)

[Figure 2‑10. Workflow for the incorporation of the S2 red-edge features 20](#_Toc485395244)

[Figure 2‑11. Workflow of the classification model construction 21](#_Toc485395245)

[Figure 2‑12. Workflow of the in-situ dataset splitting sub-task 21](#_Toc485395246)

[Figure 2‑13. Illustration of the in-situ dataset split into two subsets for training and validation using a ration equal to 0.5 22](#_Toc485395247)

[Figure 2‑14. Workflow for reducing the size of the training sample 23](#_Toc485395248)

[Figure 2‑15. Workflow of the classifier training 25](#_Toc485395249)

[Figure 2‑16. Workflow of the RF classification 27](#_Toc485395250)

[Figure 2‑17. “In-situ” classification chain running at instants T and T+1, when we are during the first S2 year 28](#_Toc485395251)

[Figure 2‑18. “In-situ” classification chain running in Year X-1, when we are not during the first S2 year 29](#_Toc485395252)

[Figure 2‑19. “In-situ” classification chain running in Year X, when we are not during the first S2 year 29](#_Toc485395253)

[Figure 3‑1. Workflow of the smoothing preliminary step 30](#_Toc485395254)

[Figure 3‑2. Workflow of the temporal and spectral features extraction process 32](#_Toc485395255)

[Figure 3‑3. Workflow of the reference map preparation 37](#_Toc485395256)

[Figure 3‑4. Workflow of the trimming process 39](#_Toc485395257)

[Figure 4‑1. Workflow of the classification assessment 44](#_Toc485395258)

[Figure 4‑2. Workflow of the post-filtering step 45](#_Toc485395259)

[Figure 4‑3. Post-filtering workflow, at instant T 48](#_Toc485395260)

[Figure 4‑4. Post-filtering workflow, at instant T+1 48](#_Toc485395261)

***List of tables***

[Table 2‑1. Variables for the linear interpolation and gap filling algorithm 12](#_Toc485395262)

[Table 2‑2. Variables for the spectral indices computation 13](#_Toc485395263)

[Table 2‑3. Variables for the spectral indices computation 14](#_Toc485395264)

[Table 2‑4. Global features description 14](#_Toc485395265)

[Table 2‑5. Description of features detecting largest local NDVI transitions 15](#_Toc485395266)

[Table 2‑6. Maximum NDVI features description 15](#_Toc485395267)

[Table 2‑7. Greenness period features description 16](#_Toc485395268)

[Table 2‑8. Senescence onset features description 17](#_Toc485395269)

[Table 2‑9. Bare soil transition features description 18](#_Toc485395270)

[Table 2‑10. Variables for the spectral indices computation 19](#_Toc485395271)

[Table 2‑11. Statistic features description 19](#_Toc485395272)

[Table 2‑12. Variables for the spectral features concatenation 19](#_Toc485395273)

[Table 2‑13. Variables for the splitting the field dataset for training and validation 22](#_Toc485395274)

[Table 2‑14. Variables for the splitting the field dataset for training and validation 24](#_Toc485395275)

[Table 2‑15. Variables for training the Radom Forest 25](#_Toc485395276)

[Table 2‑16. Variables for the RF classification algorithm 27](#_Toc485395277)

[Table 3‑1. Variables for the data smoothing 30](#_Toc485395278)

[Table 3‑2. Variables for the temporal features identification 33](#_Toc485395279)

[Table 3‑3. Variables for the spectral features extraction 35](#_Toc485395280)

[Table 3‑4. Variables for the erosion of the reference 37](#_Toc485395281)

[Table 3‑5. Variables for the trimming 39](#_Toc485395282)

[Table 3‑6. Variables for the RF model estimation in the « no in-situ » mode 42](#_Toc485395283)

[Table 3‑7. Variables for the RF classification for the « no in-situ » mode 43](#_Toc485395284)

[Table 4‑1. Variables for the map validation algorithm 45](#_Toc485395285)

[Table 4‑2. Variables for the Principal Component Analysis 46](#_Toc485395286)

[Table 4‑3. Variables for the Mean-Shift segmentation 46](#_Toc485395287)

[Table 4‑4. Variables for the majority voting filtering 47](#_Toc485395288)

***List of algorithms***

[Algorithm 2‑1. Linear interpolation and gap filling 12](#_Toc485395289)

[Algorithm 2‑2. Spectral indices computation 13](#_Toc485395290)

[Algorithm 2‑3. OTB application performing the feature images concatenation 20](#_Toc485395291)

[Algorithm 2‑4. OTB application performing the feature images concatenation 23](#_Toc485395292)

[Algorithm 2‑5. Random selection of training samples 24](#_Toc485395293)

[Algorithm 2‑6. OTB application computing the vector image statistics 26](#_Toc485395294)

[Algorithm 3‑1. Smoothing/Gap Filling based on the Whitaker function 31](#_Toc485395295)

[Algorithm 3‑2. Temporal features identification 33](#_Toc485395296)

[Algorithm 3‑3. Spectral features extraction 36](#_Toc485395297)

[Algorithm 3‑4. « SelectInList » program 36](#_Toc485395298)

[Algorithm 3‑5. Reference erosion 37](#_Toc485395299)

[Algorithm 3‑6. Trimming 40](#_Toc485395300)

[Algorithm 3‑7. MVG\_IT function 41](#_Toc485395301)

[Algorithm 4‑1. Computation of quality indices 45](#_Toc485395302)

[Algorithm 4‑2. Majority vote filtering 47](#_Toc485395303)

# Logical model

The classification process transform the L2A\_R imagery into a 1/0 mask, 1 indicating the presence of cropland and 0 the presence of non-cropland.

The processing chain is run by tiles.

The processing chain can run in two different modes, depending if the user counts with field data for training the classifier or not:

* If the user has field data to train the algorithm, an extensive number of features related to the NDVI, the NDWI profiles and the brightness will be extracted from the imagery time series for feeding a supervised algorithm based on a Random Forest (RF) classifier;
* If there is no field data, training samples will be selected from an existing reference map (local/regional/global land cover map, crop mask, etc.) through an iterative trimming. These samples will be used to feed the same supervised algorithm than in the first case (i.e. based on a RF classifier) but the this classifier will here be run based on reflectance values coming from determinate moments of the growing cycle.

The choice between the two modes (“field data” and “no field data”) is made once for all the Area of Interest.

The principal steps of the classification process are schematized in Figure 1‑1.

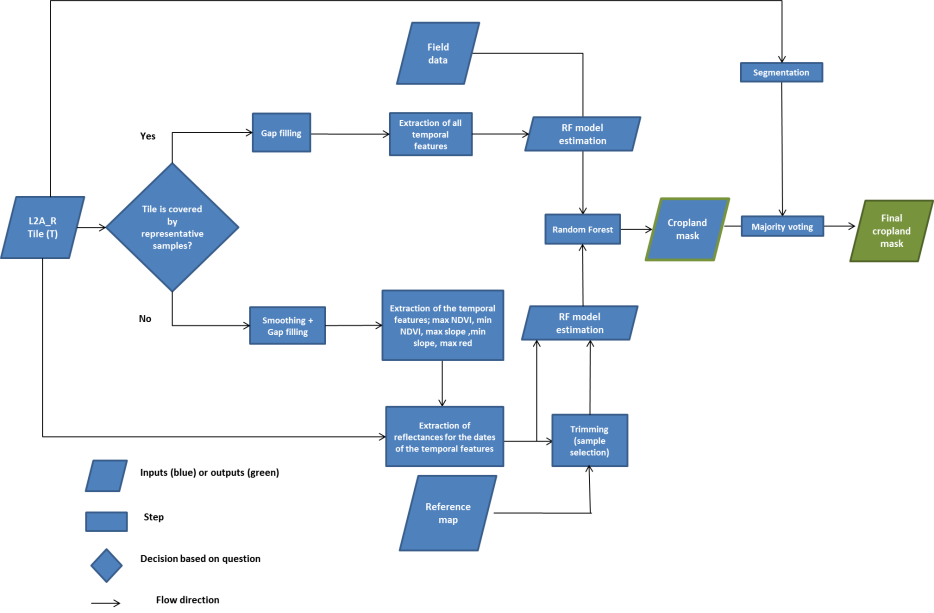
****

Figure 1‑1. Flowchart of the processing chain developed to generate the cropland mask

Two steps are common for the 2 modes:

* the supervised algorithm based on the RF classifier;
* the a posteriori filtering of the cropland mask.

The next sections of the document present in detail all the steps of the processing chain, distinguishing between the 2 modes and highlighting the common ones.

# Mode based on in-situ data

The processing chain to run when field data exist is illustrated in Figure 2‑1, which must be interpreted from left to right. This flowchart can be easily divided in three main tasks: the feature extraction, the classification and the post-filtering step. The post-filtering step is common with the “no field data module” and will thus be detailed later, in section 4. All other steps are detailed in this section.

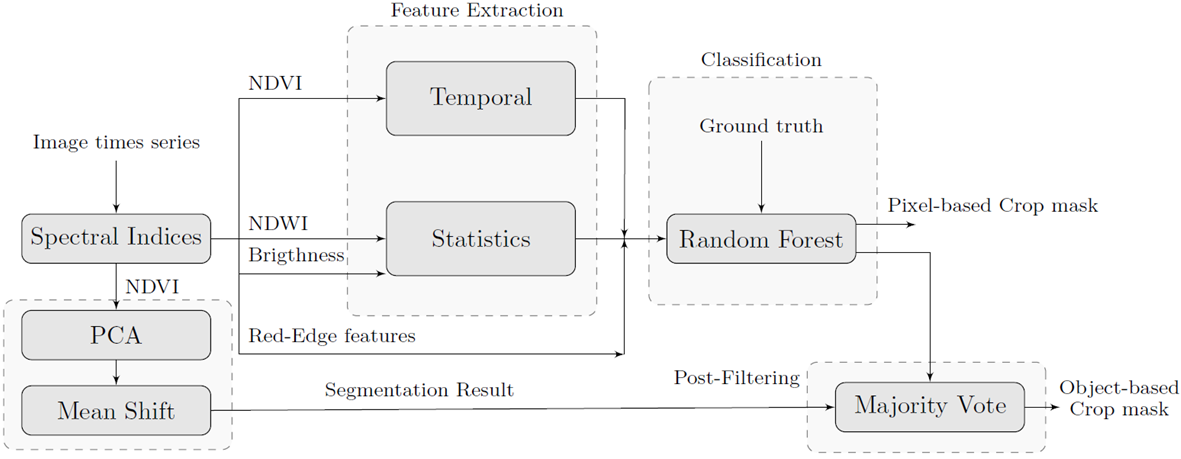


Figure 2‑1. Flowchart describing the processing chain when field data exists

The classification chain uses as input the reflectance image times series that don’t contain gaps (i.e. the output of the preliminary step detailed in section 2.1). As shown in Figure 2‑1, these time series will be used in the first purple block whose name is *Spectral Index*. This task corresponds to the computation of three spectral indices: the NDVI, the NDWI and the Brightness index (which is described in section 2.2). The output of the classification chain (i.e. before the common filtering step) is the pixel-based crop mask product will be the output of the classification chain.

## Preliminary step : gap filling

The goal of the linear interpolation is to produce a reflectance image time series which is (i) gap-filled with respect to missing data (which can be due to clouds, cloud shadows and saturated pixels) and (ii) temporally sampled on a regular grid. The workflow of the gap filling is illustrated in Figure 2‑2. If several sensors are available (for instance Sentinel-2 and LANDSAT-8) their respective reflectance time series are processed independently. These time series are afterwards concatenated before providing them to the classification step and being processed as single time series.

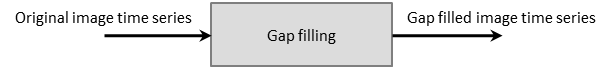


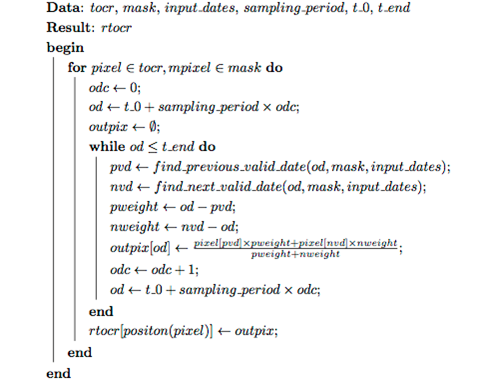
Figure 2‑2. Gap filling workflow

Algorithm 2‑1 describes the procedure, using the parameters given in Table 2‑1. In the case where the whole series is available, t\_0 and t\_end correspond to the first and last dates of the regular grid.

Table 2‑1. Variables for the linear interpolation and gap filling algorithm

|  |  |  |
| --- | --- | --- |
| **Input variables** | **Role** | **Default value** |
| tocr | Original L2A surface reflectance values | - |
| mask | Validity mask for each acquisition date | - |
| input\_dates | Dates of each image acquisition | - |
| sampling\_period | Temporal sampling rate | 5 |
| t\_0 | Starting sampling date | - |
| t\_end | Last date | - |
| radius | Radius of the temporal window | 15 |
| **Output variables** | **Role** | **Default value** |
| rtocr | Resampled L2A surface reflectance values | - |

Algorithm 2‑1. Linear interpolation and gap filling



The same algorithm can be implemented on-line, since the research of valid dates is performed inside a temporal window of size 2 × radius + 1. The function *find\_previous\_valid date* (resp. *find\_next\_valid\_date*) searches backward (resp. forward) the first date for which a valid value is available.

The minimum time series length required for the gap filling computation is 2 dates: one from the past and one from the future to fill the gap. It does not exist any condition about the minimum distance between the dates.

## Pre-processing: spectral indices computation

The pre-processing workflow is illustrated in Figure 2‑3, resulting in the three spectral indices: NDVI, NDWI and Brightness.

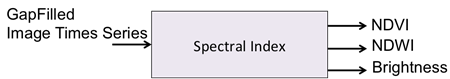


Figure 2‑3. Pre-processing workflow, computing NDVI, NDWI and Brightness indices

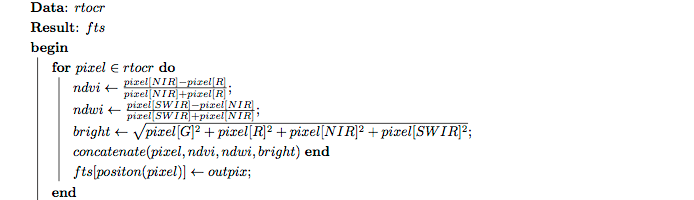
The spectral indices computation step produces the relevant features for the classification. The features are computed for each date of the resampled and gap-filled time series and concatenated together into a single multi-channel image file.

Algorithm 2‑2 describes the procedure, which can easily be implemented using the ORFEO Toolbox BandMath application. Variables of the algorithm are given in Table 2‑2. The NDVI computation will use the B8 band (not the B8a). The SWIR band will be resampled at 10 m.

Table 2‑2. Variables for the spectral indices computation

|  |  |  |
| --- | --- | --- |
| **Input variables** | **Role** | **Default value** |
| rtocr | Resampled L2A surface reflectance values | - |
| **Output variables** | **Role** | **Default value** |
| fts | Full gap-filled spectral indices times series | - |

Algorithm 2‑2. Spectral indices computation



## Feature extraction step

Three different indices have been generated: the NDVI, the NDWI and the Brigthness. These indices are used in order to compute different features: temporal features (see sub-section 2.3.1) and statistic features (see sub-section 2.3.2).

17 temporal features are derived from the NDVI. Concerning the other two spectral indices, 5 statistics features are computed for each of them. In total, 27 features are computed, which are concatenated as a vector image (see sub-section 2.3.3) to be used in the subsequent classification step (that will be detailed in section 2.3.4).

### Temporal features

The workflow developed to generate temporal features is illustrated in Figure 2‑4.



Figure 2‑4. Temporal features generation workflow

The features are computed using a C++ code compatible with OTB has been developed (see the section *NDVI feature computation task* presented in the Appendix of Section 0). Input and output variables are given in Table 2‑3. The different features are described below.

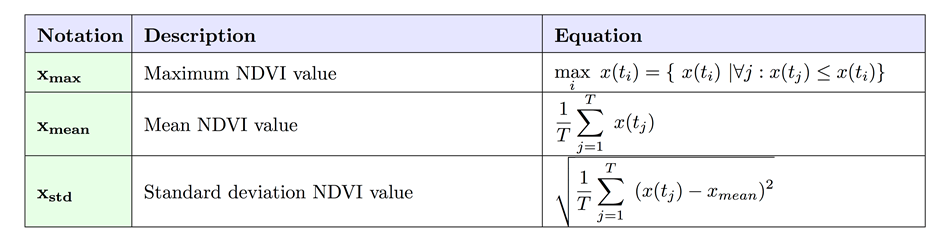
Table 2‑3. Variables for the spectral indices computation

|  |  |  |
| --- | --- | --- |
| **Input variables** | **Role** | **Default value** |
| fts - NDVI | The full gap-filled NDVI times series | - |
| **Output variables** | **Role** | **Default value** |
| temporal\_feat | A vector image made of 17 channels, one for each temporal feature | - |

* **Global features**

These features characterize the global trends on NDVI times series. The three features detailed Table 2‑4 are computed.

Table 2‑4. Global features description

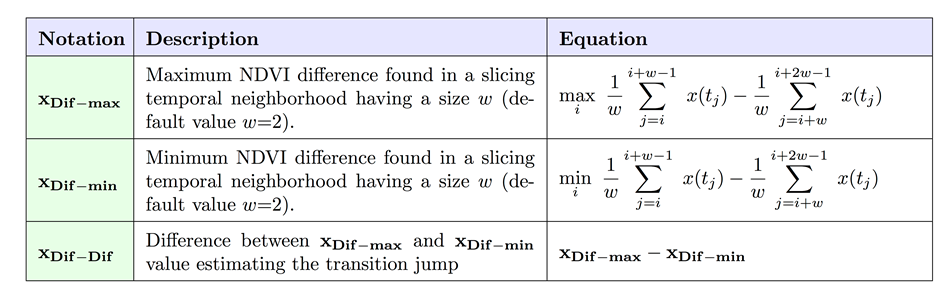


Parameters:

* x(ti) corresponds to the NDVI value at instant ti.
* T is the length of the NDVI times series
* mean is the mean NDVI value computed by using the T values
* **Features detecting largest local NDVI transitions**

These features describe the difference between consecutive windowed NDVI values. They are presented in Table 2‑5. The goal is to detect the largest transitions corresponding to the greenness and the senescence onset periods.

Table 2‑5. Description of features detecting largest local NDVI transitions



Parameters:

* x(ti) and x(tj) correspond to the NDVI values at instants ti and tj.
* w is the size of the slicing temporal window used to compute the differences
* **mbiFeatures associated to the maximum NDVI value:**

These features are associated to the area containing the maximum NDVI value, as identified in Figure 2‑5. The features are described in Table 2‑6.

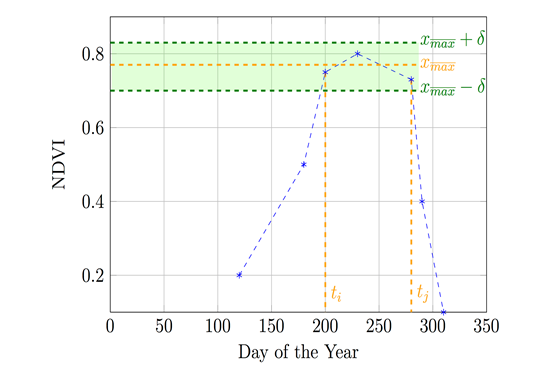
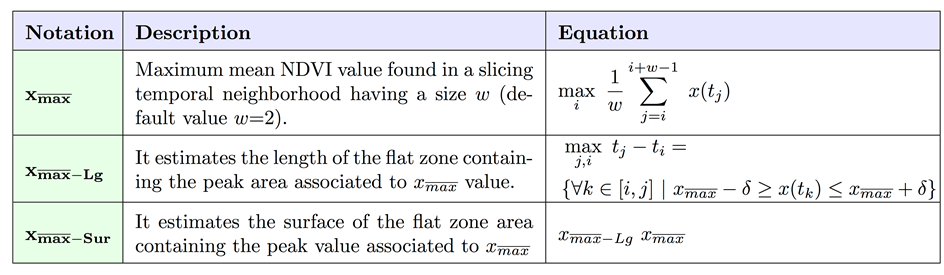


Figure 2‑5. Maximum NDVI values identification from a full NDVI time series

Table 2‑6. Maximum NDVI features description



Parameters:

* x(ti) and x(tj) correspond to the NDVI values at instants ti and tj.
* w is the size of the slicing temporal window used to compute the differences
* delta δ parameter is a user threshold, whose default value is equal to 0.05. In order to estimate the surface, a rectangular shape has been considered.
* **Features associated to the greenness onset**

These features are associated to the largest increasing period of the NDVI function. As it can be seen in Figure 2‑6, the greenness onset period is highlighted in orange. These temporal features are subject to the next constraint :

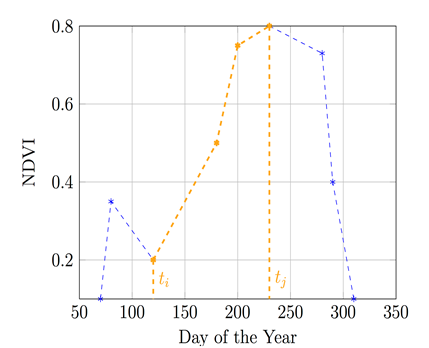
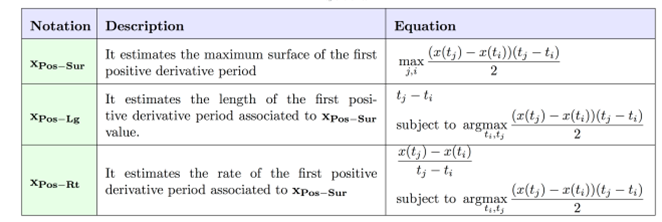


Figure 2‑6. Identification of the largest increasing period in a full NDVI time series

Table 2‑7 describes the features associated to the greenness period. In order to estimate the surface, a triangular shape has been considered.

Table 2‑7. Greenness period features description

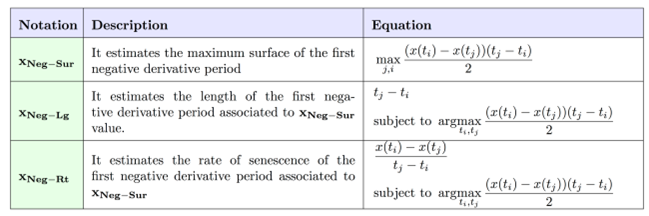


Parameters:

* x(ti) and x(tj) correspond to the NDVI values at instants ti and tj.
* **Features associated to the senescence onset:**

These features are associated to the largest decreasing period of the NDVI function. They are presented in Table 2‑8.

Table 2‑8. Senescence onset features description



Parameters:

* x(ti) and x(tj) correspond to the NDVI values at instants ti and tj.
* **Features characterizing bare soil transitions**

These features detect if there is a transition between the bare soil before the greenness onset or after the senescence onset. Bare soil transition is illustrated in Figure 2‑7 and corresponding features are given in Table 2‑9. The resulting value is 0 or 1.

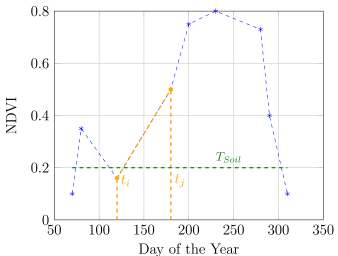
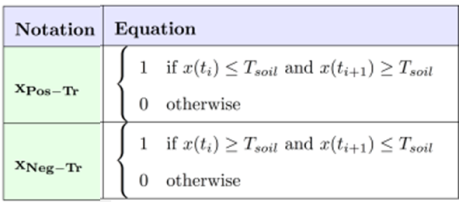


Figure 2‑7. Identification of bare soil transitions in a full NDVI time series

Table 2‑9. Bare soil transition features description



Parameters:

* x(ti) corresponds to the NDVI values at instant ti
* T\_soil is a threshold defining non cultivated areas

### Statistic features

The workflow developed to generate statistic features is illustrated in Figure 2‑8.



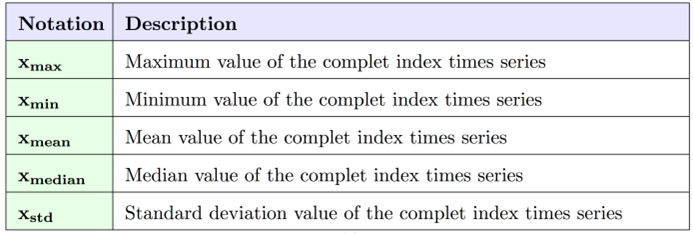
Figure 2‑8. Statistic features generation workflow

The features are computed using a C++ code compatible with OTB has been developed (see the section *Statistics feature computation task* from the Appendix presented in Section 0). Input and output variables are given in Table 2‑10. The statistic features are described in Table 2‑11.

Table 2‑10. Variables for the spectral indices computation

|  |  |  |
| --- | --- | --- |
| **Input variables** | **Role** | **Default value** |
| fts - NDWI | The full gap-filled NDWI times series | - |
| fts - Brightness | The full gap-filled Brightness times series | - |
| **Output variables** | **Role** | **Default value** |
| statistic\_feat | A vector image made of 10 channels: 5 statistic features computed on Brightness and 5 statistic features computed on NDWI | - |

Table 2‑11. Statistic features description



### Feature image concatenation

The complete set of temporal and statistic features are then concatenated as shown in Figure 2‑9. The resulting feature vector image is the output of the “feature extraction” step and will be used as a input for the next step : the classification.



Figure 2‑9. Workflow of the temporal and statistic features concatenation

Concatenation can be done using the OTB application presented in Algorithm 2‑3. Input and output variables are given in Table 2‑12.

Table 2‑12. Variables for the spectral features concatenation

|  |  |  |
| --- | --- | --- |
| **Input variables** | **Role** | **Default value** |
| temporal\_feat | A vector image made of 17 channels, one for each temporal feature | - |
| statistic\_feat | A vector image made of 10 channels: 5 statistic features computed on Brightness and 5 statistic features computed on NDWI | - |
| **Output variables** | **Role** | **Default value** |
| Feature Vector Image | A vector image containing the 27 temporal and statistic features | - |

Algorithm 2‑3. OTB application performing the feature images concatenation

otbcli\_ConcatenateImages -il Temporal\_Features.tif Statistic\_Features.tif

-out Feature\_Concatenate\_Images.tif

### Incorporating S2 red-edge features

In addition, the following 4 red-edge features computed from Sentinel-2 data must be added:

1. Red edge NDVI = (B08 -B06) / (B08+B06)
2. Sentinel-2 Red Edge Position = 705 + 35 \*( 0.5\*(B07+B04)-B05) /(B06-B05)
3. Plant Senescence Reflectance Indice PSRI = (B04-B02)/B05
4. Chlorophyll Red-Edge = B05/B08

These features must be computed after each acquisition. Therefore, each feature has its own times series profile composed of N values (one for each instant ti of time). The complete times profiles composed by all the possible values will be added to the feature vector image obtained after the concatenation described in section 2.3.3. The final input data vector used in the classification system is given in Figure 2‑10.



Figure 2‑10. Workflow for the incorporation of the S2 red-edge features

## Classification step

The classification step can be divided into two subtasks:

1. The construction of the classification model ;
2. The use of the classification model in the complete image times series and its evaluation.

### Constructing the classification model

The workflow underlying the construction of the classification model is illustrated in Figure 2‑11. It is made of 3 main tasks: splitting polygons for training and testing, randomly selecting training pixel samples and random forest training. They are detailed in the above sub-sections.

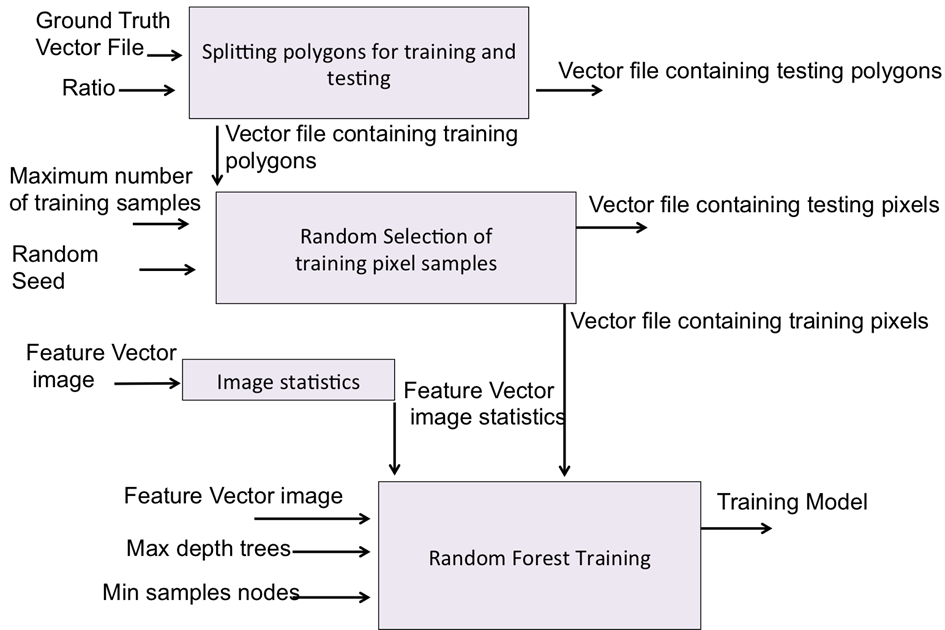


Figure 2‑11. Workflow of the classification model construction

#### Splitting polygons for training and testing

This first task consists in splitting the in-situ dataset into 2 disjoint subsets: the training set and the validation set (Figure 2‑12).

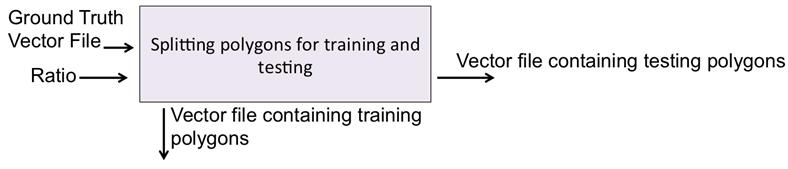


Figure 2‑12. Workflow of the in-situ dataset splitting sub-task

These sets are composed of polygons, not individual pixels. The algorithm is therefore a random sampling without replacement of the polygons of each class with probability p = ratio value for the training set and 1 − p for the validation set. An illustration is the whole process is provided in Figure 2‑13, with a ratio of 0.5 between the 2 subsets.

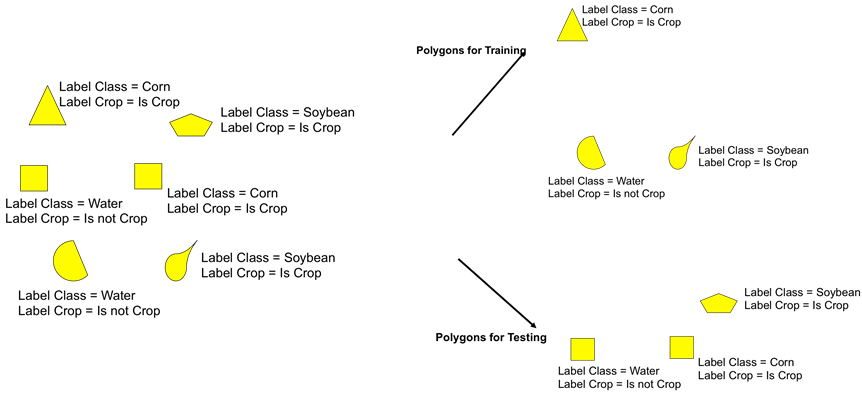


Figure 2‑13. Illustration of the in-situ dataset split into two subsets for training and validation using a ration equal to 0.5

Algorithm 2‑4 describes the procedure. Input and output variables of the algorithm are given in Table 2‑13. The output sets are made of polygons and not individual pixels.

Table 2‑13. Variables for the splitting the field dataset for training and validation

|  |  |  |
| --- | --- | --- |
| **Input variables** | **Role** | **Default value** |
| Reference\_polygons | Vector file containing in-situ data | - |
| Sample\_ratio | Parameter expressing the ratio between the number of training and validation polygons per class (corn, soybean, rice, no-crop, etc.) | 0.75 |
| **Output variables** | **Role** | **Default value** |
| Training\_polygons | Vector file containing in-situ data for training | - |
| Validation\_polygons | Vector file containing in-situ data for validation | - |

Algorithm 2‑4. OTB application performing the feature images concatenation

|  |
| --- |
|  |

#### Random selection of training pixel samples : Crops and Not Crops

This step consists in randomly selecting a number of samples (i.e. pixels) for each class : “crop” and “no-crop”. The workflow is illustrated in Figure 2‑14. The output selection will be class consistent, meaning that if 33% of crops belong to soybean class, 33% of the crop class training samples will also belong to the soybean class.

This reduction of the training sample size is needed to reduce the time complexity of the RF classifier. If it is possible from the operational point of view, this step could be removed and the set of the training pixels samples will correspond to the set of the training polygon samples.

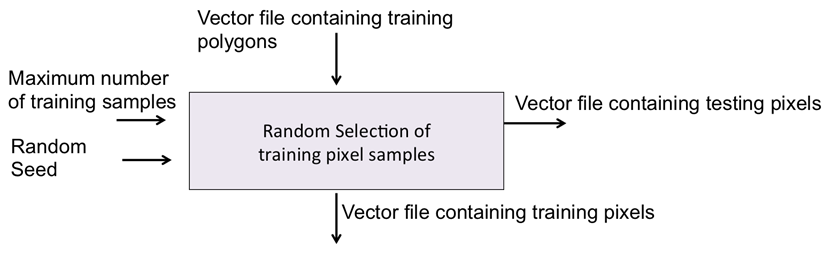


Figure 2‑14. Workflow for reducing the size of the training sample

This random selection is done using a C++ code compatible with OTB, which has been developed (Algorithm 2‑5). This code is inspired from the OTB class “*otbListSampleGenerator*”.

Input and output variables are given in Table 2‑14. As for the input vector file containing training polygons, it must be remarked that the polygons of this vector file as associated with two kinds of label: class label and crop label. For instance, a polygon will be composed by the class label “corn” and the crop label “Is a crop”.

Table 2‑14. Variables for the splitting the field dataset for training and validation

|  |  |  |
| --- | --- | --- |
| **Input variables** | **Role** | **Default value** |
| reference\_polygons | Vector file containing in-situ data for training obtained by the previous splitting algorithm | - |
| Nb\_tr\_samples | Maximum number of training samples: this is the limited number of training samples that will be used for training. For instance, if this number is equal to 1000, it means that the output data will be made of 1000 crop and 1000 no crop samples. | 1000/5000 |
| random\_seed | Random seed that will be used to do the random selection of the pixels. If the third parameter is not specified, a random seed will be generated automatically | 0 |
| **Output variables** | **Role** | **Default value** |
| training\_vector\_file | Vector file containing the limited in-situ data for training. Polygons will be associated with one label: crop (with value 1) or no-crop (with value 0) | - |
| testing\_vector\_file | vector file containing the data from Reference\_polygons which are not included in the training\_vector\_file. Polygons will be associated with one label: crop (with value 1) or no-crop (with value 0) | - |

Algorithm 2‑5. Random selection of training samples

|  |
| --- |
|  |

#### Random Forest Training step

The training of the classifier step is illustrated in Figure 2‑15.



Figure 2‑15. Workflow of the classifier training

This task has been developed on C++ (see the section *Random Forest Training Step task* from the Appendix presented in Section 6) and it is based on the “*otbcli\_TrainImagesClassifier*” application. The OTB application is composed of two steps:

1. Random selection of training and testing samples;
2. Training of the classifier by using the previous training samples.

In our case, the splitting task is done by using the algorithm presented in the previous section (2.4.1.2). For this reason, the developed code implemented in C++ and compatible with OTB Library focuses on the second step.

Input and output variables are presented in Table 2‑15.

Table 2‑15. Variables for training the Radom Forest

|  |  |  |
| --- | --- | --- |
| **Input variables** | **Role** | **Default value** |
| Feature Vector Image | vector image containing the 27 temporal and statistic features | - |
| Feature Vector Image Statistics | the mean and the variance of each feature composing the feature vector image. This result can be obtained by applying the OTB application presented in 2.3.1 of the ATBD crop type document | - |
| Max depth tree | maximum depth of the trees used for Random Forest classifier | 25 |
| Min samples nodes | minimum number of samples in each node used by the classifier | 5 |
| training\_vector\_file | vector file containing the training pixels | - |
| **Output variables** | **Role** | **Default value** |
| Training model | Resulting RF model to be used in the classification step | - |

Concerning the feature vector image statistics, the goal here is to compute the mean and the standard deviation of each input feature of the classifier so that the samples can be normalized inside the training and classification steps.

In order to compute it, the Orfeo Toolbox ComputeImagesStatistics application is used in order to produce an XML file contain- ing the statistics for each channel of the image of features (Algorithm 2‑6).

Algorithm 2‑6. OTB application computing the vector image statistics

otbcli\_ComputeImagesStatistics -il feature-time-series.tif -out statistics.xml

### Classification of the image times series : Random Forest

This step consists in the classification of all the pixels of the image times series made of all the statistic and temporal features (see section 2.3). The general workflow is illustrated in Figure 2‑16.

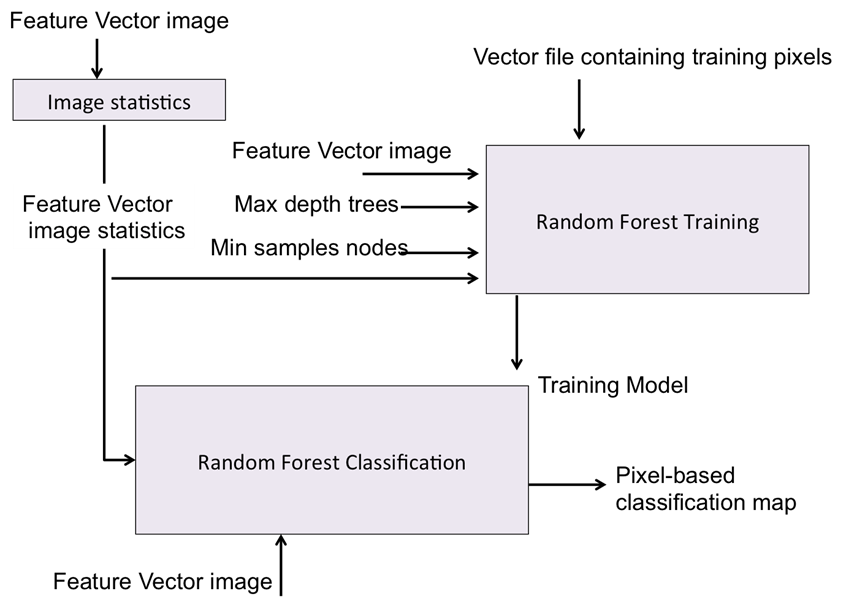
****

Figure 2‑16. Workflow of the RF classification

A C++ code compatible with OTB has been developed for this application. The sources can be found in the files presented in the section *Classification of the complete image times series task*of the Appendix presented at Section 0) and it is based on the “*otbcli\_ImageClassifier”* application.

Input and output variables are given in Table 2‑16.

Table 2‑16. Variables for the RF classification algorithm

|  |  |  |
| --- | --- | --- |
| **Input variables** | **Role** | **Default value** |
| Feature Vector Image | Vector image containing the 27 temporal and statistic features | - |
| Feature Vector Image Statistics | Mean and variance of each feature composing the feature vector image | - |
| Training model | Resulting RF model to be used in the classification step | - |
| **Output variables** | **Role** | **Default value** |
| Pixel-based classification map | Resulting crop mask product | - |

## Processing chain working in the iterative operational model

The processing chain detailed in the section 2.3.4 (i.e. the classification step) will have to run in an operational mode in which the cropland mask will be obtained iteratively. Two different cases must be taking into account regarding the satellite acquisitions: the first year of Sentinel-2 images and the other years. These two situations are analyzed here after.

### The first year of Sentinel-2 images

The “in-situ processing chain” works if it exists ground truth data from the current year. Satellite data captured from the 6 first months will be necessary to produce the first cropland mask product. Considering that the first cropland map will be constructed at instant T, Figure 2‑17 shows how the cropland product will be computed in an iterative way for the following instants of time.

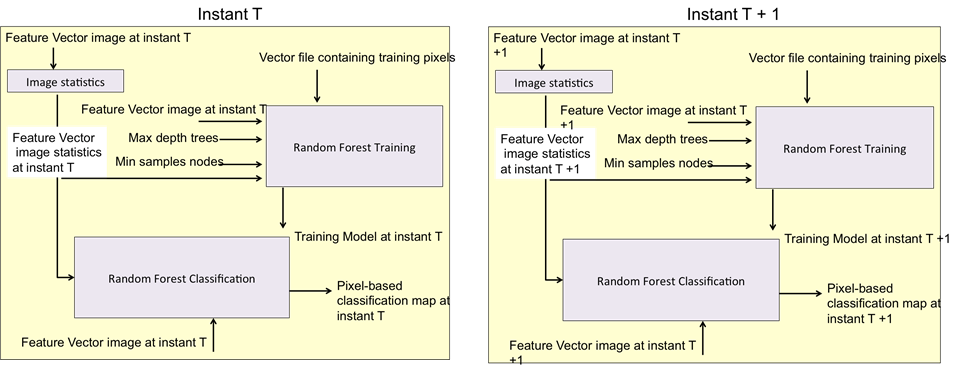


Figure 2‑17. “In-situ” classification chain running at instants T and T+1, when we are during the first S2 year

The right figure shows how since the instant T, the feature image will be recomputed at each instant of time. Besides, this image will be used: (i) to compute the image statistics, (ii) to train the classifier and (iii) to classify the complete image.

### The rest of the years

For the rest of the years, some considerations have been done:

1. A complete year of data acquisitions exists from the previous year;
2. The ground truth in-situ data exists from the previous year;
3. The satellite data of the current year have been interpolated, in order to have the same acquisition dates than during the previous year.

According to these last assumptions, a classifier model is computed for the previous year. This allows developing the system presented in Figure 2‑18 and Figure 2‑19. In these figures, Year X-1 corresponds to the previous year, whereas Year X corresponds to the current year.

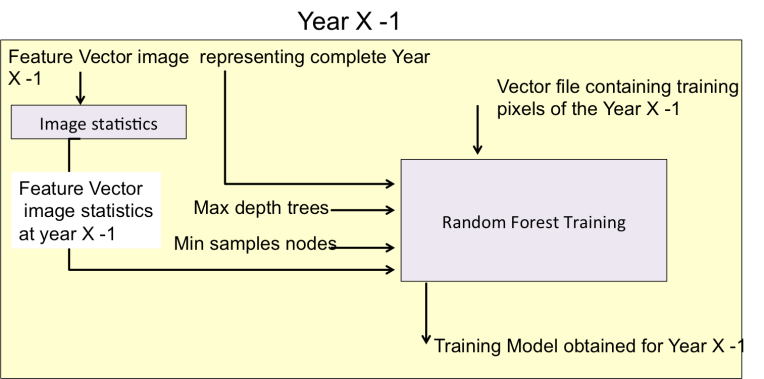


Figure 2‑18. “In-situ” classification chain running in Year X-1, when we are not during the first S2 year

By using the classifier model obtained for the previous year (i.e. built on the in-situ data and the images from the previous years), the classification of the current year will be done, after each new acquisition captured at instant T, as illustrated in Figure 2‑19.

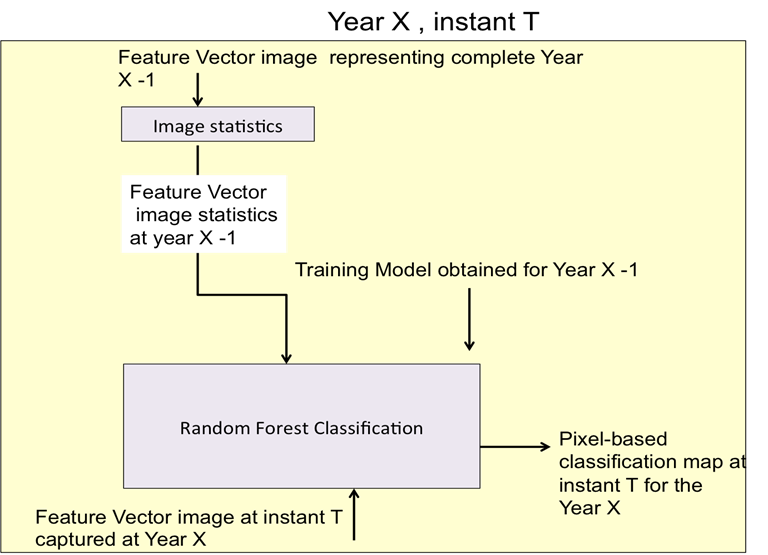
****

Figure 2‑19. “In-situ” classification chain running in Year X, when we are not during the first S2 year

As an implementation note, each input tile is classified only once instead of once per stratum intersecting it, therefore reducing the amount of disk IO required.

# Mode without in-situ data

## Preliminary step: data smoothing and gap filling

A Whittaker filter is applied to the time series, both on the reflectance bands and on the NDVI (Figure 3‑1). The filter is applied using the R function “*Whit1*”. Whit1 is a function from the “ptw” R package, which is a weighted Whittaker smoothing with a first order finite difference penalty. Additional info can be found at <http://cran.r-project.org/web/packages/ptw/ptw.pdf>.

The goal of this Whittaker filter is (i) to smooth the reflectance time series taking into account the non-constant temporal step. To do so, the temporal gaps in the original time series are filled by empty images in order to get one image for each date of the period. Every pixels of the empty images have a weight of 0 in the weight vector of the Whittaker function .

And (ii) to fill the gap in the reflectance time series. To do so, the missing data in the original acquisitions are identified thanks to validity masks which distinguish the valid pixels (value=1) from non-valid pixels (value=0) based on each image cloud mask. These validity masks feed the Whittaker and correspond to the weights of the acquisitions.

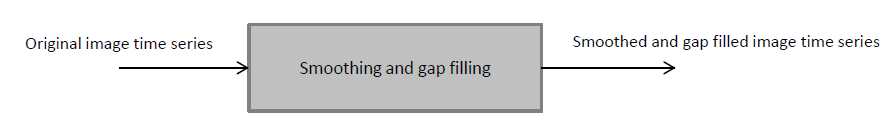


Figure 3‑1. Workflow of the smoothing preliminary step

The Whittaker filter is presented in detail in Algorithm 3-1, while all input and output variables are listed in Table 3‑1. Inputs are the original image time series ; they are presented here separately for the reflectance channels (band<n>) and for the NDVI (NDVI). Reflectance channels include B3, B4, B5, B6, B7, B8, B8a and B11 for Sentinel-2 images and B3, B4, B5 and B6 for Landsat 8. In the outputs, each raster file will have a number of bands equal to the number of image acquisition dates.

The smoothing is applied separately for each time series (green, red, near infrared, red-edge bands and NDVI) and the results are then concatenated. The time series is given as a vector.

Table 3‑1. Variables for the data smoothing

|  |  |  |
| --- | --- | --- |
| **Input variables** | **Role** | **Default value** |
| band<n> | original time series (reflectance values) temporally filled with empty images to get a new time series with one image per date. | - |
| NDVI | original NDVI time series temporally filled with empty images to get a new time series with one image per date. | - |
| Lambda | Smoothing parameter of the Whitaker function (larger values lead to more smoothing) | 2 |
| weights | Weights time series: Default weights are equal to one signal to be smoothed.  -  (i) weight of 0 related to the empty images.  (ii) the “validity masks” related to each acquisition. The validity mask has to be preprocessed to have a value of 1 when the pixel is marked as valid, and a value of 0 in all other cases (a vector of same length as the time series to be smoothed). | 1 |
| **Output variables** | **Role** | **Default value** |
| band<n>\_smoothed | Raster files including the smoothed and gap-filled reflectance values for all the dates | - |
| NDVI\_smoothed | Raster files including the smoothed and gap-filled NDVI values for all the dates | - |

Algorithm 3‑1. Smoothing/Gap Filling based on the Whitaker function

|  |  |  |
| --- | --- | --- |
| |  | | --- | | **Algorithm “**smoothing/gap-filling” **is**  **Temporary output:**  *n* raster files, *n* corresponding to the different spectral bands  *band\_<n>*  *## Creating the validity masks of each acquisition*  For each of the acquisition:  select the cloud masks of the original image time series corresponding  Assign a value of 1 to the *land* pixels and a value of 0 to the other categories  Store and save these layers in *validity\_mask* (one band per input image).  *## Creating the smoothed and gap-filled temporal series of the different spectral bands####*  *dates* <- length in days of input date interval  For each pixel of the *band<n>*:  *weight* <- new array of length `*dates*` filled with 0  for each pixel index, value in *validity\_mask*:  if value is VALID:  *d* <- date of image `index`  *weight*[d] <- 1  for each band in the input raster:  *pixel* <- band reflectance values for the current pixel  *vec* <- array of length `dates` filled with 0  for each index, value in pixel:  *d* <- date of image `index`  *vec*[d] <- value of the pixel | |  | |

## Features extraction

This step is organized in two consecutive sub-tasks: the definition of temporal features and the spectral features extraction (Figure 3‑2). They are detailed in the two sub-sections 3.2.1 and 3.2.2.

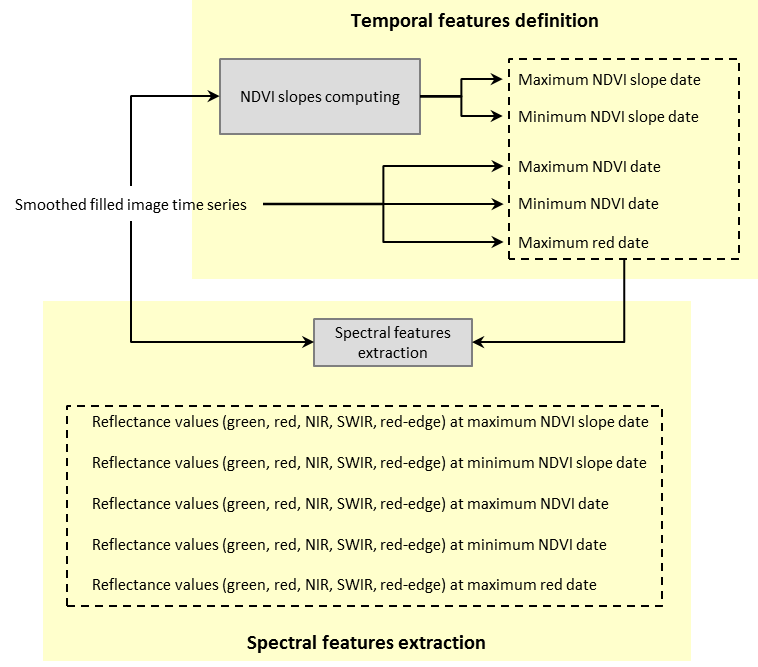


Figure 3‑2. Workflow of the temporal and spectral features extraction process

### Temporal features definition

This step identifies key dates during the growing cycle, based on the reflectance values and NDVI time series.

First, it computes the slopes of the “NDVI *vs* time” curve using the OTB function “*otbcli\_BandMath*” (<https://www.orfeo-toolbox.org/CookBook/CookBooksu132.html>). The algorithm calculates the slope of the NDVI vs time curve according to the following equation:



Then, it identifies on a per-pixel basis the date when the maximum and minimum slopes are achieved and the maximum and minimum NDVI are achieved. The algorithm also identifies when the maximum value of the red band is achieved. These processes are based on the R functions “*which.max*” and “*which.min*”. They are detailed in Algorithm 3‑2. The algorithm runs by tile and stores the dates in 5 separate raster (one by feature).

Input and output of this step are presented in Table 3‑2, as well as the parameter of the otbcli\_BandMath function. The R functions“*which.max*”, “*which.min”* don’t have parameters.

Table 3‑2. Variables for the temporal features identification

|  |  |  |
| --- | --- | --- |
| **Input variables** | **Role** | **Default value** |
| band<n>\_smoothed | Raster files including the smoothed reflectance values for all the dates | - |
| NDVI\_smoothed | Raster files including the smoothed NDVI values for all the dates | - |
| Image dates | Text file giving the acquisition dates of each image within the time series | - |
| Exp | Parameter of the otbcli\_BandMath function, which is a mathematical expression | E.g. (im1(b3)-im1(b1))/difft) |
| **Output variables** | **Role** | **Default value** |
| date\_max\_slope | Raster file which stores, for each pixel, the date corresponding to the maximum slope of the “NDVI vs time” curve | - |
| date\_min\_slope | Raster file which stores, for each pixel, the date corresponding to the minimum slope of the “NDVI vs time” curve | - |
| date\_max\_NDVI | Raster file which stores, for each pixel, the date corresponding to the maximum NDVI value | - |
| date\_min\_NDVI | Raster file which stores, for each pixel, the date corresponding to the minimum NDVI value | - |
| date\_max\_red | Raster file which stores, for each pixel, the date corresponding to the maximum red value | - |

Algorithm 3‑2. Temporal features identification

|  |
| --- |
| **Algorithm “**temporal features identification” **is**  **Temporary output:**  *d* raster (Tif) files, *d* corresponding to the different dates of acquisition corresponding to the slope of the curve “NDVI *vs* time”.  *NDVI\_slope\_<d>*  *## Computing the slopes of NDVI####*  Read the image dates in the *Image\_dates* text file  For *d* in 3:(length(dates)) *(//The loop starts with the 3rd date, because the equation needs three followed dates to be computed)*  Compute “diffdate” as : dates[d] - dates[d-2]  Run the **otbcli\_BandMath** function:  system( **otbcli\_BandMath** -il *NDVI\_smoothed* -out *NDVI\_slope\_<d>* -exp '(im1b[d-2]-im1b[d])/diffdate’)  End of For  *## Identification of the temporal features ####*  *## Look for maximum and minimum NDVI slope####*  List the files which contain dates of slopes (output of previous steps):  inFileList <-list of files including all *NDVI\_slope\_<d>*  In the list, identify the maximum and minimum slopes using the **which.max** and **which.min** R functions:  whmax = **which.max**(inFileList)  whmin = **which.min**(inFileList)  save “whmax” as a raster in *date\_max\_slope*  save “whmin” as a raster in *date\_min\_slope*    *## Look for max/min ndvi value####*  Identify the maximum and minimum NDVI values using the **which.max** and **which.min** R functions:  whMm = **which.max**(*NDVI\_smoothed*)  whmm = **which.min**(*NDVI\_smoothed*)  save “whMm” as a raster in *date\_max\_NDVI*  save “whmm” as a raster in *date\_min\_NDVI*    *## Look for max red value####*  Identify the maximum Red value using the **which.max** R function:  whmax = **which.max** (*band2\_smoothed*)  save “whmax” as a raster in *date\_max\_red*  *## Correction of dates to correspond with smoothed data (since we cannot compute the slope for the first & last dates) => offset=+1 ####*  Run the **otbcli\_BandMath** function:  system( **otbcli\_BandMath** -il *date\_max\_slope* -out *date\_max\_slope* -exp 'im1b1+1')  system( **otbcli\_BandMath** -il *date\_min\_slope* -out *date\_min\_slope* -exp 'im1b1+1') |

### Spectral features extraction

This step consists in associating the key dates to their spectral values: for each pixel, the spectral information corresponding to the dates identified for the 5 key dates of the growing cycle are identified and stored in a new raster. This is achieved thanks to a C++ program developed in the project and presented in Algorithm 3‑3.

Input and output data are listed in Table 3‑3. Reflectance channels include B3, B4, B5, B6, B7, B8, B8a and B11 for Sentinel-2 images and B3, B4, B5 and B6 for Landsat 8. In the outputs, each raster file contains n layers (n being the number of spectral band).

Table 3‑3. Variables for the spectral features extraction

|  |  |  |
| --- | --- | --- |
| **Input variables** | **Role** | **Default value** |
| band<n>\_smoothed | Raster files including the smoothed reflectance values for all the dates | - |
| date\_max\_slope | Raster file which stores, for each pixel, the date corresponding to the maximum slope of the “NDVI vs time” curve | - |
| date\_min\_slope | Raster file which stores, for each pixel, the date corresponding to the minimum slope of the “NDVI vs time” curve | - |
| date\_max\_NDVI | Raster file which stores, for each pixel, the date corresponding to the maximum NDVI value | - |
| date\_min\_NDVI | Raster file which stores, for each pixel, the date corresponding to the minimum NDVI value | - |
| date\_max\_red | Raster file which stores, for each pixel, the date corresponding to the maximum red value | - |
| inbands | Vector of the different raster files corresponding to each of the smoothed spectral bands | E.g. (‘band1\_smo.tif', ‘band2\_smo.tif', 'band3\_smo.tif, 'band4\_smo.tif', ‘band5\_smo.tif’) |
| **Output variables** | **Role** | **Default value** |
| ref\_max\_slope | Raster file which stores, for each pixel, the reflectance values corresponding to the maximum slope date | - |
| ref\_min\_slope | Raster file which stores, for each pixel, the reflectance values corresponding to the minimum slope date | - |
| ref\_max\_NDVI | Raster file which stores, for each pixel, the reflectance values corresponding to the maximum NDVI date | - |
| ref\_min\_NDVI | Raster file which stores, for each pixel, the reflectance values corresponding to the minimum NDVI date | - |
| ref\_max\_red | Raster file which stores, for each pixel, the reflectance values corresponding to the maximum red date | - |

Algorithm 3‑3. Spectral features extraction

|  |
| --- |
| **Algorithm “**spectral features extraction” **is**  *## Extraction of reflectance values in each spectral band for the 2 dates corresponding to the maximum and minimum slope ####*  define the parameters:  infile<- *date\_max\_slope*  inbands<-(*bandi\_smoothed*)  outfile<- *ref\_max\_slope*  run the **selectInList**  function:  system (**selectInList**  outfile infile inbands)  define the parameters:  infile<- *date\_min\_slope*  inbands<-( *bandi\_smoothed*)  outfile<- *ref\_min\_slope*  run the **selectInList**  function:  system (**selectInList**  outfile infile inbands)  *## Extraction of reflectance values in each spectral band for the 2 dates corresponding to the maximum and minimum NDVI ####*  define the parameters:  infile<- *date\_max\_NDVI*  inbands<-( *bandi\_smoothed*)  outfile<- *ref\_max\_NDVI*  run the **selectInList**  function:  system(**selectInList**  outfile infile inbands)  define the parameters:  infile<- *date\_min\_NDVI*  inbands<-( *bandi\_smoothed*)  outfile<- *ref\_min\_NDVI*  run the **selectInList**  function:  system (**selectInList**  outfile infile inbands)  *## Extraction reflectance values in each spectral band for the date corresponding to the maximum Red ####*  define the parameters:  infile<- *date\_max\_red*  inbands<-( *bandi\_smoothed*)  outfile<- *ref\_max\_red*  run the **selectInList**  function (see Algorithm 3‑4):  system( **selectInList** outfile infile inbands) |

Algorithm 3‑4. « SelectInList » program

|  |
| --- |
| **Program** SelectInList **is**:  for each pixel : (// loop over all the pixels)  id = dateID (// dateID is the file containing the key date to select (*date\_max\_slope, date\_min\_slope, date\_max\_NDVI, date\_min\_NDVI, date\_max\_red*)  for each input file i:  output(i) = pixelMB(i)(id) (// pixelMB is a list of multiband files (*band<n>\_smoothed*) with the first index for list item and the second for band) |

## Reference map preparation

The reference is the map that will be used to define the training samples that will feed the coming classification algorithm. This is the key dataset that allows generating a cropland mask in the absence of field data.

This step consists in removing, from the reference map, the pixels that are at the border of the different classes (or said differently, that are between two classes) as they have a higher probability to be wrongly classified (Figure 3‑3).



Figure 3‑3. Workflow of the reference map preparation

The process is based on a function named **classErosion** (Algorithm 3‑5), written in python during the benchmarking and that relies on the python **erosion** function from the “*skimage.morphology*” module.

Input and output variables are given in Table 3‑4, as well as the parameters of the python **erosion** function.

Table 3‑4. Variables for the erosion of the reference

|  |  |  |
| --- | --- | --- |
| **Input variables** | **Role** | **Default value** |
| reference | Raster file containing the original reference map, with the same extent and projection than the image time series | - |
| Image | Image array corresponding to each class | classinRst |
| selem | Binary array made of 0 and 1 values | sq(3) |
| **Output variables** | **Role** | **Default value** |
| eroded\_reference | Raster file containing the eroded reference map, with the same extent and projection than the image time series. Removed pixels are coded with NaN values | - |

Algorithm 3‑5. Reference erosion

|  |
| --- |
| **Function** classErosion **is**  def **classErosion**(*Reference*, *Eroded\_reference*, size=3):  ## Reading the input ####  *Reference* is opened and stored in “inRst”  ## Identifying the different classes included in the reference ####  The different classes of “inRst” are identified and included in “uniqueClass” *(where the classes are identified from 1 to u)*  ## Defining the size of the erosion ####  selem = sq(size) *(where size is “3” according to Table 3‑4 => selem is squared matrix of 3x3 ‘ones’)*  ## Erosion of the different classes ####  for *u* in “uniqueClass”:  all the areas where “inRst” == u are saved in “classinRst”  “eroded” = **erosion**(classinRst, selem)  the resulting eroded areas are saved in “outRst”    ## Saving the output file ####  An array is written compiling the “outRst” coming from each class  The array is saved in *Eroded\_reference* |

## Trimming

Trimming consists in truncating a distribution from its least probable values that behave like outliers. The common purpose of this procedure is to reduce the sensitivity to outliers for many parameter estimates, such as the sample mean and variance.

Trimming, coupled with the previous reference map erosion, are two key steps that give robustness to the classification process. They will allow obtaining, from an inexact pre-existing reference map data that will be “clean” enough to train a classifier. They are therefore the core of the “without in-situ mode”.

The spectral distribution corresponding to each class of the eroded reference is iteratively trimmed in order to eliminate the pixels that behave as “outliers” from their Gaussian distribution point of view.

The trimming will not be performed on all spectral channels, but on the ones that are the most characteristic for crops – no crops. The selected channels are the red and NIR reflectance values at the minimum NDVI date as well as the green, red and NIR reflectance values at the maximum NDVI date.

The trimming algorithm is described in Algorithm 3‑6. It uses a “*MVG\_IT*” function (Algorithm 3‑7) specifically created in R that computes iteratively the outliers of a Gaussian distribution. “*MGV\_IT*” uses, in turn, two functions from the base core of R, which are the “*qchisq***”** and “*mahalanobis***”** ones.

From the trimming output (i.e. the identification of clean pixels in the reference map), a selection of samples is done randomly. This selection will constitute the training dataset for the subsequent RF classifier.

It shall be noted that even if the trimming was done on a selection of spectral channels, the RF classifier will run with all available spectral information: green, red, NIR, SWIR and red-edge reflectance values from the dates corresponding to the maximum red, minimum and maximum NDVI and minimum and maximum slope.

The workflow of the trimming process is presented in Figure 3‑4.

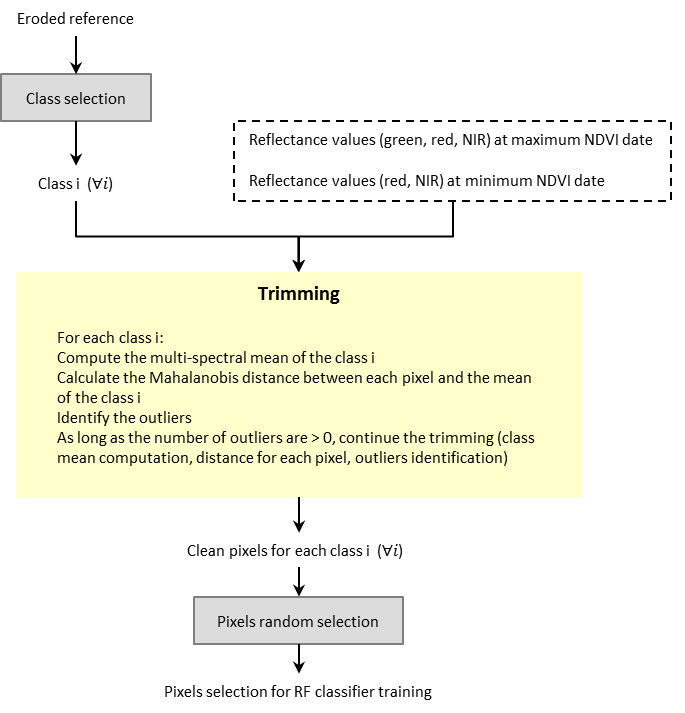


Figure 3‑4. Workflow of the trimming process

Input and output variables of the trimming are presented in Table 3‑5.

Table 3‑5. Variables for the trimming

|  |  |  |
| --- | --- | --- |
| **Input variables** | **Role** | **Default value** |
| eroded\_reference | Raster file containing the eroded reference map, with the same extent and projection than the image time series. Removed pixels are coded with NaN values | - |
| ref\_max\_NDVI | Raster file which stores, for each pixel, the reflectance values corresponding to the maximum NDVI date | - |
| ref\_min\_NDVI | Raster file which stores, for each pixel, the reflectance values corresponding to the minimum NDVI date | - |
| ntrain | Number of samples needed for the RF training | 1000 |
| TrimBands | Selection of spectral bands that will be used for the trimming | Red and NIR from the minimum NDVI date // green, red and NIR from the maximum NDVI date |
| x | Parameter of the MVG\_IT function: vector or matrix of reflectance values of a specific class for which the Gaussian model will be fitted | E.g. subset(trimRst, LC\_class == class.i) where trimRst are the selected spectral bands defined by TrimBands |
| alpha | Parameter of the MVG\_IT function: confidence interval used to define what is an “outlier” | 0.01 |
| x | Parameters of the qchisq function: vector of quantile values | (1-alpha), where alpha is 0.01 |
| df | Parameters of the qchisq function: degrees of freedom | number of spectral band used for the trimming : 5 |
| x | Parameters of the mahalanobis function: vector or matrix of data | x.trim (matrix of reflectance values considered as “clean” at the current iteration) |
| center | Parameters of the mahalanobis function: vector of the mean distribution of each spectral band | colMeans(x.trim) |
| cov | Parameters of the mahalanobis function: covariance matrix of the distribution | cov(x.trim) |
| **Output variables** | **Role** | **Default value** |
| training\_sample | Data frame which contains the samples that will be used for the training | - |

Algorithm 3‑6. Trimming

|  |
| --- |
| **Algorithm** Trimming **is**  *## Define input working files ####*  Store *Eroded\_Reference* in “labelRst”  Store the spectral features (*ref\_max\_ndvi, ref\_min\_ndvi*) in “featRst”  Define the number of samples needed for the RF training in “ntrain”  *## Convert input working files in a data frame ####*  Store “featRst” as data frame in “metricRst”  Store “labelRst” as data frame in “LCstack”  Store “metricRst” and “LCstack” as data frame in “Rststack”  *## Select reflectance values to use in the trimming (i.e. reflectance values whose Gaussian distribution will be cleaned) ###*  Select the bands *TrimBands* (see Table 3‑5 for this parameter) from “Rststack”  Select the class information corresponding to “LCstack” (class information) from “Rststack”  Store this information (selected bands and class information) in “trimRst”  *## Create a list of classes ###*  List all classes ID in a list “classList”  *## Run the trimming, with the function MVG\_IT ###*  Create the list “trimList” that will receive the results of the trimming for the different classes  For each class ‘i ‘ from the “classList”  class.i <- i  X <- selection of trimRst corresponding to class.i  X.NP\_IT <- **MVG\_IT**(X)  trimList[i]<- including the result X.NP\_IT for the class ‘i’    *## From the trimming output trimList, make a random selection for each class of n pixels that will be used to train the RF ####*  For each class ‘i ‘ from the “classList”  subset.idx<- selects a random sample of “ntrain” pixels of the class i  out.id<-list of ones and zeros with the size of “trimList”, ‘1’ corresponding to the selected pixels (subset.idx)  trimmed\_values.sbs<-subset of “trimList” corresponding to out.id==1  *Training\_Sample* <- trimmed\_values.sbs |

Algorithm 3‑7. MVG\_IT function

|  |
| --- |
| **Function** MVG\_IT **is**  **MVG\_IT** <- function(x, alpha = 0.01){  chi <- **qchisq**((1-alpha), df=NCOL(x))  *(computes the Chi-Squared distribution and selects the threshold corresponding to the alpha p-value of the distribution per class)*  x.trim <- x  (do) while a outlier is still identified  md <- as.matrix(**mahalanobis**(x.trim, colMeans(x.trim), cov(x.trim)))  *(Computing the Mahalanobis distance to the mean for each pixel)*  x.trim <- x.trim which md is lower or equal to chi  *(Selecting the “clean” pixels, i.e. those pixels whose mahalanobis distance to the mean is lower than the threshold)*  }  it returns x.trim as output of the function  } |

## Random forest classifier

The classification is a supervised method relying on a RF classifier, just like in the “in-situ” mode. The RF classifier is applied to the reflectance values associated with the key dates (maximum and minimum NDVI slopes, maximum and minimum NDVI, maximum red). Reflectance values are those associated with the following spectral channel: B3, B4, B5, B6, B7, B8, B8a and B11 for Sentinel-2 images and B3, B4, B5 and B6 for Landsat 8. The RF classifier uses as training data the “clean” pixels selected by the trimming.

This step follows exactly the same logical than the RF classification used for the “in-situ” mode. The presentation is thus rather brief.

### RF model estimation

The process follows the same logical than for the “in-situ” mode, detailed in section 2.4.1.3. It is thus based on the “*otbcli\_TrainImagesClassifier*” OTB application. The OTB application is composed of two steps, which are run here:

1. Random selection of training and testing samples;
2. Training of the classifier by using the previous training samples.

In our case, the random selection is already done in the previous step (as a final step of the trimming - see section 3.4). Just like for the “in-situ mode”, the developed code implemented in C++ and compatible with OTB Library therefore focuses on the second step.

Input and output variables listed in Table 3‑6.

Table 3‑6. Variables for the RF model estimation in the « no in-situ » mode

|  |  |  |
| --- | --- | --- |
| **Input variables** | **Role** | **Default value** |
| Feature Vector Image | Vector image containing the spectral features “ref\_max\_slope”, “ref\_min\_slope”, “ref\_max\_NDVI”, ref\_min\_NDVI”, “ref\_max\_date” | - |
| Feature Vector Image Statistics | Mean and variance of each feature composing the feature vector image | - |
| training\_sample | Data frame which contains the samples that will be used for the training | - |
| rf.nbtrees | Number of decision trees | 100 |
| rf.min | Minimum number of samples per nodes | 5 |
| rf.max | Maximum depth of each tree | 25 |
| **Output variables** | **Role** | **Default value** |
| training\_model | Resulting RF model to be used in the classification step | - |

### Classification

The classification process applies the RF model built in the previous step using the training dataset on the image time series in order to generate the cropland mask.

Just like for the in-situ data mode, the RF algorithm is based on a C++ code compatible with OTB which has been developed for this application. The sources can be found in the files presented in the section *Classification of the complete image times series task*of the Appendix presented at Section 0) and it is based on the “*otbcli\_ImageClassifier”* application.

Input and output variables are listed in Table 3‑7.

Table 3‑7. Variables for the RF classification for the « no in-situ » mode

|  |  |  |
| --- | --- | --- |
| **Input variables** | **Role** | **Default value** |
| Feature Vector Image | Vector image containing the spectral features “ref\_max\_slope”, “ref\_min\_slope”, “ref\_max\_NDVI”, ref\_min\_NDVI”, “ref\_max\_date” | - |
| Feature Vector Image Statistics | Mean and variance of each feature composing the feature vector image | - |
| training\_model | Resulting RF model to be used in the classification step | - |
| **Output variables** | **Role** | **Default value** |
| Pixel-based classification map | Resulting crop mask product | - |

# Common final steps

## Classifier Evaluation: Computing evaluation quality measures

The classification results will be evaluated by using the in-situ data:

* Supervised Approach: in this case, it will be done by using all the vector files which contain the testing pixels and polygons which have not been used for the training. (Table 2‑13).
* Unsupervised Approach: the evaluation can be done by using all the in-situ data.

The workflow of this evaluation is illustrated in Figure 4‑1.

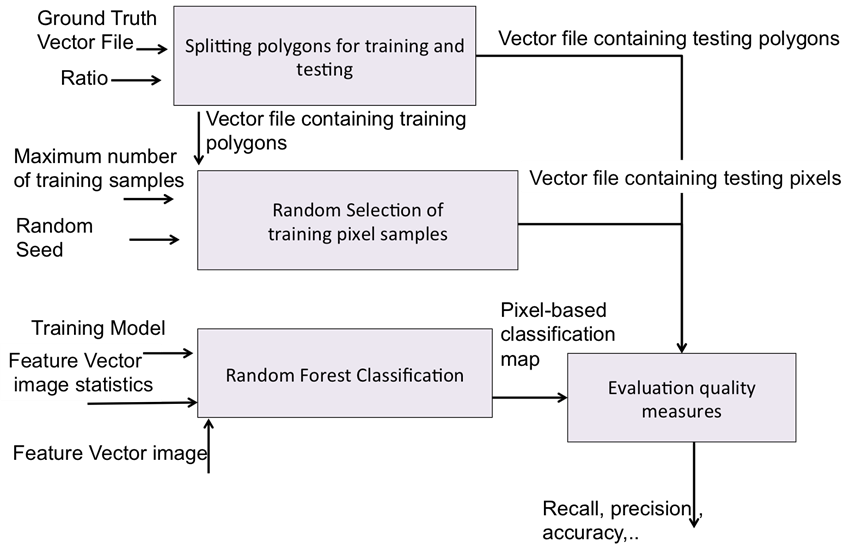


Figure 4‑1. Workflow of the classification assessment

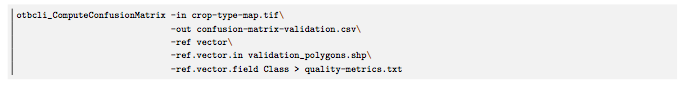
The evaluation quality indices (recall, precision, accuracy, etc. ) that are computed are detailed in crop mask chapter of the Design Justification File.

All these indices are derived from a confusion matrix result. This confusion matrix can be computed by using the OTB application “*otbcli\_ComputeConfusionMatrix*”, as it is shown in the Appendix of Section 0). The algorithm is run by tiles, as the whole processing chain.

Table 4‑1. Variables for the map validation algorithm

|  |  |  |
| --- | --- | --- |
| **Input variables** | **Role** | **Default value** |
| Pixel-based classification map | Resulting crop mask product | - |
| Validation\_polygons | Vector file containing in-situ data for validation (to be used only to validate the product obtained without in-situ data) | - |
| Reference\_polygons | Vector file containing all in-situ data (to be used only to validate the product obtained without in-situ data) | - |
| **Output variables** | **Role** | **Default value** |
| confusion\_matrix\_validation | File containing the confusion matrix | - |
| quality\_metrics | File containing the quality metrics | - |

Algorithm 4‑1. Computation of quality indices



## Crop mask post-filtering

As presented in Figure 4‑2, the post-filtering task is made of 3 steps: (i) the Principal Component Analysis (PCA), (ii) the mean-shift algorithm and (iii) the majority vote. These steps are detailed in Figure 4‑2.

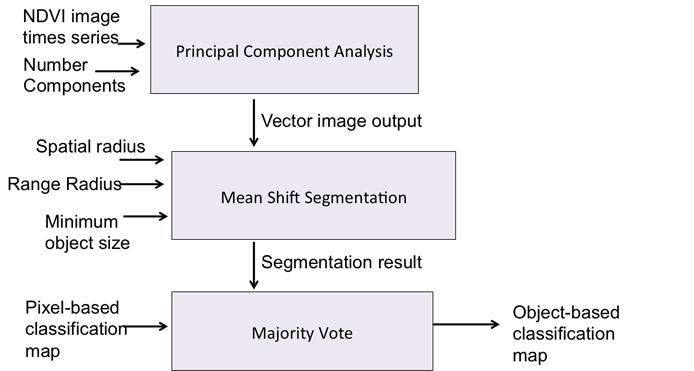


Figure 4‑2. Workflow of the post-filtering step

### Principal Component Analysis

The Principal Component Analysis steps is run with a ++ code compatible with OTB, which has been developed (see code described in section *Principal Component Analysis task*of theAppendix presented in Section 0). It uses the otb filter class “*otb::PCAImageFilter*”.

Input and output variables are listed in Table 4‑2.

Table 4‑2. Variables for the Principal Component Analysis

|  |  |  |
| --- | --- | --- |
| **Input variables** | **Role** | **Default value** |
| NDVI image times series | Vector image containing the complete NDVI image times series | - |
| Number of Components | Number of principal components composing the output image | Maximum: 6 images  Minimum: number of available images |
| **Output variables** | **Role** | **Default value** |
| Vector image | Vector image containing the “Number of Components” principal component images | - |

### Mean-Shift segmentation algorithm

The Mean-Shift segmentation is run with a ++ code compatible with OTB, which has been developed (see the code described in the section *Mean Shift Segmentation task* of theAppendix presented in Section 0). It uses the OTB filter class “*otb::MeanShiftVectorImageFilter*”.

Input and output variables are listed in Table 4‑3.

Table 4‑3. Variables for the Mean-Shift segmentation

|  |  |  |
| --- | --- | --- |
| **Input variables** | **Role** | **Default value** |
| Vector image output | vector image containing the result of the PCA | - |
| Spatial radius | spatial radius of the neighborhood | 10 |
| Range radius | range radius defining the radius (expressed in radiometry unit) in the multispectral space | 0.65 |
| Minimum object size | Minimum size of a region (in pixel unit) in segmentation. Smaller clusters will be merged to the neighboring cluster with the closest radiometry. If set to 0 no pruning is done | 10 |
| **Output variables** | **Role** | **Default value** |
| Segmentation result | segmentation result image | - |

### Majority vote

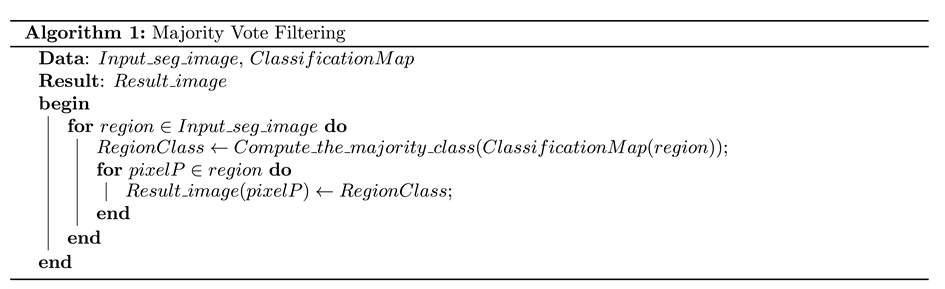
The majority vote algorithm is presented in Algorithm 4‑2.

Input and output variables are listed in Table 4‑4.

Table 4‑4. Variables for the majority voting filtering

|  |  |  |
| --- | --- | --- |
| **Input variables** | **Role** | **Default value** |
| Segmentation result | Segmentation result image | - |
| Pixel-based classification map | Pixel-based classification results obtained by the RF classifier (coming from the “in-situ” mode or not) | - |
| **Output variables** | **Role** | **Default value** |
| Object-based classification map | Object-based classification map result | - |

Algorithm 4‑2. Majority vote filtering



### Operational context

It must be remembered that in the operational context, the post-filtering task must also be done following an iterative procedure. This iterative approach will be possible from the first satellite data acquisitions. This can be obtained by following the next flowcharts, Figure 4‑3 and Figure 4‑4.

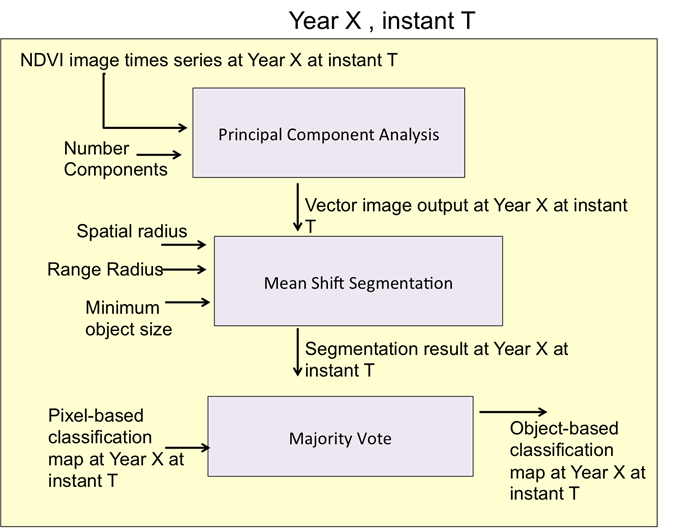


Figure 4‑3. Post-filtering workflow, at instant T

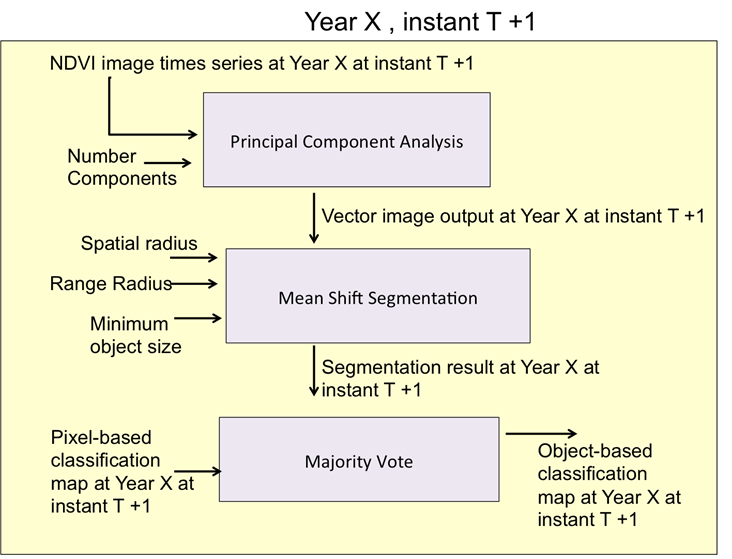


Figure 4‑4. Post-filtering workflow, at instant T+1

# Processing chain model working for a large area covering several satellite footprints

## Large region classification

The processor must work with sites that larger than e.g. a Sentinel-2 scene. As the training data is hard to acquire and will probably not be distributed uniformly, it must be used to the full extent. This means that a single classification model should be used for the entire site instead of one per input tile.

Because of the size of the input data, it's preferable to use an online approach, where the classification features are computed on-the-fly instead of saving them to disk. This is especially important for larger sites, as the amount of disk space taken by the extracted features would be prohibitive. Another advantage of an online method is that computing the features is faster than saving them to disk and reading them back, even when multiple passes over are needed. This was an important optimization that enabled running the processor over national sites within the available time and hardware constraints.

## Stratification

Some larger sites might have multiple regions, which we'll call strata, with different climatic characteristics. In these cases it might be desirable to use a different classification model for each stratum instead of a single one for the entire site. There are two potential advantages to this approach:

* it's possible that using stratum-specific models gives better classification results;
* training a classifier for a single stratum can require fewer resources than for the complete site.

Thus, the system should allow users to define a list of strata of a site. It can take the form of a shapefile with one feature (most often a polygon) per strata. Besides the extent, the only required information is an identifier, which is then used in the output product metadata. The implementation expects numeric identifiers, although there is no special reason for this.

The stratum shapefile is used as follows:

1. if available, the in-situ data is intersected with the strata and split into one data set for each stratum;  
   2. for each stratum, the data set preparation and training is performed just like in the single-stratum case;
2. during the classification, the corresponding model is applied to each pixel, yielding a single raster for each input file;
3. any post-filtering that might be required is applied afterwards;

The current implementation does not use any transition zones between strata. Some discontinuities will be present at the stratum boundaries, but the results were found to be acceptable in practice.

# Quality flags

For each release, the cropland mask will be provided along with four following quality flags (REQ-2.9\_URD):

* the number of dates (i.e. L2A products) which are associated with the “land” status during the period used to generate the mask;
* the number of dates (i.e. L2A products) which are associated with the “water” status during the period used to generate the mask;
* the number of dates (i.e. L2A products) which are associated with the “snow” status during the period used to generate the mask;
* the number of dates (i.e. L2A products) which are associated with the other statuses (“cloud”, “cloud shadow”, “no data”’) during the period used to generate the mask. During the processing chains, the dates associated with these statuses will be interpolated.

# Appendix - Additional information on the codes of the « in-situ » processing chain

The code, associated with the « in-situ » mode (including the common steps with the “no in-situ” mode) is available on: <http://tully.ups-tlse.fr/valeros/s2agri_cropmask>

Codes from the OTB project are the following ones:

* **NDVI feature computation task:**

Includes/NdviParameterExtraction.hpp

tests/IndicesComputation/NdviParameterExtractionApplication.cxx

* **Statistics feature computation task:**

includes/IndexParameterExtraction.hpp

tests/IndicesComputation/IndexParameterExtractionApplication.cxx

* **Random selection of training pixel samples task:**

includes/ShpFileSampleGeneratorFast.hpp

src/ShpFileSampleGeneratorFast.cxx

* **Random Forest Training Step task:**

test/sClassification/ ShpFileMRFClassificationFast.cxx

* **Classifiication of the complete image times series task:**

tests/Classification/SVMClassificationMap.cxx

* **Principal Component Analysis task:**

tests/PostFiltering/ACP.cxx

* **Mean Shift Segmentation task:**

tests/PostFiltering/SegmentationACP.cxx

Two algorithms were coded in Python:

* Quality measures (how to evaluate the crop masks – section 4.1)
* Majority vote (how to apply the majority vote – section 4.2.3)