Sen4CAP - Sentinels for Common Agricultural Policy

Design Justification File

ATBD for L4A crop type mapping

Milestone	Milestone 1
Authors	Nicolas BELLEMANS, Sophie BONTEMPS, Pierre DEFOURNY, Laurentiu NICOLA, Philippe MALCORPS
Distribution	ESA - Benjamin KOETZ

This page is intentionally left blank

Table of recorded changes

Version	Date	Reason
v1.0		First version
v1.1	11/04/2019	Update to fit with new developments in the processor, mainly concerning the object classification steps
v1.2	23/07/2019	Implementation of the crop diversification use case in the ATBD
v1.3	01/04/2021	End of project - final version

Table of contents

1. Logical model – overview of the system 10
2. Data preparation 12
 2.1 Subsidy application layer 12
  2.1.1 Input data 12
  2.1.2 L4A crop code and crop diversification information 15
 2.2 Optical data 16
 2.3 SAR data 16
  2.3.1 Mosaicking and formatting 16
  2.3.2 Gap-filling consideration 21
3. Feature extraction 22
 3.1 Optical data 22
  3.1.1 Feature extraction 22
  3.1.2 Parcel level statistics extraction 23
 3.2 SAR data 24
  3.2.1 Ratio VV/VH 24
  3.2.2 Feature concatenation 24
  3.2.3 Temporal features 26
  3.2.4 Parcel level statistics extraction 31
4. Object classification 32
 4.1 Format object feature statistics 32
  4.1.1 Import object feature statistics: from ‘line raster’ to csv 32
  4.1.2 Features concatenation 32
 4.2 Select parcels for training, classification and validation 33
  4.2.1 Non-assessed parcels 35
  4.2.2 Parcels used for calibration and validation 35
 4.3 Apply SMOTE algorithm to synthetically over-sample the minority classes 37
 4.4 Train the Random Forest model 38
 4.5 Classify and format the classification output table 39
 4.6 Update the subsidy application layer with the classification results 40
5. Validation 41
6. Crop diversification use case 43
 6.1 Context 43
 6.2 Preparation 45

6.2.1 Standardized subsidy application layer with quality flags and results of the classification 45

6.2.2 Crop code LUT 47

6.3 Process 47

6.3.1 Parameters definition 47

6.3.2 Conformity assessment at the parcel level 48

6.3.3 Summarized factors by holding 50

6.3.4 Category assessment at the holding level 52

6.3.5 Crop diversification assessment at the holding level 54

6.4 Output 57

7. Output 59

7.1 Results of the crop type mapping 59

7.2 Results of the crop diversification assessment 59

7.3 Validation results 60

7.4 Classification related data 61

List of figures

Figure 1-1. General workflow of the L4A crop type mapping production 11
Figure 2-1. Selection of the used S2 pixels by parcel 13
Figure 2-2. Workflow of the temporal resampling of the optical data 16
Figure 2-3.Workflow of the mosaicking step to produce weekly mosaics of S1 images 16
Figure 3-1. Workflow of the features extraction of the optical data 22
Figure 3-2.Workflow of the parcel level statistics extraction 23
Figure 3-3. Workflow of the SAR data feature extraction steps 24
Figure 3-4. (a) Exemple of monthly coherence over a Winter Wheat field in Netherlands (b) Mean of the coherence value of March 2017 over Netherlands 28
Figure 3-5.Workflow of the parcel level statistics extraction 31
Figure 4-1. Example of csv containing the unique ID and the declared crop type oof the parcel along with the optical features statistic per parcel 32
Figure 4-2. Specific workflow for the selection of the parcels used by the classification scheme. The by-default values are the following ones: LC_monitored = [1,2,3,4], S2pixMIN = 3, S1pixMIN = 1, PaMIN = 30, S2pixBEST = 10, PaCalibH = 4000, PaCalibL = 1333, Sample_ratioH = 0.25, Sample_ratioL = 0.75, smote_size = 1000. 33
Figure 4-3. Non-assessed parcels by the classification scheme. The by-default values are the following ones: LC_monitored = [1,2,3,4], S2pixMIN = 3, S1pixMIN = 1, PaMIN = 30. 35
Figure 4-4. Parcels classified and used for calibration and validation. The by-default values are the following ones: S2pixBEST = 10, PaCalibH = 4000, PaCalibL = 1333, Sample_ratioH = 0.25, Sample_ratioL = 0.75, smote_size = 1000. 36
Figure 6-1 Crop diversification regulations from the Technical guidance for the On-The-Spot checks of Crop Diversification [RD.7] considered in the Sen4CAP project and correspondence with the Sen4CAP crop diversification categories 44

List of tables

Table 2-1. Standardized subsidy application layer with quality flags 12
Table 2-2. Content of the standardized subsidy application layer with quality flags 12
Table 2-3. Parcels raster layers 14
Table 2-4. Content of the L4A crop code LUT 14
Table 2-5. Specific variables for the mocaicking and formatting of the S1 time series 17
Table 3-1. Specific input variables for the optical data features extraction 22
Table 3-2. Specific input variables for the extraction of feature statistics per parcel 23
Table 3-3. Specific variables for the computation of the SAR ratio 24
Table 3-4. Specific variables for the feature concatenation of the SAR time series data 25
Table 3-5. Specific variables for the temporal features computation based on the S1 time series 28
Table 3-6. Specific input variables for the extraction of feature statistics by parcel from the S1 time series 31
Table 4-1. Specific variables for the selection of parcels for classification, training and validation 34
Table 4-2. Specific input variables for the SMOTE algorithm 38
Table 4-3. Specific input variables for the model Random Forest 38
Table 4-4. Specific variables for the RF model application for S2 and S1 time series classification 39
Table 4-5. Output attribute fields with the results of the RF classification (Predict_classif.csv) 40
Table 5-1. Specific variables for the validation 41
Table 6-1 Crop diversification categories considered in the Sen4CAP crop diversification use cases 45
Table 6-2. Content of the csv file exported from the subsidy application layer 46
Table 6-3 Input variables of the crop diversification process 48
Table 6-4. Summarized factors by holding 50
Table 6-5. Crop diversification category assessment 52
Table 6-6. Crop diversification compliancy assessment 54
Table 6-7. Content of the crop diversification output crop_div.csv 57
Table 6-8. Content of the crop diversification output crop_div_holding.csv 58
Table 7-1. Content of the output shapefile export 59
Table 7-2. Content of the crop diversification output crop_div.csv 59
Table 7-3. Content of the crop diversification output crop_div_holding.csv 60

References

ID	Title
RD.1	Inglada J. and Arias M. 2017, Sen2-Agri project, Design Definition File - Algorithm Therotical Basis Document for L4 Crop type product
RD.2	Chawla N., Bowyer K., Hall L., Kegelmeyer W., 2002. SMOTE: synthetic minority over-sampling technique, Journal of artificial intelligence research, 16, 321-357.
RD.3	Bunkhumpornpat C., Sinapiromsaran K., Lursinsap C.. 2009. Safe-level-smote: Safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem. Pacific-Asia conference on knowledge discovery and data mining. 475-482.
RD.4	Breiman, L. Machine Learning (2001) 45: 5. https://doi.org/10.1023/A:1010933404324
RD.5	Sen4CAP Design Justification File: ATBD for Subsidy application layer Preparation (April 2019)
RD.6	Vajsova B., Fasbender DK, Wirnhardt C., Lemajic S., Astrand P.J., 2018, Addressing the small parcels issue (https://ec.europa.eu/jrc/sites/jrcsh/files/11-sifting_and_hhr.pdf)
RD.7	JRC (2015), Technical guidance for the On-The-Spot checks of Crop Diversification, DS-CDP-2015-08, JRC96614

List of acronyms

Acronym	Definition
ATBD	Algorithm Theoretical Basis Document
GSAA	GeoSpatial Aid Application
LPIS	Land Parcel Identification System
LUT	Look-Up Table
NDVI	Normalized Difference Vegetation Index
NDWI	Normalized Difference Water Index
PA	Paying Agency
RF	Random Forest
S1	Sentinel-1
S2	Sentinel-2
SAR	Synthetic Aperture Radar
Sen2-Agri	Sentinel-2 for Agriculture
SMOTE	Synthetic Minority Over-Sampling Technique
SWIR	Short-Wave Infrared
UTM	Universal Transvere Mercator

1. Logical model – overview of the system

The developed object-based classification scheme relies on a Random Forest (RF) algorithm, run on optical Sentinel-2 (S2) surface reflectance and Synthetic Aperture Radar (SAR) Sentinel-1 (S1) statistics time series at the parcel level.

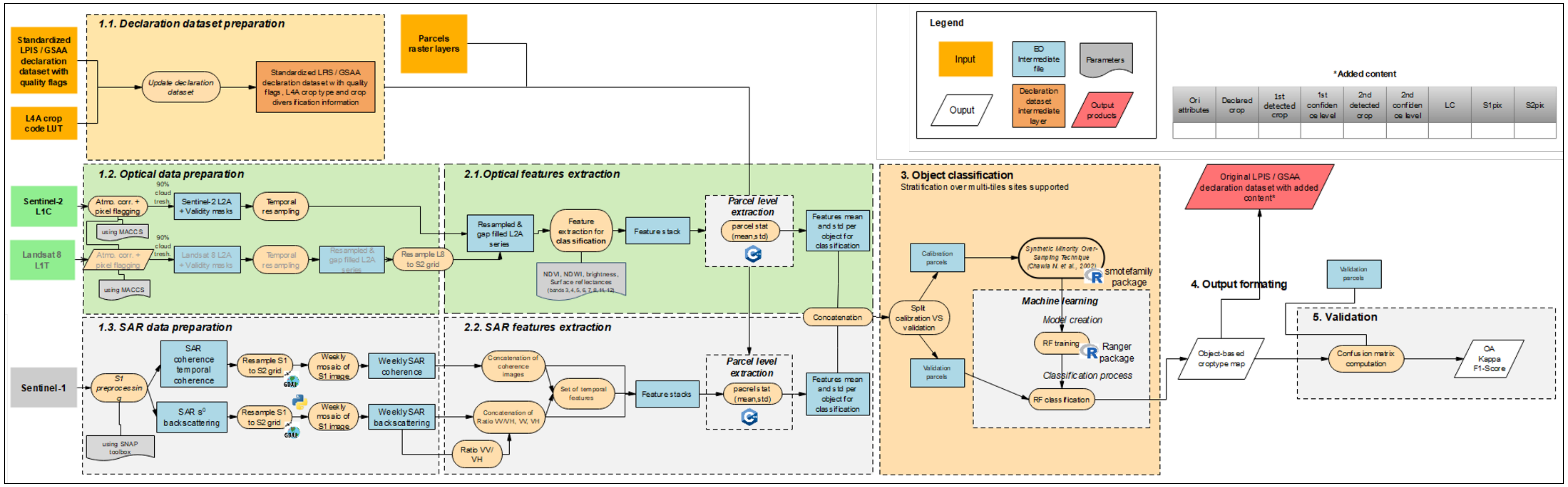
Figure 1-1 presents the general workflow of the crop type mapping processing chain. There are 5 main components:

1. Data preparation
   a. Declaration data
   b. Optical data
   c. SAR data
2. Features extraction
   a. Optical data
   b. SAR data
3. Object classification
   a. Format object feature statistics
   b. Select parcels for training, classification and validation
   c. Apply SMOTE algorithm to synthetically over-sample the minority classes
   d. Train the Random Forest model
   e. Classify and format the classification output table
4. Output formating
5. Validation

Optical data preparation and optical features extraction modules were directly inherited from existing modules in the ESA Sentinel-2 for Agriculture (Sen2-Agri) processing system.

The developed classification scheme has been designed to allow classifying an area covering multiple tiles, meaning that a single classification model is used for an entire stratum instead of one by input tile. This leads in more consistent results in similar eco-climatic environement, better classification results and fewer resources usage than single tile model for multi-tile site.

In the following sections, the different steps are presented in details. For most of these steps, the specific input and output variables, as well as the code or pseudo-code, are given.



Standardized LPIS / GSAA declaration dataset with quality flags
L4A crop code LUT
1.1. Declaration dataset preparation
Update declaration dataset
Standardized LPIS / GSAA declaration dataset with quality flags, L4A crop type and crop diversification information
Parcels raster layers
Legend
Input
EO Intermediate file
Parameters
Ouput
Declaration dataset intermediate layer
Output products
*Added content
Ori attributes
Declared crop
1st detected crop
1st confidence level
2nd detected crop
2nd confidence level
LC
S1pix
S2pix
Sentinel-2 L1C
Landsat 8 L1T
1.2. Optical data preparation
Atmo. corr. + pixel flagging
using MACCS
90% cloud tresh.
Sentinel-2 L2A + Validity masks
Temporal resampling
Landsat 8 L2A + Validity masks
Resampled & gap filled L2A series
Resample L8 to S2 grid
2.1.Optical features extraction
Resampled & gap filled L2A series
Feature extraction for classification
NDVI, NDWI, brightness, Surface reflectances (bands 3, 4, 5, 6, 7, 8, 11, 12)
Feature stack
Parcel level extraction
parcel stat (mean,std)
Features mean and std per object for classification
Concatenation
Sentinel-1
1.3. SAR data preparation
S1 preprocessing
using SNAP toolbox
SAR coherence temporal coherence
SAR s⁰ backscattering
Resample S1 to S2 grid
Weekly mosaic of S1 image
Weekly SAR coherence
Weekly SAR backscattering
2.2. SAR features extraction
Concatenation of coherence images
Concatenation of Ratio VV/VH, VV, VH
Ratio VV/VH
Set of temporal features
Feature stacks
Parcel level extraction
pacrelstat (mean,std)
Features mean and std per object for classification
3. Object classification
Stratification over multi-tiles sites supported
Split calibration VS validation
Calibration parcels
Validation parcels
Synthetic Minority Over-Sampling Technique (Chawla N. et al., 2002)
smotefamily package
Machine learning
Model creation
RF training
Ranger package
Classification process
RF classification
Original LPIS / GSAA declaration dataset with added content*
4. Output formating
Object-based croptype map
Validation parcels
5. Validation
Confusion matrix computation
OA Kappa F1-Score

Figure 1-1. General workflow of the L4A crop type mapping production

2. Data preparation

2.1 Subsidy application layer

2.1.1 Input data

In order to ensure a certain level of consistency between the different Sen4CAP processors (L4A crop type mapping, L4B grassland mowing detection and L4C agricultural practices monitoring), the preparation of the subsidy application layer is performed prior to the execution of these processors. The subsidy application layer preparation is described in a dedicated ATBD (RD.5). The outputs of the subsidy application layer preparation that are used by the L4A crop type mapping processor are described below.

2.1.1.1 Standardized subsidy application layer with quality flags

The standardized subsidy application layer with quality flags:

- is stored as a PostGIS layer in the PostgreSQL database of the system;
- is projected in national projection;
- has the following name: decl_{site_name}_{year};
- has the same number of rows (parcels) than the original subsidy application layer.

Table 2-1. Standardized subsidy application layer with quality flags

Output data	Description	Default value [format]
decl_{site_name}_{year}	The standardized version of the subsidy application layer with the quality flags: geometry and spectral information	[PostGIS]

It contains the following attribute fields (fields in orange are already present in the original subsidy application layer) (Table 2-2).

Table 2-2. Content of the standardized subsidy application layer with quality flags

Field name	Role	Default value [format]
Ori attributes	All the original attributes of the original delaration dataset	[integer, float or string]
ori_id	Copy of the content of the attribute field defined by the user with the parcel id	[string]
ori_hold	Copy of the content of the attribute field defined by the user with the holding id	[string]
ori_crop	Copy of the content of the attribute field defined by the user with the crop code	[input format: string or integer]
NewID	New sequential ID of the parcel	[integer]
HoldID	New sequential ID of the holdings	[integer]

GeomValid	Identify parcels for which no polygon exists in the subsidy application layer or with a not valid geometry	[integer, binary]
Duplic	Identify parcels that have the exact same geometry as another	[integer, binary]
Area_meters	Parcel area in the UTM projection (m²)	[integer]
Overlap	Identify parcels which overlaps with neighbouring parcels	[integer, binary]
ShapeInd	The crop type name	[float]
S1pix	Indicates the number of used S1 pixels in the parcel	[integer]
S2pix	Indicates the number of used S2 pixels in the parcel	[integer]

2.1.1.2 Parcels raster layers

The parcels raster layers are the rasters that are produced for both data (S2 and S1) and by tile, with the NewID as value. Only the pixels that have their centroid located in the parcels inner buffer layers of 5 m for S2 and 10 m for S1, have been assigned the NewID of the parcel as value (Figure 2-1). Only these pixels will be used to extract the spectral values from the Sentinel-1 (S1) and Sentinel-2 (S2) data.

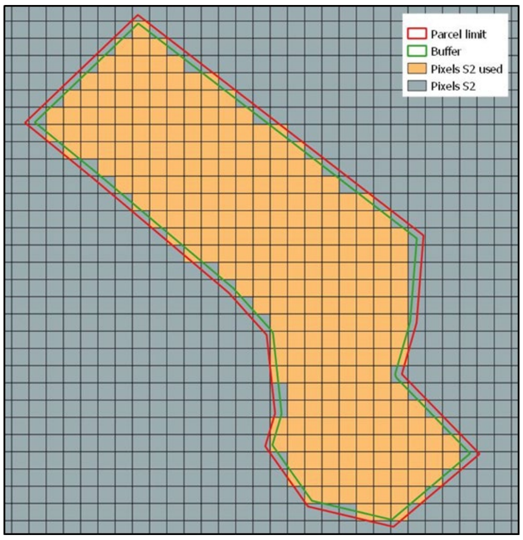


Figure 2-1. Selection of the used S2 pixels by parcel

These layers:

- are .tif files;
- are produced by S2 tile;
- are projected in the WGS 84 / UTM zone {x} corresponding to the UTM zone of the S2 tile;
- have a value = NewID of the parcels;
- have a resolution of 10m for S2 and 20m for S1.

Table 2-3. Parcels raster layers

Output variable	Role	Default value [format]
{country}_{year}_decl_{i}_S2	Raster of all the used S2 pixels by parcel (value as NewID); {i} = name of the tile (ex. 31UFS)	[tif] (nr of tiles)
{country}_{year}_decl_{i}_S2	Raster of all the used S1 pixels by parcel (value as NewID); {i} = name of the tile (ex. 31UFS)	[tif] (nr of tiles)

2.1.1.3 Crop code LUT

If the original subsidy application layer contain a large number of crop types, it considerably improves the classification accuracy to group together the crop types that are by definition very similar or that has a very similar phenology. It is done by defining a new crop code (CTnumL4A) with a new crop name (CTL4A) in the crop code LUT.

In addition, to check the compliancy of the holdings regarding the crop diversification rules, a series of information should be defined by crop type: the crop diversification class (CTnumDIV and CTDIV) and whether or not it belongs to one or more of the categories Eligible Agricultural Area (EAA), Arable Land (AL), Permanent grassland, Temporary grassland, Fallow land and Crop under water.

All this information is summarized in a csv file, the crop code Look-Up Table (LUT), which:

- is stored as a table in the PostgreSQL database of the system;
- is named: lut_{country}_{year};
- contains the following information (Table 2-4).

Table 2-4. Content of the L4A crop code LUT

Field name	Role	Default value [format]
Ori_crop	The initial crop code from the subsidy application layer	[integer or string]
CTnum	The new crop type code (each Ori_crop being associated to a unique CTnum)	[integer]
CT	The name of the crop type in English	[string]
LC	The main land cover class of the crop type: ○ 0: other natural areas ○ 1: annual crop ○ 2: permanent crop ○ 3: grassland ○ 4: fallow land ○ 5: greenhouse and nursery	[integer]
CTnumL4A	The new crop type code resulting of the grouping of the CTnum for the classification	[integer]
CTL4A	The crop type name associated to CTnumL4A	[string]
CTnumDIV	The crop diversification class code	[integer]

CTDIV	The crop diversification class name	[string]
EAA	Eligble agricultural area: value 1 if the crop type belongs to this category, value 0 otherwise	[integer, binary]
AL	Arable Land: value 1 if the crop type belongs to this category, value 0 otherwise	[integer, binary]
PGrass	Permanent grassland: value 1 if the crop type belongs to this category, value 0 otherwise	[integer, binary]
TGrass	Temporary grassland: value 1 if the crop type belongs to this category, value 0 otherwise	[integer, binary]
Fallow	Fallow land: value 1 if the crop type belongs to this category, value 0 otherwise	[integer, binary]
Cwater	Crop under water: value 1 if the crop type belongs to this category, value 0 otherwise	[integer, binary]

2.1.2 L4A crop code and crop diversification information

This step joins the information contained in the L4A crop type LUT to the standardized subsidy application layer with quality flags. The following fields, for which the definition is given in Table 2-4, are added to the attribute table:

- CTnumL4A;
- CTL4A;
- CTnumDIV;
- CTDIV;
- EAA;
- AL;
- PGrass;
- TGrass;
- Fallow;
- Cwater.

The join is made using the **ori_crop** field from the stansardized subsidy application layer with quality flags and the **Ori_crop** field of the crop code LUT.

The join shall keep all the original parcels of the subsidy application layer (same number of rows as the original subsidy application layer).

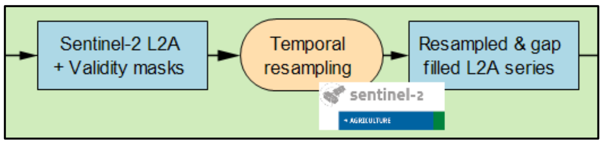
The standardized subsidy application layer with quality flags is updated with the following PostGIS query (Algorithm 2-1):

Algorithm 2-1. Update of the subsidy application layer with related quality flags

```
update declaration_dataset
set "CTnumL4A" = L4A_lut.ctnuml4a,
    "CTL4A" = L4A_lut.ctl4a,
    "CTnumDIV" = L4A_lut.ctnumdiv,
    "CTDIV" = L4A_lut.ctdiv,
    "EAA" = L4A_lut.eaa,
    "AL" = L4A_lut.al,
    "PGrass" = L4A_lut.pgrass,
    "TGrass" = L4A_lut.tgrass,
    "Fallow" = L4A_lut.fallow,
    "Cwater" = L4A_lut.cwater,
from L4A_lut
where L4A_lut.Ori_crop = subsidy_application_layer.ori_crop;
```

2.2 Optical data

The optical data preparation step is directly inherited from the *Sen2-Agri* system. It corresponds to a temporal resampling and a gapfilling through linear interpolation of valid data (Figure 2-2).



Sentinel-2 L2A + Validity masks
Temporal resampling
Resampled & gap filled L2A series
sentinel-2

Figure 2-2. Workflow of the temporal resampling of the optical data

As stated in the RD.1, the objective of the temporal resampling is to generate a reflectance image time series which is gap-filled with respect to missing data and temporally resampled on a regular 10-day grid. Missing data are refered to as the data masked as cloud, cloud shadow and saturated pixels. The 10-day period of the resampling was chosen instead of the 5-day to avoid an over-fitting of the RF classifier that can be easily caused by using very correlated data.

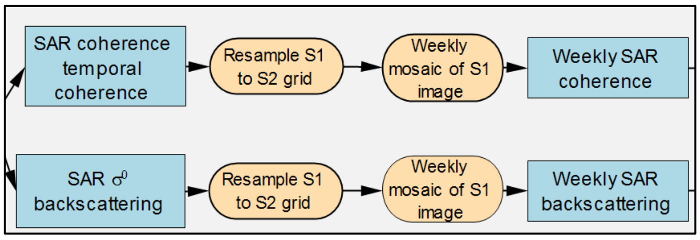
For the implementation details, please refer to the RD.1.

Within the *Sen2-Agri* system, this step is coded in the form of the OTB application and is merged together with the feature extraction step. The input and output variables are discussed in the section 3.1.1.

2.3 SAR data

2.3.1 Mosaicking and formatting

The method is designed to look for all the S1 images overlapping a given S2 tile within a given period. It produces time series of S1 mosaics combining images acquired within 6-day period. This step produces a temporally regular time series of mosaics fully covered by valid pixel (Figure 2-3). For classification purposes, the mosaics shall not be covered by no data pixels.



SAR coherence temporal coherence
Resample S1 to S2 grid
Weekly mosaic of S1 image
Weekly SAR coherence
SAR σ0 backscattering
Resample S1 to S2 grid
Weekly mosaic of S1 image
Weekly SAR backscattering

Figure 2-3.Workflow of the mosaicking step to produce weekly mosaics of S1 images

The mosaics are formated to match the format (resolution, extent and projection) of the S2 image given as input.

The weekly mosaic are generated for each calendar week; this way, the mosaics are independent of the season duration.

The mosaicking step is performed independently for the two passes – ascending and descending – and for the two polarimetry – VV and VH – respectively for the coherence and the backscattering. It therefore results in 8 independent time series of

1. Coherence – Ascending – VV
2. Coherence – Ascending – VH
3. Coherence – Descending – VV
4. Coherence – Descending – VH
5. Backscattering intensity – Ascending – VV
6. Backscattering intensity – Ascending – VH
7. Backscattering intensity – Descending – VV
8. Backscattering intensity – Descending – VH

This operation can be performed using the python script `s2TileExtent_to_s1Mosaic.py`. The needed inputs are presented in Table 2-5 and the script is presented in Algorithm 2-2.

Table 2-5. Specific variables for the mocaicking and formatting of the S1 time series

Input variable	**Role**	**Default value [format]**
S2_ref	Reference S2 raster	[character]
S1_dir	S1 directory	[character]
scratch_dir	Temporary folder to write the s1 tiles warped	
start_date	Start date of the temporal extent	[YYYYMMDD]
end_date	End date of the temporal extent	[YYYYMMDD]
temporal_step	Temporal extent over which the code performs the mean composite i.e. the first composite will be from -start to "start"+"step" included.	7 [DD]
pattern	Space separated list of coma separated patterns that will be considered separately for the composite. Multiple entries allowed NB: only files matching at least one of the pattern will be considered. If you do not want to separate different patterns put at least one pattern to get correct tif in the s1 folders.	VV,VH asc,desc
min	Minimal value for the s1 data. Any smaller value is considered as nodata	[numeric]
max	Maximal value for the s1 data. Any higher value is considered as nodata.	[numeric]
Output variable	**Role**	**Default value [format]**
output_dir	Output folder to write the S1 mean composties. Absolute path required	fid [character]

The script is performing the following operations:

(1) Get extent of the S2 tile given as input by *S2_ref;*
(2) Get list of overlapping S1 files in the *S1_dir;*
(3) Filter with the *pattern;*
(4) Filter with the *start_date* and *end_date;*
(5) Warp each selected S1 images to the S2 tile format (extent and resolution);
(6) Mean compositing over iterative period (expressed in number of days given by the *temporal_step* parameter).

Algorithm 2-2. Mosaicking and formatting of the S1 times series (s2TileExtent_to_s1Mosaic.py)

```
import os
import argparse
import argparse, os, sys, subprocess
from osgeo import gdal, osr, ogr
from shapely import geometry
from datetime import datetime, timedelta

parser = argparse.ArgumentParser(description='Take an S2 image in input and a temporal extent to return
a mosaic of the S1 images overlaping with the S2 spatial extent using mean composite. NB: S1 images must
be in same projection and aligned.')
parser.add_argument('-i', help='Input s2 tile', required=True)
parser.add_argument('-s1', help='Input folder with the S1 tiles', required=True)
parser.add_argument('-o', help='Output folder to write the S1 mean composties. Absolute path required',
required=True)
parser.add_argument('-temp', default='/scratch/', help='Temporary folder to write the s1 tiles warped.')
parser.add_argument('-start', default='20170101', type=str, help='Start date of the temporal extent.
Format YYYYMMDD.')
parser.add_argument('-end', default='20171231', type=str, help='End date of the temporal extent. Format
YYYYMMDD.')
parser.add_argument('-step', default=5, type=float, help='Temporal extent over which the code performs
the mean composite i.e. the first composite will be from -start to "start"+"step" included.')
parser.add_argument('-p', "--patternToSeparate", nargs='*', default=['VV,VH', 'asc,desc',
'cohe,slc_coreg_amp_calib'], help='Space separated list of coma separated patterns that will be
considered separately for the composite. Multiple entries allowed e.g. -p VV,VH asc,desc . NB: only files
matching at least one of the pattern will be considered. If you do not want to separate different patterns
put at least one pattern to get correct tif in the s1 folders. e.g. -p patternInS1productName. Warning:
avoid all special characters like slashes!')
parser.add_argument('-min', default=0, type=float, help='Minimal value for the s1 data. Any smaller value
is considered as nodata.')
parser.add_argument('-max', default=100, type=float, help='Maximal value for the s1 data. Any higher
value is considered as nodata.')
parser.add_argument('-v', help='Verbose version', action='store_true')

args = parser.parse_args()

patternToSeparate = [item.split(',') for item in args.patternToSeparate]
print(patternToSeparate)

def GetExtent(gt, cols, rows):
    ''' Return list of corner coordinates from a geotransform

        @type gt:   C{tuple/list}
        @param gt: geotransform
        @type cols:   C{int}
        @param cols: number of columns in the dataset
        @type rows:   C{int}
        @param rows: number of rows in the dataset
        @rtype:    C{[float,...,float]}
        @return:   coordinates of each corner
    '''
    ext=[]
    xarr=[0,cols]
    yarr=[0,rows]

    for px in xarr:
        for py in yarr:
            x=gt[0]+(px*gt[1])+(py*gt[2])
            y=gt[3]+(px*gt[4])+(py*gt[5])
            ext.append([x,y])
        yarr.reverse()
    return ext

def ReprojectCoords(coords, src_srs, tgt_srs):
    ''' Reproject a list of x,y coordinates.

        @type geom:     C{tuple/list}
        @param geom:    List of [[x,y],...[x,y]] coordinates
        @type src_srs:  C{osr.SpatialReference}
        @param src_srs: OSR SpatialReference object
        @type tgt_srs:  C{osr.SpatialReference}
        @param tgt_srs: OSR SpatialReference object
        @rtype:         C{tuple/list}
        @return:        List of transformed [[x,y],...[x,y]] coordinates
    '''
    trans_coords=[]
    transform = osr.CoordinateTransformation( src_srs, tgt_srs)
    for x,y in coords:
        x,y,z = transform.TransformPoint(x,y)
        trans_coords.append([x,y])
    return trans_coords

def getListOfOverlappingFiles(s2_tile_path, s1_tile_path_list):
  s1_tile_overlapping_list_to_return = []
  s2_tile_datasource = gdal.Open(s2_tile_path)
  s2_srs = osr.SpatialReference()
  s2_proj = s2_tile_datasource.GetProjection()
  s2_srs.ImportFromWkt(s2_proj)
  s2_extent    =    GetExtent(s2_tile_datasource.GetGeoTransform(),    s2_tile_datasource.RasterXSize,
s2_tile_datasource.RasterYSize)
  s2_extent.append(s2_extent[0])
  s2_pol = geometry.Polygon(s2_extent)
  for s1_tile_path in s1_tile_path_list:
    s1_tile_datasource = gdal.Open(s1_tile_path)
    s1_srs = osr.SpatialReference()
    s1_proj = s1_tile_datasource.GetProjection()
    s1_srs.ImportFromWkt(s1_proj)
    s1_extent    =    GetExtent(s1_tile_datasource.GetGeoTransform(),    s1_tile_datasource.RasterXSize,
s1_tile_datasource.RasterYSize)
    s1_extent_reproj = None
    if not s1_proj == s2_proj:
      s1_extent_reproj = ReprojectCoords(s1_extent, s1_srs, s2_srs)
    else:
      s1_extent_reproj = s1_extent
    s1_extent_reproj.append(s1_extent_reproj[0])
    s1_pol = geometry.Polygon(s1_extent_reproj)
    if s1_pol.intersects(s2_pol):
      s1_tile_overlapping_list_to_return.append(s1_tile_path)
  return s1_tile_overlapping_list_to_return

s1tiles = [os.path.join(dirpath, filename) for dirpath, dirnames, filenames in os.walk(args.s1) for
filename    in    filenames    if    (os.path.splitext(filename)[1].lower()    ==    ".tiff"    or
os.path.splitext(filename)[1].lower() == ".tif")]
print("Found %d images (.tiff or .tif, capital letters allowed) in the s1 folder."%len(s1tiles))
s1tiles_with_match = []
for tile in s1tiles:
  hasMatches = []
  for pattern_set in patternToSeparate:
    for pattern in pattern_set:
      if pattern in tile:
        hasMatches.append(True)
  if len(hasMatches) > len(patternToSeparate):
    print("Error: the pattern you provided lead to double match for file %s."%tile)
    sys.exit()
  elif len(hasMatches) == len(patternToSeparate):
    s1tiles_with_match.append(tile)


print("Found %d images matching the patterns in the s1 folder."%len(s1tiles_with_match))

s1_tile_overlapping_list = getListOfOverlappingFiles(args.i, s1tiles_with_match)

print("Found %d images in the s1 folder which overlap with the s2 tile provided in input. The number
includes all product required."%len(s1_tile_overlapping_list))

# Deal with the start/end date and extent
date_interval_list = []
start_date = datetime.strptime(args.start, '%Y%m%d')
interval_start = start_date
end_date = datetime.strptime(args.end, '%Y%m%d')
while interval_start <= end_date:
  interval_end = interval_start + timedelta(days=args.step)
  date_interval_list.append([interval_start, interval_end])
  interval_start += timedelta(days=args.step + 1)

# Prepare for the clip and reprojection of the images to the S2 tile extent/projection
s2_tile_datasource = gdal.Open(args.i)
s2_srs = osr.SpatialReference()
s2_proj = s2_tile_datasource.GetProjection()
s2_srs.ImportFromWkt(s2_proj)
s2_proj_for_warp = s2_srs.GetAttrValue("AUTHORITY", 0) + ":" + s2_srs.GetAttrValue("AUTHORITY", 1)
s2_extent     =     GetExtent(s2_tile_datasource.GetGeoTransform(),     s2_tile_datasource.RasterXSize,
s2_tile_datasource.RasterYSize)
s2_extent_for_gdalwarp = str(s2_extent[1][0]) + " " + str(s2_extent[1][1]) + " " + str(s2_extent[3][0])
+ " " + str(s2_extent[3][1])

slurmconfigtemplate = """#!/bin/bash
#SBATCH --job-name={0}
#SBATCH --output={0}.out
#SBATCH --error={0}.err
#SBATCH --partition=E3_32_new

source /home/tom/libs/bashrc
"""

dict_pattern_images = {}
dict_pattern_warpedimages = {}
dict_pattern_slurm_conf = {}
for s1_tile in s1_tile_overlapping_list:
  # Prepare the gdal warp
  warped_output_name                              =                              os.path.join(args.temp,
os.path.splitext(os.path.basename(s1_tile))[0]+"_warpedToS2.tiff")
  warp_command = "gdalwarp -t_srs {0} -tr {1} -te {2} -tap -r 'near' -dstnodata '{5}' -overwrite {3}
{4}\n".format(s2_proj_for_warp, "10 10", s2_extent_for_gdalwarp, s1_tile, warped_output_name, args.min-
1)

  tile_time_str = os.path.basename(s1_tile)[0:8]
  tile_time = datetime.strptime(tile_time_str, '%Y%m%d')
  tilefamily = ""
  for date_interval in date_interval_list:
    if tile_time >= date_interval[0] and tile_time <= date_interval[1]:
      tilefamily += date_interval[0].strftime('%Y%m%d') + "to" + date_interval[1].strftime('%Y%m%d')
  if tilefamily == "":
    continue
  for patterns in patternToSeparate:
    for pattern in patterns:
      if pattern in s1_tile:
        tilefamily += "_" + pattern
  if not tilefamily in dict_pattern_warpedimages.keys():
    dict_pattern_images[tilefamily] = []
    dict_pattern_warpedimages[tilefamily] = []
    dict_pattern_slurm_conf[tilefamily] = slurmconfigtemplate.format(os.path.join(args.o, tilefamily))
  dict_pattern_images[tilefamily].append(s1_tile)
  dict_pattern_warpedimages[tilefamily].append(warped_output_name)
  dict_pattern_slurm_conf[tilefamily] += "echo Launch %s\n" %warp_command
  dict_pattern_slurm_conf[tilefamily] += warp_command

for tilefamily in dict_pattern_slurm_conf.keys():
  output_mc_name              =              os.path.join(args.o,             tilefamily             +
"_nObs%d.tiff"%len(dict_pattern_images[tilefamily]))
  temp_output = os.path.join(args.temp, tilefamily + ".tiff")
  mc_command    =    "/export/apps/os422/enge/bin/projects/lifewatch/vgt/MCfloat    {0}    {1}    {2}
{3}\n".format(temp_output,  args.min,  args.max,  "  ".join(str(warpeds1tile)  for  warpeds1tile  in
dict_pattern_warpedimages[tilefamily]))
  dict_pattern_slurm_conf[tilefamily] += "echo Launch %s\n"%mc_command
  dict_pattern_slurm_conf[tilefamily] += mc_command
  dict_pattern_slurm_conf[tilefamily]  +=  "echo  File  temporarily  written  at  %s  and  moved  at
%s\n"%(temp_output, output_mc_name)
  dict_pattern_slurm_conf[tilefamily] += "mv %s %s\n"%(temp_output, output_mc_name)
  for warpedimage in dict_pattern_warpedimages[tilefamily]:
    dict_pattern_slurm_conf[tilefamily] += "echo remove %s\n"%warpedimage
    dict_pattern_slurm_conf[tilefamily] += "rm %s\n"%warpedimage
  dict_pattern_slurm_conf[tilefamily] += "echo Everything finished!\n"
  slurmconfname = tilefamily + ".srun"
  with open(slurmconfname, 'w') as slurmconf:
    slurmconf.write(dict_pattern_slurm_conf[tilefamily])

print("All the .srun have been written. You can launch them with sbatch *.srun. (Check one first).")
```

Side note: The `s2TileExtent_to_s1Mosaic.py` *script is configured to generate srun script to be easily launched by slurm.*

2.3.2 Gap-filling consideration

At the time of the processing chain design, no data filtering was applied on the SAR time series. The added value of applying a temporal resampling was therefore not highlighted. Only in the case of missing acquisitions, a gap-filling step must be applied.

3. Feature extraction

3.1 Optical data

3.1.1 Feature extraction

The optical data feature extraction step is directly inherited from the *Sen2-Agri* system using a dedicated OTB application called `otbcli FeatureExtractor` with the "rededge" mode enabled (Figure 3-1). It produces the relevant features used for the classification. The temporal resampling and gap-filling step discussed in the section 2.2 are integrated in this application.

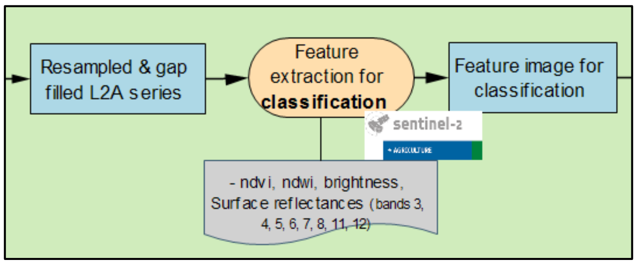

Resampled & gap filled L2A series
Feature extraction for classification
Feature image for classification
sentinel-2
- ndvi, ndwi, brightness, Surface reflectances (bands 3, 4, 5, 6, 7, 8, 11, 12)

Figure 3-1. Workflow of the features extraction of the optical data

As described in RD.1, the features are computed for each date of the resampled and gapfilled time series and conacetenaed together into a single multi-channel image file at 10-m resolution. The selected features are the surface reflectance time series (green (B3), red (b4), NIR (B8), Red-edges (B5-6-7), Short-Wave Infrared (SWIR) 1 (B11) and SWIR 2 (B12)), the Normalized Difference Vegetation Index (NDVI), the Normalized Difference Water Index (NDWI) and the brightness. It results in 11 bands per synthetic date.

One should notice that the NDVI computation uses the 10m broad NIR band (not the B8a). Concerning the 20m resolution bands, only the SWIR 1 (B11) band is resampled at 10m spatial resolution, for the calculation of the NDWI. The rest of these bands (Red-edges (B5-6-7) and SWIR 2 (B12)) are kept at 20m resolution, and the extraction is done using the 20m resolution raster layers derived from the preparation of the subsidy application layer.

The `FeatureExtractor` Orfeo Toolbox application is used as follows (Algorithm 3-1):

Algorithm 3-1. Optical features extraction

```
otbcli FeatureExtractor -il $INPUT_DESCRIPTORS \
                        -out $featureTimeSeries int16 \
                        -rededge true \
                        -sp $SP \
                        -ram 6400 \
                        -mission $MISSION
```

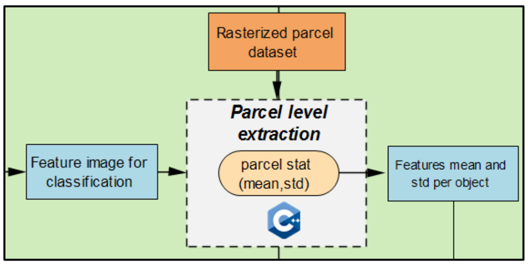
Table 3-1. Specific input variables for the optical data features extraction

Input variable	Role	Default value [format]
INPUT_DESCRIPTORS	S2 input descriptors (HDR files)	[character]
SP	S1 directory	SENTINEL 10

MISSION	Temporary folder to write the s1 tiles warped	SENTINEL
Output variable	**Role**	**Default value [format]**
featureTimeSeries	Output feature time series (multiband raster)	[character]

3.1.2 Parcel level statistics extraction

This step allows extracting mean and standard deviation of each feature over each parcel. It can also extract the number of pixel (*npix*) that is covering each parcel. It crosses the information of the `DeclSTD_Format_raster` (the declaration shapefile rasterized) and the `featureTimeSeries` to create a multiband "line raster" whose pixel index can be link to the parcel ID (Figure 3-2).



Rasterized parcel dataset
Parcel level extraction
Feature image for classification
parcel stat (mean,std)
Features mean and std per object

Figure 3-2.Workflow of the parcel level statistics extraction

It uses a compilled C++ code called `parcelStat` (Algorithm 3-2). It produces as output one multi-band raster per statistic (mean and standard deviation).

Algorithm 3-2. ParcelStat algorithm

```
parcelStat $featureTimeSeries $DeclSTD_Format_raster $nfield $Object_feat_stat -1
```

Table 3-2. Specific input variables for the extraction of feature statistics per parcel

Input variable	**Role**	**Default value [format]**
DeclSTD_Format_raster	S2 input descriptors (HDR files)	[character]
featureTimeSeries	Path of the feature time series	[character]
nfield	Total number of fields	[numeric]
NoData	No data value, value of the pixels not belonging to a parcel	-1 [numeric]
Output variable	**Role**	**Default value [format]**
Object_feature	Name of the output 'line raster' (without extension). Output feature time series is a multiband raster. One raster is produced per statistics. the name of the stat is added to the output name. Output names: - *S2_featureTimeSeries_OBJECT_mean* - *S2_featureTimeSeries_OBJECT_std* - *S2_featureTimeSeries_OBJECT_npix*	[character]

3.2 SAR data

As shown in the Figure 3-3, the VV/VH ratio is computed for each weekly S1 mosaic. The sets of features (coherence and backscattering) are then concatenated independently; VV, VH, ratio and ascending and descending orbits being considered separetly. A set of temporal features are extracted from the coherence, backscaterring and ratio time series. Finally, feature statistics are extracted at the level of the parcel. These statistics will be used as input for the classification.

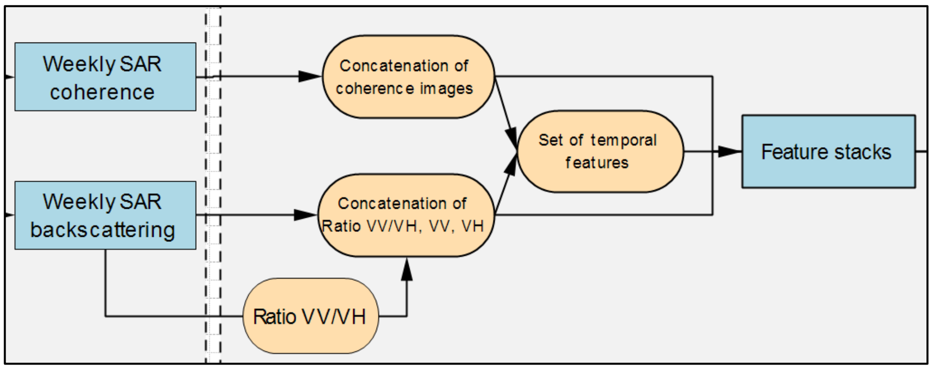

Weekly SAR coherence
Weekly SAR backscattering
Concatenation of coherence images
Concatenation of Ratio VV/VH, VV, VH
Ratio VV/VH
Set of temporal features
Feature stacks

Figure 3-3. Workflow of the SAR data feature extraction steps

3.2.1 Ratio VV/VH

For each S1 acquisition, the VV/VH ratio is computed using a generic OTB application called `otbcli_BandMath` (Algorithm 3-3).

Algorithm 3-3. OTB application to compute the S1 VV/VH ratio

```
otbcli_BandMath -il $SARimage_datei_VV $SARimage_datei_VH \
                -out $SARimage_ratioVVVH?&gdal:co:TILED=YES&gdal:co:COMPRESS=LZW \
                -exp im1b1/im2b1
```

Table 3-3. Specific variables for the computation of the SAR ratio

Input variable	Role	Default value [format]
SARimage_datei_VV	SAR backscattering for date i in VV polarization	[character]
SARimage_datei_VH	SAR backscattering for date i in VH polarization	[character]
Output variable	**Role**	**Default value [format]**
SARimage_ratioVVVH	Output ratio for date i (single band raster)	[character]

3.2.2 Feature concatenation

The complete sets of weekly SAR features (VV, VH and Ratio) are then concatenated to produce multiband rasters or vrt (Virtual Raster Table - gdal driver). Each raster corresponds to one feature time series. **10 feature time series rasters** are therefore produced:

- Backscattering – VH – Ascending
- Backscattering – VV – Descending
- Backscattering – VH – Descending

- Backscattering – VV – Ascending
- Backscattering – RATIO – Ascending
- Backscattering – RATIO – Descending
- Coherence – VV – Ascending
- Coherence – VH – Ascending
- Coherence – VV – Descending
- Coherence – VH – Descending

The concatenation step can be performed using the GDAL function `gdalbuildvrt`. The needed variables and the script are presented in Table 3-4 and in Algorithm 3-4.

Table 3-4. Specific variables for the feature concatenation of the SAR time series data

Input variable	Role	Default value [format]
N_MONTHS	The number of months from January to be monitored	[MM]
ORBIT	The orbit number of the S1 time series	[numeric]
YEAR	The year	[YYYY]
COHE_FOLDER	The folder containing the coherence data	[character]
BACK_FOLDER	The folder containing the backscattering data	[character]
RATIO_FOLDER	The folder containing the backscattering ratio data	[character]
Output variable	**Role**	**Default value [format]**
OUT_FOLDER	Output folder to write multiband rasters or vrt (Virtual Raster Table - gdal driver)	[character]

Algorithm 3-4. Feature concatenation of the SAR time series data

```
#! /bin/bash

#Update the following variables to match the build folder and modules output folder
N_MONTHS=$1
ORBIT=$2
YEAR=$3
COHE_FOLDER=$4
BACK_FOLDER=$5
RATIO_FOLDER=$6
OUT_FOLDER=$7

echo  "COHE  FOLDER  is  $COHE_FOLDER"  ;  echo  "AMPL  FOLDER  is  $AMPL_FOLDER"  ;  echo  "RATIO  FOLDER  is
$RATIO_FOLDER" ; echo "OUT FOLDER is $OUT_FOLDER"

MONTHS=(${YEAR}01  ${YEAR}02  ${YEAR}03  ${YEAR}04  ${YEAR}05  ${YEAR}06  ${YEAR}07  ${YEAR}08  ${YEAR}09
${YEAR}10 ${YEAR}11 ${YEAR}12)

echo Number of months to process for $YEAR is $N_MONTHS
echo
echo Starting writing listing the input descriptors
echo
INPUT_DESCRIPTORS_COHE_VV=$(find $COHE_FOLDER -name ${MONTHS[0]}*_20*_VV_*${orbit}*_cohe.tiff)
INPUT_DESCRIPTORS_COHE_VH=$(find $COHE_FOLDER -name ${MONTHS[0]}*_20*_VH_*${orbit}*_cohe.tiff)
INPUT_DESCRIPTORS_VV=$(find $AMPL_FOLDER -name ${MONTHS[0]}*_20*_VV_*${orbit}*_amp.tiff)
INPUT_DESCRIPTORS_VH=$(find $AMPL_FOLDER -name ${MONTHS[0]}*_20*_VH_*${orbit}*_amp.tiff)
INPUT_DESCRIPTORS_RATIO=$(find $RATIO_FOLDER -name ${MONTHS[0]}*_20*_RATIO_*${orbit}*_amp.tiff)
for ((i=1; i<${N_MONTHS}; i++)); do
```

```
echo adding ${MONTHS[i]} to list of input
INPUT_DESCRIPTORS_COHE_VV="${INPUT_DESCRIPTORS_COHE_VV} "
INPUT_DESCRIPTORS_COHE_VH="${INPUT_DESCRIPTORS_COHE_VH} "
INPUT_DESCRIPTORS_VV="${INPUT_DESCRIPTORS_VV} "
INPUT_DESCRIPTORS_VH="${INPUT_DESCRIPTORS_VH} "
INPUT_DESCRIPTORS_RATIO="${INPUT_DESCRIPTORS_RATIO} "
INPUT_DESCRIPTORS_COHE_VV+=$(find $COHE_FOLDER -name ${MONTHS[i]}*_20*_VV_*${orbit}*_cohe.tiff)
INPUT_DESCRIPTORS_COHE_VH+=$(find $COHE_FOLDER -name ${MONTHS[i]}*_20*_VH_*${orbit}*_cohe.tiff)
INPUT_DESCRIPTORS_VV+=$(find $AMPL_FOLDER -name ${MONTHS[i]}*_20*_VV_*${orbit}*_amp.tiff)
INPUT_DESCRIPTORS_VH+=$(find $AMPL_FOLDER -name ${MONTHS[i]}*_20*_VH_*${orbit}*_amp.tiff)
INPUT_DESCRIPTORS_RATIO+=$(find $RATIO_FOLDER -name ${MONTHS[i]}*_20*_RATIO_*${orbit}*_amp.tiff)
done
echo INPUT_DESCRIPTORS_COHE_VV is $INPUT_DESCRIPTORS_COHE_VV

echo Writing COHE_VV_${orbit}.vrt
gdalbuildvrt    -separate    $OUT_FOLDER/COHE_VV_${orbit}.vrt    $INPUT_DESCRIPTORS_COHE_VV    --config
GDAL_CACHEMAX 4000
echo Writing $OUT_FOLDER/SAR_featuretimeseries_COHE_VV_${orbit}_${YEAR}_${N_MONTHS}MONTHS.tif
gdalwarp                  -of                  Gtiff                  $OUT_FOLDER/COHE_VV_${orbit}.vrt
$OUT_FOLDER/SAR_featuretimeseries_COHE_VV_${orbit}_${YEAR}_${N_MONTHS}MONTHS.tif  -co  COMPRESS=LZW  --
config GDAL_CACHEMAX 8000 -dstnodata -10000 -tr 10 10 -co BIGTIFF=YES -multi -wo NUM_THREADS=ALL_CPUS

echo Writing COHE_VH_${orbit}.vrt
gdalbuildvrt    -separate    $OUT_FOLDER/COHE_VH_${orbit}.vrt    $INPUT_DESCRIPTORS_COHE_VV    --config
GDAL_CACHEMAX 4000
echo Writing $OUT_FOLDER/SAR_featuretimeseries_COHE_VH_${orbit}_${YEAR}_${N_MONTHS}MONTHS.tif
gdalwarp                  -of                  Gtiff                  $OUT_FOLDER/COHE_VH_${orbit}.vrt
$OUT_FOLDER/SAR_featuretimeseries_COHE_VH_${orbit}_${YEAR}_${N_MONTHS}MONTHS.tif  -co  COMPRESS=LZW  --
config GDAL_CACHEMAX 8000 -dstnodata -10000 -tr 10 10 -co BIGTIFF=YES -multi -wo NUM_THREADS=ALL_CPUS

echo Writing VV_${orbit}.vrt
gdalbuildvrt -separate $OUT_FOLDER/VV_${orbit}.vrt $INPUT_DESCRIPTORS_VV --config GDAL_CACHEMAX 4000
echo Writing $OUT_FOLDER/SAR_featuretimeseries_AMPL_VV_${orbit}_${YEAR}_${N_MONTHS}MONTHS.tif
gdalwarp                   -of                   Gtiff                   $OUT_FOLDER/VV_${orbit}.vrt
$OUT_FOLDER/SAR_featuretimeseries_AMPL_VV_${orbit}_${YEAR}_${N_MONTHS}MONTHS.tif  -co  COMPRESS=LZW  --
config GDAL_CACHEMAX 8000 -dstnodata -10000 -tr 10 10 -co BIGTIFF=YES -multi -wo NUM_THREADS=ALL_CPUS

echo Writing VH_${orbit}.vrt
gdalbuildvrt -separate $OUT_FOLDER/VH_${orbit}.vrt $INPUT_DESCRIPTORS_VH --config GDAL_CACHEMAX 4000
echo Writing $OUT_FOLDER/SAR_featuretimeseries_AMPL_VH_${orbit}_${YEAR}_${N_MONTHS}MONTHS.tif
gdalwarp                   -of                   Gtiff                   $OUT_FOLDER/VH_${orbit}.vrt
$OUT_FOLDER/SAR_featuretimeseries_AMPL_VH_${orbit}_${YEAR}_${N_MONTHS}MONTHS.tif  -co  COMPRESS=LZW  --
config GDAL_CACHEMAX 8000 -dstnodata -10000 -tr 10 10 -co BIGTIFF=YES -multi -wo NUM_THREADS=ALL_CPUS

echo Writing RATIO_${orbit}.vrt
gdalbuildvrt  -separate  $OUT_FOLDER/RATIO_${orbit}.vrt  $INPUT_DESCRIPTORS_RATIO  --config  GDAL_CACHEMAX
4000
echo Writing $OUT_FOLDER/SAR_featuretimeseries_AMPL_RATIO_${orbit}_${YEAR}_${N_MONTHS}MONTHS.tif
gdalwarp                  -of                  Gtiff                  $OUT_FOLDER/RATIO_${orbit}.vrt
$OUT_FOLDER/SAR_featuretimeseries_AMPL_RATIO_${orbit}_${YEAR}_${N_MONTHS}MONTHS.tif -co COMPRESS=LZW --
config GDAL_CACHEMAX 8000 -dstnodata -10000 -tr 10 10 -co BIGTIFF=YES -multi -wo NUM_THREADS=ALL_CPUS

echo "--- Cleaning temporary files ---"
rm $OUT_FOLDER/RATIO_${orbit}.vrt
rm $OUT_FOLDER/VH_${orbit}.vrt
rm $OUT_FOLDER/VV_${orbit}.vrt
rm $OUT_FOLDER/COHE*
```

3.2.3 Temporal features

This step synthetises the backscattering and coherence values for key period of the growing cycle of crops. It is performed on pixel-basis and uses four descriptive statistics which are applied over different period. The four statistical variables are the following:

- Mean;
- Standard deviation;

- Coefficient of variation;
- Value of the quantile 10, as a proxy of the minimun value.

Such temporal features allow to reduce typical noise of the SAR data while keeping a strong temporal dimension.

This temporal feature extraction step uses a python code that allow writing srun script to be easily launch with slurm. Each statistic uses a dedicated C++ code, except the coefficient of variation which is computed directly from the mean and the standard deviation.

The 2 following sections detail the statistics and period to be used respectively for the backscattering and the coherence.

3.2.3.1 Backscattering intensity temporal features

For the backscattering (VV, VH and Ratio), the temporal features consist in:

- The **mean** over iterative 2-month periods. If the whole year is considered for the classification, the key periods are: Jan-Feb / Mar-Apr / May-Jun / Jul-Aug / Sep-Oct / Nov-Dec.
- The **coefficient of variation** over iterative 2 months periods. If the whole year is considered for the classification, the key periods are: Jan-Feb / Mar-Apr / May-Jun / Jul-Aug / Sep-Oct / Nov-Dec.

For the computation of the Backscatteringitude temporal features, VV and VH polarizations were considered separately as well as the ascending and descending path.

The number of backscattering temporal features per period and per statistic measure is therefore of 6:

1. Ascending – VV
2. Ascending – VH
3. Descending – VV
4. Descending – VH
5. Ascending – Ratio
6. Descending – Ratio

3.2.3.2 Coherence temporal features

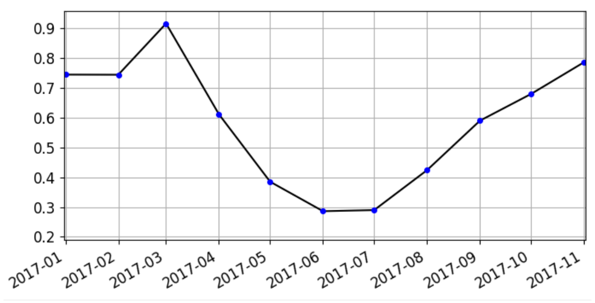
For the coherence, the temporal features consist in:

- The **standard deviation** of the coherence along the whole period of interest. If the whole year is considered for the classification, this feature corresponds to the standard deviation of the coherence of each pixel through the entire year and one temporal feature is produced.
- The value of the **quantile 10** of the coherence for each month of the period of interest, as a proxy of the minimun coherence value. If the whole year is considered for the classification, 12 temporal features are produced.
- The **mean** value of the coherence of each month of the period of interest. If the whole year is considered for the classification, 12 temporal features are produced.

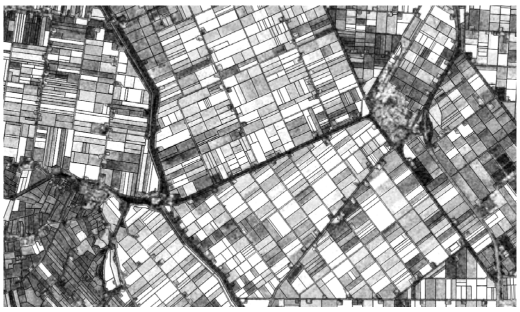
For the computation of the coherence temporal features, VV and VH coherence were considered separately while ascending and descending path were considered together.

The number of coherence temporal features per period and per statistic measure is therefore of 2:

1. Asending&Descending – VV
2. Asending&Descending – VH



(a)



(b)

Figure 3-4. (a) Exemple of monthly coherence over a Winter Wheat field in Netherlands (b) Mean of the coherence value of March 2017 over Netherlands

The calculation of the temporal features can be performed using the python script `launch_temporal_features.py`. It computes various statistical quantities over a given temporal extent. The needed input variables are presented in the Table 2-1 and the script in Algorithm 3-5.

Table 3-5. Specific variables for the temporal features computation based on the S1 time series

Input variable	Role	Default value [format]
S1_dir	S1 directory	[character]
start_date	Start date of the temporal extent	[YYYYMMDD]
end_date	End date of the temporal extent	[YYYYMMDD]
pattern	Space separated list of coma separated patterns that will be considered separately for the calculations. Multiple entries allowed. NB: only files matching at least one of the pattern will be considered. If you do not want to separate different patterns put at least one pattern to get correct tif in the s1 folders.	VV,VH asc,desc
min	Minimal value for the s1 data. Any smaller value is considered as nodata	[numeric]
max	Maximal value for the s1 data. Any higher value is considered as nodata.	[numeric]
computeMean	Compute or not the Mean over temporal extent.	[boolean] True/False
computeStd	Compute or not the standard deviation over the temporal extent.	[boolean] True/False
computeQuantile10	Compute or not the quantile 10 (more precisely the percentile 10) over the temporal extent.	[boolean] True/False
computeQuantile90	Compute or not the quantile 90 (more precisely the percentile 90) over the temporal extent.	[boolean] True/False
computeCoefVar	Compute or not the coeficient of variation over the temporal extent.	[boolean] True/False
Output variable	**Role**	**Default value [format]**

output_dir	Output folder for the created output tif.	fid [character]

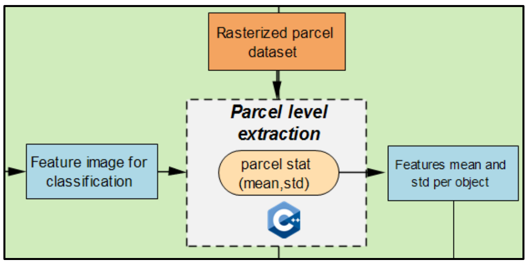
Algorithm 3-5. Temporal features computation from SAR data (launch_temporal_features.py)

```
import os
import argparse
import argparse, os, sys, subprocess
from datetime import datetime, timedelta

# Example: mkdir slurm_jobs; cd slurm_jobs;
parser = argparse.ArgumentParser(description='Compute various statistical quantities over a given
temporal extent NB: images must be in same projection and aligned.')
parser.add_argument('-i', help='Input image directory', required=True)
parser.add_argument('-o', default="./", type=str, help='Directory of the created output tif.')
parser.add_argument('-start', default='20170101', type=str, help='Start date of the temporal extent.
Format YYYYMMDD.')
parser.add_argument('-end', default='20170601', type=str, help='End date of the temporal extent. Format
YYYYMMDD.')
parser.add_argument('-p', '--patterns', nargs='*', default=['VV', 'asc', 'cohe'], help='Patterns to
select images participating in the temporal statistics calculation')
parser.add_argument('-min', default=0, type=float, help='Minimal value for the s1 data. Any smaller value
is considered as nodata.')
parser.add_argument('-max', default=100, type=float, help='Maximal value for the s1 data. Any higher
value is considered as nodata.')
parser.add_argument('-computeMean', default=False, type=bool, help='Compute or not the Mean over temporal
extent.')
parser.add_argument('-computeStd', default=False, type=bool, help='Compute or not the standard deviation
over the temporal extent.')
parser.add_argument('-computeQuantile10', default=False, type=bool, help='Compute or not the quantile 10
(more precisely the percentile 10) over the temporal extent.')
parser.add_argument('-computeQuantile90', default=False, type=bool, help='Compute or not the quantile 90
(more precisely the percentile 90) over the temporal extent.')
parser.add_argument('-computeCoefVar', default=False, type=bool, help='Compute or not the coeficient of
variation over the temporal extent.')
args = parser.parse_args()

slurmconfigtemplate = """#!/bin/bash
#SBATCH --job-name={0}
#SBATCH --output={0}.out
#SBATCH --error={0}.err
#SBATCH --partition=sen2agri

#source /home/tom/libs/bashrc
"""

# Extract file list matching the required pattern and within the temporal extent
s1tiles = [os.path.join(dirpath, filename) for dirpath, dirnames, filenames in os.walk(args.i) for
filename in filenames if (os.path.splitext(filename)[1].lower() == ".tiff" or
os.path.splitext(filename)[1].lower() == ".tif")]
print("Found %d images (.tiff or .tif, capital letters allowed) in the s1 folder."%len(s1tiles))
s1tiles_with_match = []
for tile in s1tiles:
  match = True
  for pattern in args.patterns:
    if not pattern in tile:
      match = False
  if match:
    s1tiles_with_match.append(tile)
print("Found %d images matching the patterns in the s1 folder."%len(s1tiles_with_match))

start_date = datetime.strptime(args.start, '%Y%m%d')
end_date = datetime.strptime(args.end, '%Y%m%d')
s1tiles_to_process = []
for tile in s1tiles_with_match:
  tile_time_str = os.path.basename(tile)[0:8]
  tile_time = datetime.strptime(tile_time_str, '%Y%m%d')
  if tile_time > start_date and tile_time < end_date:
    s1tiles_to_process.append(tile)
```

```
print("Found   %d   images   matching   the   provided   patterns   and   being   within   the   time
interval."%len(s1tiles_to_process))

mean_output_file = ""
std_output_file = ""
sbatch_command = ""

def write_slurm_config(config_name, stat_type = "mean"):
  global mean_output_file
  global std_output_file
  global sbatch_command
  srun_name = config_name + ".srun"
  slurm_conf_text = slurmconfigtemplate.format(config_name)
  outfile_name = config_name + ".tif"
  outfile_path = os.path.join(args.o, outfile_name)
  temp_outfile_path = os.path.join("/d8/UCL/SAR_temporalstats/scratch/", outfile_name)
  command_string = ""
  if stat_type == "mean":
    command_string = "/d38/UCL/enge/code/otb/pixel/build/MC_noround %s %s %s %s\n"%(temp_outfile_path,
args.min, args.max, " ".join(s1tiles_to_process))
    mean_output_file = outfile_path
  elif stat_type == "std":
    command_string  =  "/d38/UCL/enge/code/otb/pixel/build/STD  %s  %s  %s  %s\n"%(temp_outfile_path,
args.min, args.max, " ".join(s1tiles_to_process))
    std_output_file = outfile_path
  elif stat_type == "quant10":
    temp_vrt_path = temp_outfile_path.replace("tif","vrt")
    command_string = "gdalbuildvrt -separate %s %s\n"%(temp_vrt_path, " ".join(s1tiles_to_process))
    command_string     +=     "/d38/UCL/enge/code/otb/pixel/build/minimumQuantile_noMC     %s     %s
%s\n"%(temp_outfile_path, temp_vrt_path, temp_vrt_path)
    command_string += "rm %s\n"%(temp_vrt_path)
  elif stat_type == "quant90":
    temp_vrt_path = temp_outfile_path.replace("tif","vrt")
    command_string = "gdalbuildvrt -separate %s %s\n"%(temp_vrt_path, " ".join(s1tiles_to_process))
    command_string     +=     "/d38/UCL/enge/code/otb/pixel/build/maximumQuantile_noMC     %s     %s
%s\n"%(temp_outfile_path, temp_vrt_path, temp_vrt_path)
    command_string += "rm %s\n"%(temp_vrt_path)
  elif stat_type == "coefVar":
    command_string  =  "otbcli_BandMath  -il  %s  %s  -out  %s  -exp  'im1b1/im2b1'\n"%(std_output_file,
mean_output_file, temp_outfile_path)

  with open(srun_name, 'w') as slurmconf:
    slurmconf.write(slurm_conf_text)
    slurmconf.write(command_string)
    slurmconf.write("mv %s %s\n"%(temp_outfile_path, outfile_path))
    slurmconf.write("touch %s\n"%(outfile_path.replace(".tif",".end")))
  sbatch_command += "sbatch %s;"%srun_name

base_output_name = "SAR_temporalStat_"+args.start+"_"+args.end+"_"+"_".join(args.patterns)
if args.computeMean:
  write_slurm_config(base_output_name+"_mean", "mean")

if args.computeStd:
  write_slurm_config(base_output_name+"_std", "std")

if args.computeQuantile10:
  write_slurm_config(base_output_name+"_quant10", "quant10")

if args.computeQuantile90:
  write_slurm_config(base_output_name+"_quant90", "quant90")

if args.computeCoefVar:
  write_slurm_config(base_output_name+"_coefVar", "coefVar")

print("To launch all slurm jobs do\n\t \033[92m %s \033[0m"%sbatch_command)
```

3.2.4 Parcel level statistics extraction

This step allows extracting mean and standard deviation of each feature over each parcel. It can also extract the number of pixel (*npix*) that is covering each parcel. It crosses the information of the `DeclSTD_Format_raster` (the declaration shapefile rasterized) and each `featureTimeSeries` to create a multiband 'line raster' whose pixel index can be link to the parcel ID (Figure 3-5).



Rasterized parcel dataset
Parcel level extraction
Feature image for classification
parcel stat (mean,std)
Features mean and std per object

Figure 3-5.Workflow of the parcel level statistics extraction

It uses a compilled C++ code called `parcelStat` (the same than the one used for optical data, Algorithm 3-6). It produces as output one multi-band raster per statistic (mean and standard deviation).

Algorithm 3-6. ParcelStat algorithm

```
parcelStat $featureTimeSeries $DeclSTD_Format_raster $nfield $Object_feat_stat -1
```

Table 3-6. Specific input variables for the extraction of feature statistics by parcel from the S1 time series

Input variable	Role	Default value [format]
DeclSTD_Format_raster	S2 input descriptors (HDR files)	[character]
featureTimeSeries	Path of the feature time series	
nfield	Total number of fields	[numeric]
NoData	No data value, value of the pixels not belonging to a parcel	-1 [numeric]
Output variable	**Role**	**Default value [format]**
Object_feature	Name of the output 'line raster' (without extension) Output feature time series is a multiband raster. One raster is produced per statistics. the name of the stat is added to the output name. Output names: - *S1_featureTimeSeries*_OBJECT*_mean - *S1_featureTimeSeries*_OBJECT*_std - *S1_featureTimeSeries*_OBJECT*_npix	[character]

4. Object classification

The object classification step is performed using R software and several dedicated libraries. They will be deeply detailed and referenced so that one can choose to implement them in another scripting language.

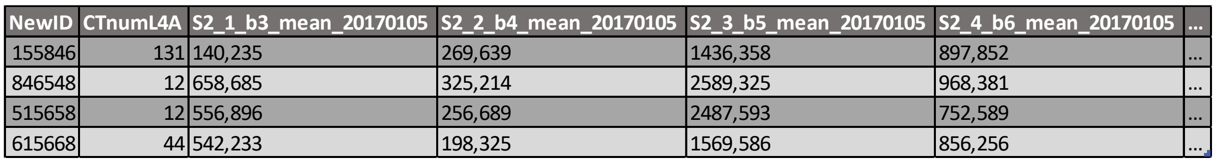
4.1 Format object feature statistics

4.1.1 Import object feature statistics: from ‘line raster’ to csv

This step extracts the statistical values (mean and std) from the raster containing the parcel level statistics (*Object_feat_stat* rasters) to include them in a csv file which make the link with the the parcel unique ID (*NewID*) and the declared crop ID (*CTnumL4A*).

It creates one csv per feature group:

- **The temporally resampled and gap-filled optical features:** gathering spectral bands + NDVI, NDWI, Brightness;
- **The weekly SAR features:** gathering "COHE_VV_asc" ,"RATIO_VVVH_asc", gathering "AMPL_VV_asc","AMPL_VH_asc","AMPL_VV_desc","AMPL_VH_desc", "COHE_VH_asc","COHE_VV_desc","COHE_VH_desc","RATIO_VVVH_desc"
- **The SAR temporal features:** gathering “VH_sc_cohe_mean", "VV_sc_cohe_std", "VH_sc_cohe_quant10","VV_sc_cohe_mean","VH_sc_cohe_std","VV_sc_cohe_quant10","VH__asc_amp_mean","VH__desc_amp_mean","VV__asc_amp_mean","VV__desc_amp_mean","VH__asc_amp_coefVar","VH__desc_amp_coefVar","VV__asc_amp_coefVar","VV__desc_amp_coefVar","VVVH_asc_RATIO_mean","VVVH_desc_RATIO_mean","VVVH_asc_RATIO_coefVar","VVVH_desc_RATIO_coefVar"



NewID	CTnumL4A	S2_1_b3_mean_20170105	S2_2_b4_mean_20170105	S2_3_b5_mean_20170105	S2_4_b6_mean_20170105	...
155846	131	140,235	269,639	1436,358	897,852	...
846548	12	658,685	325,214	2589,325	968,381	...
515658	12	556,896	256,689	2487,593	752,589	...
615668	44	542,233	198,325	1569,586	856,256	...

Figure 4-1. Example of csv containing the unique ID and the declared crop type oof the parcel along with the optical features statistic per parcel

Figure 4-1 illustrates the format of the output csv containing the optical features. Each row corresponds to a parcel:

- the first column refers to the unique ID (*NewID*);
- the second column refers to the CTnumL4A corresponding to the declared crop (*CTnumL4A*);
- the 3rd to *n* columns (*n* being the number of features * 2 (mean and std)) contains the value of the features. The names of the columns are explicit enough to be able to retrieve and analyze the relative importance of each feature in the classification results.

The two first columns are extracted from the standardized subsidy application layer.

4.1.2 Features concatenation

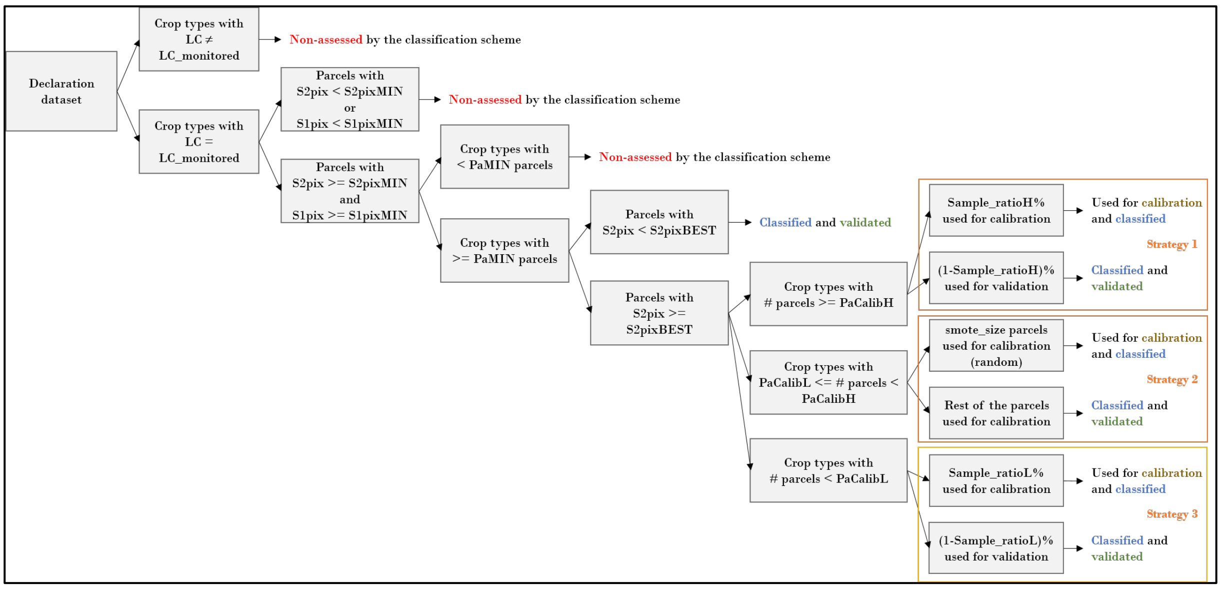
The 3 csv are then merged by doing an inner join operation on the unique parcel ID column (*NewID*) (Algorithm 4-1).

Algorithm 4-1. Features concatenation by merging CSV files

```
data_joined=inner_join(declSTD_proj_buffer_crop_conformCSV,SAR_features,by="NewID")
data_joined=inner_join(data_joined,Opt_features,by="NewID")
data_joined=inner_join(data_joined,SAR_tempStats,by="NewID")
```

4.2 Select parcels for training, classification and validation

For this selection step, different filtering operations are applied. They are presented in the next sub-sections and are summarized in Figure 4-2.



Declaration dataset
Crop types with LC ≠ LC_monitored
Non-assessed by the classification scheme
Crop types with LC = LC_monitored
Parcels with S2pix < S2pixMIN or S1pix < S1pixMIN
Non-assessed by the classification scheme
Parcels with S2pix >= S2pixMIN and S1pix >= S1pixMIN
Crop types with < PaMIN parcels
Non-assessed by the classification scheme
Crop types with >= PaMIN parcels
Parcels with S2pix < S2pixBEST
Classified and validated
Parcels with S2pix >= S2pixBEST
Crop types with # parcels >= PaCalibH
Crop types with PaCalibL <= # parcels < PaCalibH
Crop types with # parcels < PaCalibL
Sample_ratioH% used for calibration
Used for calibration and classified
Strategy 1
(1-Sample_ratioH)% used for validation
Classified and validated
smote_size parcels used for calibration (random)
Used for calibration and classified
Strategy 2
Rest of the parcels used for calibration
Classified and validated
Sample_ratioL% used for calibration
Used for calibration and classified
Strategy 3
(1-Sample_ratioL)% used for validation
Classified and validated

Figure 4-2. Specific workflow for the selection of the parcels used by the classification scheme. The by-default values are the following ones: *LC_monitored* = [1,2,3,4], *S2pixMIN* = 3, *S1pixMIN* = 1, *PaMIN* = 30, *S2pixBEST* = 10, *PaCalibH* = 4000, *PaCalibL* = 1333, *Sample_ratioH* = 0.25, *Sample_ratioL* = 0.75, *smote_size* = 1000.

The output of the selection step is presented as a single table of the same size as the input standardized subsidy application layer with quality flags, with two additional columns specifying the trajectory followed by the parcel: '*Trajectory*' and '*Purpose'*.

Trajectory column takes the values:

- '0' when the parcel is not assessed;
- '1' when the parcel is assessed.

Purpose column takes the values:

- '0' when the parcel is not assessed;
- '1' when the parcel is assessed and used for calibration;
- '2' when the parcel is assessed and used for validation.

All the parcels with the *Trajectory* = 1 are classified.

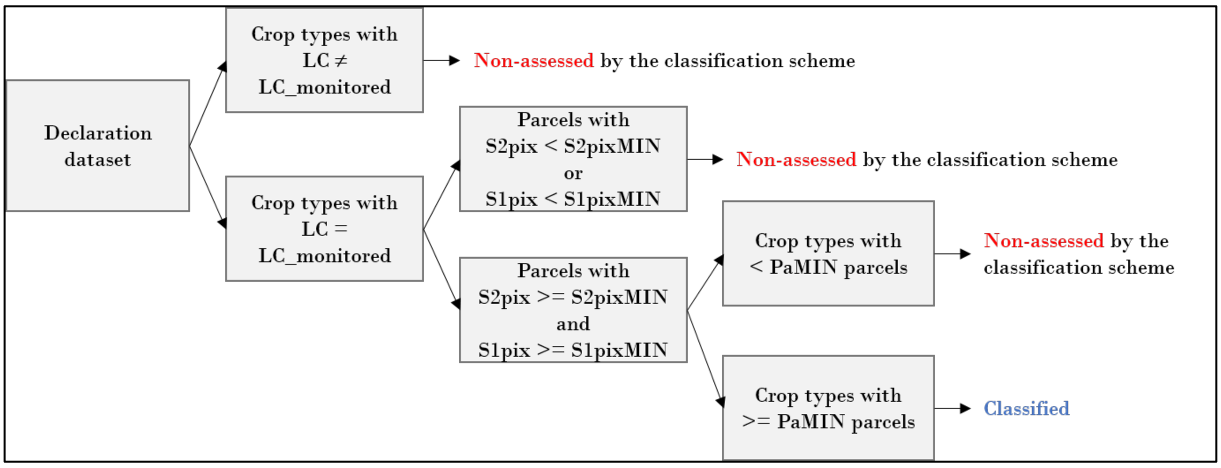
The needed input variables are presented in Table 4-1.

Table 4-1. Specific variables for the selection of parcels for classification, training and validation

Input variable	Role	Default value [format]
Declaration_dataset	Name of the PostGIS layer of the standardised subsidy application layer with quality flags	[character]
LC_monitored	List of LC classes that should be monitored	1,2,3,4 [numeric]
S2pixMIN	Minimum number of S2 pixels that contain the parcel to be assessed	3 [numeric]
S1pixMIN	Minimum number of S1 pixels that contain the parcel to be assessed	1 [numeric]
PaMIN	Minimum number of parcels with S2pix >= S2pixMIN and S1pix >= S1pixMIN by crop type for the crop type to be assessed	30 [numeric]
S2pixBEST	Minimum number of S2 pixels that contain the parcel to be used for the calibration	10 [numeric]
PaCalibH	Minimum number of parcels with S2pix >= S2pixBEST by crop type to belong to the first strategy of splitting between calibration and validation	4000 [numeric]
PaCalibL	Maximum number of parcels with S2pix >= S2pixBEST by crop type to belong to the third strategy of splitting between calibration and validation. Required: higher than smote_size.	1333 [numeric]
Sample_ratioH	Ratio of the remaining parcels by crop type to be used for the calibration, for the crop types with >= 4000 parcels with S2pix >= S2pixBEST	0.25 [numeric]
Sample_ratioL	Ratio of the remaining parcels by crop type to be used for the calibration, for the crop types with < 1100 parcels with S2pix >= S2pixBEST	0.75 [numeric]
smote_size	The number representing the desired final number of samples for each class. If the number is already reached, no synthetic samples are created. If the number is not reached, a certain amount of synthetic samples will be created to reach the smote_size (explained in section 4.3).	1000 [numeric]
Output variable	**Role**	**Default value [format]**
Trajectory	Indicates if the parcel is assessed (= 1) or not (= 0) by the classification scheme	[numeric]
Purpose	Indicates if the parcel is not assessed (= 0), is assessed and used for calibration (= 1) or is assessed and used for the validation (= 2) by the classification scheme	[numeric]

4.2.1 Non-assessed parcels

The first task consists in filtering out the parcels that are not assessable by the classification scheme (Figure 4-3). These parcels will not be included in the further steps and must be flagged as “not-assessed”.



Declaration dataset
Crop types with LC ≠ LC_monitored
Non-assessed by the classification scheme
Crop types with LC = LC_monitored
Parcels with S2pix < S2pixMIN or S1pix < S1pixMIN
Non-assessed by the classification scheme
Parcels with S2pix >= S2pixMIN and S1pix >= S1pixMIN
Crop types with < PaMIN parcels
Non-assessed by the classification scheme
Crop types with >= PaMIN parcels
Classified

Figure 4-3. Non-assessed parcels by the classification scheme. The by-default values are the following ones: *LC_monitored* = [1,2,3,4], *S2pixMIN* = 3, *S1pixMIN* = 1, *PaMIN* = 30.

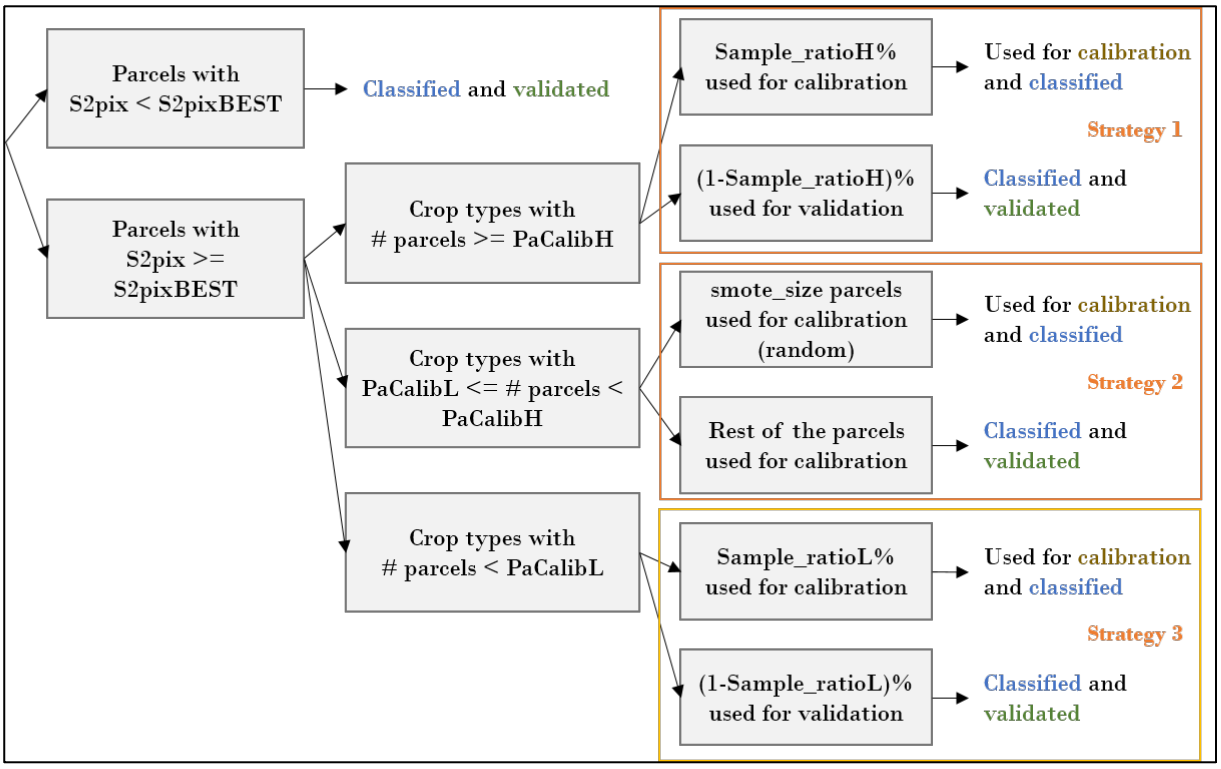
The decision about the assessibility of each parcel are taken looking at three aspects:

- **The landcover category of the crop type**: this is represented by the field LC in the standardized subsidy application layer with quality flags. By default, the LC classes 0 (other natural areas) and 5 (greenhouse and nursery) are not assessed. The remaining LC classes are 1 (annual crop), 2 (permanent crop), 3 (grassland) and 4 (fallow land) and correspond to *LC_monitored*.
- **The number of S2 pixels and S1 pixels by parcel that are used for the extraction of the spectral values**: at least one S2 pixel and one S1 pixel is needed by parcel to extract values from the the S2 and S1 time series. Given the fact that standard deviation values are calculated by parcel from the S2 time-series, the by-default *S2pixMIN* is settled to 3. Concerning the by-default *S1pixMIN*, it is settled to 1 in order to keep as many parcels as possible.
- **The number of parcels with S2pix >= S2pixMIN and S1pix >= S1pixMIN by crop type**: the crop types that have a very few parcels show a bad accuracy in the classification results. It is needed a minimum number of parcels by crop type to get enough information for the classification. The by-default *PaMIN* is settled to 30.

All the parcels that pass through these three filters are classified.

4.2.2 Parcels used for calibration and validation

Among the parcels that are classified, all the parcels are used either for the calibration of the RF model or for the validation of the results (Figure 4-4).



Parcels with S2pix < S2pixBEST
Classified and validated
Parcels with S2pix >= S2pixBEST
Crop types with # parcels >= PaCalibH
Crop types with PaCalibL <= # parcels < PaCalibH
Crop types with # parcels < PaCalibL
Sample_ratioH% used for calibration
Used for calibration and classified
Strategy 1
(1-Sample_ratioH)% used for validation
Classified and validated
smote_size parcels used for calibration (random)
Used for calibration and classified
Strategy 2
Rest of the parcels used for calibration
Classified and validated
Sample_ratioL% used for calibration
Used for calibration and classified
Strategy 3
(1-Sample_ratioL)% used for validation
Classified and validated

Figure 4-4. Parcels classified and used for calibration and validation. The by-default values are the following ones: *S2pixBEST* = 10, *PaCalibH* = 4000, *PaCalibL* = 1333, *Sample_ratioH* = 0.25, *Sample_ratioL* = 0.75, *smote_size* = 1000.

4.2.2.1 Selecting the best parcels

In order to select consistent and stable statistic values at parcel level to calibrate the RF model, only the parcels with a certain number of pixels are used for the training. The minimum number of pixels is configurable (*S2pixBEST*). The by-default value is 10 S2 pixels.

The parcels with a number of pixels below this minimum number threshold will be excluded from the calibration pool and will be part of the validation pool.

4.2.2.2 Splitting parcels for training and validation

Three "strategies" have been defined for the splitting of the parcels for training and validation, depending on the number of input parcels in the calibration pool for each crop type. It is needed to proper use the SMOTE algorithm (explained in section 4.3) which is creating in the next step synthetic samples for the crop types with a relatively low number of parcels. This step improves considerably the accuracy of the classification in these specific crop types.

- **Strategy 1**: with a large number of parcels by crop type, a relatively low ratio of parcels is used for the calibration, while the rest is used for the validation. The by-default values are settled to 4000 for the number of parcels by crop type (*PaCalibH*) and to 0.25 for the ratio between calibration and validation (*Sample_ratioH*).
- **Strategy 2**: with a medium number of parcels by crop type (but higher than the number of parcels that are reached with the SMOTE algorithm, corresponding to *smote_size*), a minimum number of parcels are kept for the calibration (equal to *smote_size*) in order to avoid to create artificial samples where the number of calibration parcels is sufficient. The slection is made

randomly by crop type. The rest of the parcels are used for the validation. The by-default values are settled to 1333 for the number of parcels by crop type (*PaCalibL*) and 1000 for the *smote_size*.

- **Strategy 3**: with a relatively low number of parcels, a relatively high ratio of parcels is used for the calibration, while the rest is used for the validation. The by-default values are settled to 1100 for the number of parcels by crop type (*PaCalibL*) and to 0.75 for the ratio between calibration and validation (*Sample_ratioL*).

4.3 Apply SMOTE algorithm to synthetically over-sample the minority classes

In order to improve to accuracy of the minor crop type classes, the Synthetic Minority Over-Sampling Technique (SMOTE) algorithm is applied to each crop type class represented by less parcels than the *smote_size* variable (by defaut, *smote_size* = 1000).

SMOTE method was introduced by Chawla N. et al (2002) and corresponds to a powerfull over-sampling method that allows avoiding over-fitting effects encountered by using more simple methods (e.g. random over-sampling of the minority class or random under-sampling of the majority class). SMOTE method creates extra training samples by analyzing the real samples in the feature space. The synthetic extra samples are added along the line segments joining any/all of the *k* nearest neighbor samples of the same class.

The pseudo code of the algorithm SMOTE can be found in RD.2 (Chawla N. et al., 2002).

The SMOTE algorithm is also available in the *smotefamily* R package (https://cran.r-project.org/web/packages/smotefamily/smotefamily.pdf) through the *SMOTE* function. Its implementation is described in Algorithm 4-2.

Algorithm 4-2. SMOTE algorithm to increase the amount of in situ data in minor crop classes

```
for each classI in the calibration_dataset
  data_calib$test = ifelse(data_calib$CTnum==classI , 1 , 0 ) # test is the column on which SMOTE is performed. Samples with test = 1 are the samples of the classI. Samples with test = 0 are all the other classes of the dataset.
  dup_size = smote_size / number of already existing sample for this class
  # dup_size is the number representing the desired times of synthetic minority instances over the original number of majority instances
   if (dup_size > 1) # the amount of original sample is lower than the smote_size. SMOTE is run.
      SMOTEd=SMOTE(data_calib[,!names(data)   %in%   c("test","CTnumL4A","Trajectory","Purpose")   , data_calib$test, K=k,   dup_size=dup_size) # keeps only the features columns and the ID.
      originals=SMOTEd$orig_P
      synthetics=SMOTEd$syn_data
      originals$SMOTE=1
      synthetics$SMOTE=0
      synthetics$CTnum=ClassI
      originals$CTnum=ClassI
      Smoted_data=rbind(Smoted_data,originals,synthetics)
   else
      originals= data_calib[which(data_calib$CTnum==ClassI,]
      colnames(originals)[which(names(originals) == "test")] <- "class"
```

```
    originals$SMOTE=1
    Smoted_data=rbind(Smoted_data,originals)
```

Table 4-2. Specific input variables for the SMOTE algorithm

Input variable	Role	Default value [format]
data_calib	The csv file with data used for the calibration (*Purpose* = 1).	[csv]
smote_size	The number representing the desired final number of samples for each class. If the number is already reached, no synthetic samples are created. If the number is not reached, a certain amount of synthetic samples will be created to reach the smote_size.	1000 [numeric]
k	The number of nearest neighbors during sampling process.	5 [numeric]
Output variable	**Role**	**Default value [format]**
Smoted_data	Matrix containing both original and smoted data.	[matrix]

4.4 Train the Random Forest model

The classifier is trained using the original and synthetic parcel feature values gathered in the new *Smoted_data* matrix and the declared crop type data (Algorithm 4-3).

Ranger is a fast implementation of original Random Forest introduced in RD.4 (Breiman, 2001).

The Random Forest algorithm is available in the *Ranger* R package (https://cran.r-project.org/web/packages/ranger/ranger.pdf) through the *ranger* function.

The *probability* option of the ranger random forest is used to grow probability forest, i.e. each tree returns a probability estimate and these estimates can be averaged to get the forest probability estimate.

Algorithm 4-3. Building the Random Forest model

```
RF_model = ranger(CTnumL4A ~ ., data = droplevels(Smoted_data), write.forest = TRUE,probability =
TRUE,num.trees = numtrees,mtry = NULL,importance = "impurity",min.node.size = 10,seed = 42)
# The random forest model can be saved as R object, which creates a serialized version of the dataset
saveRDS(Ranger_trees, ‘RF_model.rds’)
```

Table 4-3. Specific input variables for the model Random Forest

Input variable	Role	Default value [format]
Smoted_data	Matrix containing both original and smoted data with CTnum being the crop type of the parcel (= label to train the RF classifier) and all the parcel feature values.	[matrix]
numtrees	Number of trees in the random forest	300 [numeric]
min.node.size	Minimal node size.	10 [numeric]

Output variable	Role	Default value [format]
RF_model	The random forest model	[.rds file]

4.5 Classify and format the classification output table

The random forest model is then applied on all the parcels to be classified (with *Trajectory=1*) (Algorithm 4-4).

Algorithm 4-4. Classification using the Random Forest algorithm

```
## 1. Classify all the parcel applying the random forest model
predict_Ranger_trees=predict(RF_model,data_classif)
predictions=predict_Ranger_trees$predictions

## 2. Extract the most probable result of the prediction and its probability value
predict.max=apply(predictions, 1, max) # the maximum probability
predict.whichmax=apply(predictions, 1, which.max)
predict.class=colnames(predictions)[predict.whichmax]

## 3. Extract the second most probable result of the prediction and its probability value
n <- ncol(predictions)
predict.2max=apply(predictions, 1, function(x) sort(x,partial=n-1)[n-1])
predict.which2max=apply(predictions, 1, function(x) which(x==sort(x,partial=n-1)[n-1])[1])
predict.2class=colnames(predictions)[predict.which2max]
predict.2class=ifelse(predict.class==predict.2class,colnames(predictions)[apply(predictions,        1,
function(x) which(x==sort(x,partial=n-1)[n-1])[2])],predict.2class)

## 4. Format the output
Predict_classif=data.frame(data_predict$NewID,as.integer(data_predict$AREA),data_predict$TARGET,predict
.class,round(predict.max,digits=3),predict.2class,round(predict.2max,digits=3))
colnames(Predict_classif)=c('NewID','CT_decl','CT_pred_1','CT_conf_1','CT_pred_2','CT_conf_2')

## 5. Save the output
write.csv(Predict_classif,paste0(Predict_classif,".csv"))
```

Table 4-4. Specific variables for the RF model application for S2 and S1 time series classification

Input variable	Role	Default value [format]
RF_model	Random Forest model corresponding to the classification.	[.rds file]
data_classif	Table with the parcels that are classified (*Trajectory*= 1).	[csv]
Output variable	**Role**	**Default value [format]**
Predict_classif	The output classification result saved in csv.	[csv]

The first two predictions of the RF model, corresponding to the predicted crop type classes with the higher confidence levels are kept in the output *Predict_classif* file. The table contain the following fields (Table 4-5).

Table 4-5. Output attribute fields with the results of the RF classification (*Predict_classif.csv*)

Field name	Role	Default value [format]
NewID	The parcel unique ID	[integer]
CT_decl	L4A crop type code declared by the farmer	[integer]
CT_pred_1	Predicted L4A crop type code from the model with the highest degree of confidence	[integer]
CT_conf_1	Degree of confidence of CT_pred1	[float, between 0 and 1]
CT_pred_2	Predicted L4A crop type code from the model with the second highest degree of confidence	[integer]
CT_conf_2	L4A crop type code declared by the farmer	[float, between 0 and 1]

4.6 Update the subsidy application layer with the classification results

In PostGIS, the standardised subsidy application layer with quality flages is updated with the results of the RF classification (*Predict_classif.csv*).

The standardized subsidy application layer with quality flags is updated with the classification results with the following PostGIS query (Algorithm 4-5).

Algorithm 4-5. Update of the subsidy application layer with the classification results

```
update declaration_dataset
set "CT_decl" = predict_classif.ct_decl,
    "CT_pred_1" = predict_classif.ct_pred_1,
    "CT_conf_1" = predict_classif.ct_conf_1,
    "CT_pred_2" = predict_classif.ct_pred_2,
    "CT_conf_2" = predict_classif.ct_conf_2,
from predict_classif
where predict_classif.NewID = declaration_dataset.NewID;
```

5. Validation

A validation is performed directly based on the parcels that were selected for the classification and validation (*Purpose* = 2). The comparison between the declared L4A crop type and the first L4A prediction among these parcels is used to build a confusion matrix. From this matrix, the Overall Accuracy and the Kappa of the classification are calcualted, as well as the producer's and user's accuracy by crop type. These two last values are then used to calculate the F-Score of each crop type. Finally, a plot that present the results of the validation is generated.

The specific variables for the validation and the script are presented in Table 5-1 and Algorithm 5-1.

Table 5-1. Specific variables for the validation

Input variable	**Role**	**Default value [format]**
RF_model	Random Forest model corresponding to the classification.	[.rds file]
data_valid	Table with the parcels that are classified and used for the validation (*Purpose*= 2).	[csv]
Output variable	**Role**	**Default value [format]**
Accuracy_plot	Plot presenting the accuracy statistics: Overall Accuracy, Kappa and F-Score of the crop types	[jpeg]

Algorithm 5-1. Validation of the Random Forest classification

```
## 1. Extract the first prediction (with highest confidence level) of the parcels selected for the validation

predict_Ranger_trees <- readRDS(paste(workdir,predictname,".rds",sep=""))

predictions=predict_Ranger_trees$predictions

predict.max=apply(predictions, 1, max)

predict.whichmax=apply(predictions, 1, which.max)

predict.class=factor(colnames(predictions)[predict.whichmax])

data_ref=factor(data_valid_red$TARGET)

lvl = union(levels(predict.class), levels(data_ref))

predict.class = factor(predict.class, levels = lvl)

data_ref = factor(data_ref, levels = lvl)


## 2. Build the confusion matrix

Results_Ranger=confusionMatrix(predict.class,data_ref,mode="everything")

resultname=paste("Results_Ranger",Feature_type,sensor,period,landcovername,samplingmethod,sample_size,target,validation,"_smote",count_thresh,"_numtrees",numtrees,name,format(Sys.time(), "%m%d-%H%M"),sep="_")

saveRDS(Results_Ranger, paste(workdir,resultname,".rds",sep=""))

print(Results_Ranger$overall)


## 3. Calculate the accuracy statistics
```

```r
Results_ranger <- readRDS(paste(workdir,resultname,".rds",sep=""))

df                                    <-                                    data.frame(x=factor(c("Overall accuracy","Kappa",names(Results_ranger$byClass[,"F1"])),levels=(c("Overall accuracy","Kappa",names(Results_ranger$byClass[order(Results_ranger$byClass[,"Prevalence"],decreasing = TRUE),"F1"])))),y=c(Results_ranger$overall[1],Results_ranger$overall[2],Results_ranger$byClass[,"F1"]), prevalence=c(Results_ranger$overall[1],Results_ranger$overall[2],Results_ranger$byClass[,"Prevalence"]))

index=which(df$prevalence!=0)

df=df[index,]

df$x=as.character(df$x)

df$x[3:nrow(df)]=strapplyc(df$x[3:nrow(df)], "[0-9]{1,4}",simplify=TRUE)

df$area=Declarations_summary$arearatio[match(as.character(df$x), as.character(Declarations_summary$TARGET))]

df$x=factor(df$x,levels=c(df$x[1],df$x[2],df$x[3:length(df$x)][order(df$area[3:length(df$x)],decreasing =TRUE)]))

df$cumarea=0

df$cumarea[3:length(df$area)][order(df$x[3:length(df$area)])]=cumsum(df$area[3:length(df$area)][order(df$x[3:length(df$area)])])

colors=c('Overall accuracy','Kappa',rep('F-score',nrow(df)-2))


## 4. Create the plot

p<-ggplot(data=df) +

  geom_bar(aes(x=x,       y=y,fill=colors),width=.5,stat="identity",       position="dodge")       + geom_point(aes(x=x,y=cumarea),size=1)  +  scale_y_continuous(limits=c(0,1),breaks=seq(0,1,0.1),expand  = c(0, 0))

q<- p + theme_light() + theme(axis.text.x = element_text(angle = 90, hjust = 1,vjust=0.5)) + xlab("") + ylab("Performance measure or Relative cumulated area (dots)") + scale_fill_discrete("")

ggsave(paste(workdir,"Accuracy_plot",".jpg",sep=""),q,width = 13, height = 8)
```

6. Crop diversification use case

6.1 Context

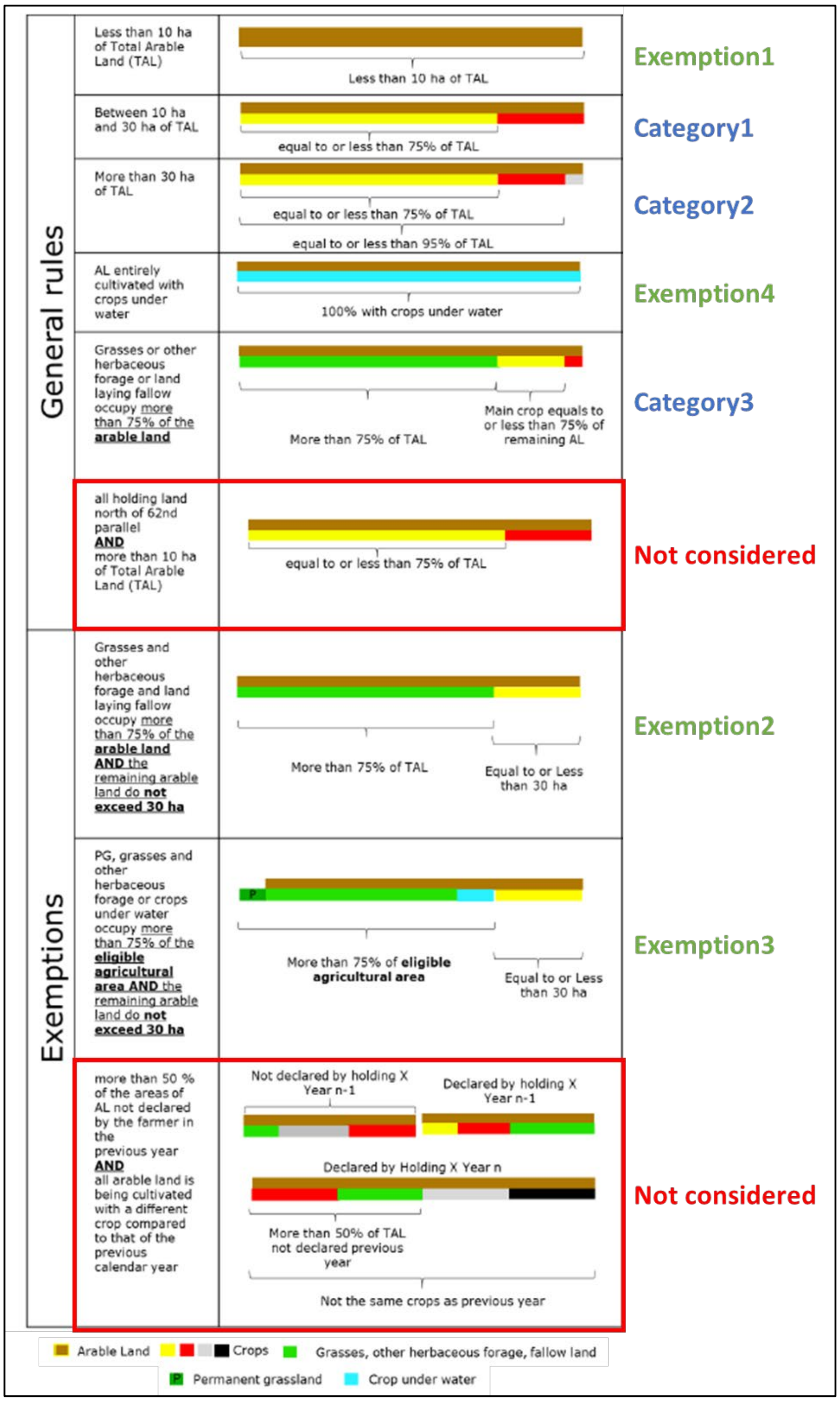
The Sen4CAP **"L4A – Crop type product"** is used to assess the compliancy of each holding regarding the crop diversification rules. A "worst case scenario" approach has been implemented to handle the small parcels and the crop types that cannot be assessed by remote sensing [RD.6].

The crop diversification monitoring approach relies on two consecutive assessments:

1) First, at the parcel-level, to verify that the crop type declared by the farmer is confirmed by the satellite signal;
2) Second, at the holding-level, to assess the compliancy with regard to the crop diversification rules.

Crop diversification regulation

Following the Technical guidance for the On-The-Spot checks of Crop Diversification [RD.7], each holding belongs to a specific crop diversification category. From the nine categories defined in the document, two have been left out either because it is a very specific case (holding land to the north of the 62nd parallel), or because it needs information from last year (which is not currently implemented). The remained seven categories considered in this use case are presented in Figure 6-1.



General rules
Less than 10 ha of Total Arable Land (TAL)
Less than 10 ha of TAL
Exemption1
Between 10 ha and 30 ha of TAL
equal to or less than 75% of TAL
Category1
More than 30 ha of TAL
equal to or less than 75% of TAL
equal to or less than 95% of TAL
Category2
AL entirely cultivated with crops under water
100% with crops under water
Exemption4
Grasses or other herbaceous forage or land laying fallow occupy more than 75% of the arable land
More than 75% of TAL
Main crop equals to or less than 75% of remaining AL
Category3
all holding land north of 62nd parallel AND more than 10 ha of Total Arable Land (TAL)
equal to or less than 75% of TAL
Not considered
Exemptions
Grasses and other herbaceous forage and land laying fallow occupy more than 75% of the arable land AND the remaining arable land do not exceed 30 ha
More than 75% of TAL
Equal to or Less than 30 ha
Exemption2
PG, grasses and other herbaceous forage or crops under water occupy more than 75% of the eligible agricultural area AND the remaining arable land do not exceed 30 ha
P
More than 75% of eligible agricultural area
Equal to or Less than 30 ha
Exemption3
more than 50 % of the areas of AL not declared by the farmer in the previous year AND all arable land is being cultivated with a different crop compared to that of the previous calendar year
Not declared by holding X Year n-1
Declared by holding X Year n-1
Declared by Holding X Year n
More than 50% of TAL not declared previous year
Not the same crops as previous year
Not considered
Arable Land
Crops
Grasses, other herbaceous forage, fallow land
Permanent grassland
Crop under water

Figure 6-1 Crop diversification regulations from the Technical guidance for the On-The-Spot checks of Crop Diversification [RD.7] considered in the Sen4CAP project and correspondence with the Sen4CAP crop diversification categories

The description of each category considered in the Sen4CAP crop diversification use case and the corresponding crop diversification rules are detailed in Table 6-1.

Table 6-1 Crop diversification categories considered in the Sen4CAP crop diversification use cases

Category	Description	Crop diversification rules
Category1	TAL between 10 and 30 ha	• At least 2 different crop types • Main crop ≤ 75% of TAL
Category2	TAL greater than 30 ha	• At least 3 different crop types • Main crop ≤ 75% of TAL • 2 main crops ≤ 95% of TAL
Category3	TGrass and Fallow greater than 75% of TAL	Main crop ≤ 75% of remaining AL
Exemption1	TAL less than 10 ha	No crop diversification required
Exemption2	TGrass and Fallow greater than 75% of TAL and remaining AL less than 30 ha	No crop diversification required
Exemption3	PGrass, TGrass and Cwater greater than 75% of EAA and remaining AL less than 30 ha	No crop diversification required
Exemption4	Cwater = TAL	No crop diversification required

TAL = Total Arable Land; AL = Arable Land; EAA = Eligible Agriculture Area; TGrass = Temporary Grassland; PGrass = Permanent Grassland; Fallow = Land Lying Fallow; Cwater = Crop Under Water

6.2 Preparation

6.2.1 Standardized subsidy application layer with quality flags and results of the classification

6.2.1.1 Crop diversification class (CTnumDIV) of the prediction 1

The possibility is given to use the prediction of the RF classification in the crop diversification assessment, when the confidence level of this prediction is above a specific threshold. To do so, and given the fact that the analysis is done on the crop diversification class and not on the L4A crop type directly, a new attribute field is added to the subsidy application layer ("CTnumDIV_pred_1"). It corresponds to the crop diversification class of the L4A crop type predicted by the RF classification with the highest level of confidence ("CT_pred_1"). It is done using the L4A crop type LUT and the following query in PostGIS (Algorithm 6-1):

Algorithm 6-1. Update of the subsidy application layer with the crop diversification code corresponding to prediction 1

```
update declaration_dataset
set "CTnumDIV_pred_1 " = L4A_lut.CTnumDIV,
from L4A_lut
where L4A_lut.CTnumL4A = declaration_dataset.CT_pred_1;
```

In this way, the user can decide to use the prediction in the crop diversification process by controlling the threshold on the confidence level (see section 6.3.1).

6.2.1.2 Export in csv

The subsidy application layer is exported in the csv format (delimiter = ‘,’), with the name {country}_{year}_DeclSTD_quality_indic_classif_results.csv. It is done with the following query in PostGIS (Algorithm 6-2):

Algorithm 6-2. Subsidy application layer export in csv

```
copy declaration_dataset to '\path\{country}_{year}_DeclSTD_quality_indic_classif_results.csv' DELIMITER
',' CSV HEADER;
```

The csv file contains the following attribute fileds (Table 6-2). The fields in grey are not used in the crop diversification process and could be removed if the file is too big.

Table 6-2. Content of the csv file exported from the subsidy application layer

Field name	Role	Default value [format]
Ori attributes	All the original attributes of the original delaration dataset	[integer, float or string]
Ori_id	The initial id from the subsidy application layer	[integer or string]
Ori_hold	The initial holding id from the subsidy application layer	[integer or string]
Ori_crop	The initial crop code name from the subsidy application layer	[integer or string]
NewID	New sequential ID of the parcel	[integer]
HoldID	New sequential ID of the holdings	[integer]
CTnum	The new croptype code	[integer]
CT	The crop type name	[string]
LC	Landcover category [optional]	[integer]
GeomValid	Identify parcels for which no polygon exists in the subsidy application layer or with a not valid geometry	[integer, binary]
Duplic	Identify parcels that have the exact same geometry as another	[integer, binary]
Area_meters	Measured parcel area using the polygon shape (m²)	[integer]
Overlap	Identify parcels which overlaps with neighbouring parcels	[integer, binary]
ShapeInd	The shape index	[float]
S1pix	Indicates the number of used S1 pixels in the parcel	[integer]
S2pix	Indicates the number of used S2 pixels in the parcel	[integer]
CTnumL4A	The new crop type code resulting of the grouping of the CTnum for the classification	[integer]
CTL4A	The crop type name associated to CTnumL4A	[string]
CTnumDIV	The crop diversification class code	[integer]
CTDIV	The crop diversification class name	[string]

EAA	Eligible agricultural area: value 1 if the crop type belongs to this category, value 0 otherwise	[integer, binary]
AL	Arable Land: value 1 if the crop type belongs to this category, value 0 otherwise	[integer, binary]
PGrass	Permanent grassland: value 1 if the crop type belongs to this category, value 0 otherwise	[integer, binary]
TGrass	Temporary grassland: value 1 if the crop type belongs to this category, value 0 otherwise	[integer, binary]
Fallow	Fallow land: value 1 if the crop type belongs to this category, value 0 otherwise	[integer, binary]
Cwater	Crop under water: value 1 if the crop type belongs to this category, value 0 otherwise	[integer, binary]
CT_decl	L4A crop type code declared by the farmer	[integer]
CT_pred_1	Predicted L4A crop type code from the model with the highest degree of confidence	[integer]
CT_conf_1	Degree of confidence of CT_pred1	[float, between 0 and 1]
CT_pred_2	Predicted L4A crop type code from the model with the second highest degree of confidence	[integer]
CT_conf_2	L4A crop type code declared by the farmer	[float, between 0 and 1]
CTnumDIV_pred_1	The crop diversification class code corresponding to the predicted L4A crop type code from the model with the highest degree of confidence	[integer]

6.2.2 Crop code LUT

The crop code LUT is the table described in the section 2.1.1.3 (p.14).

6.3 Process

6.3.1 Parameters definition

Some specific parameters are defined before the launch of the crop diversification process. Please find below the list of the input variables (Table 6-3).

Concerning the *conf_threshold* parameter, the by-default value is set at 2.0, meaning that the crop diversification class corresponding to the first prediction of the classification will not be taken into account in the process. Indeed, only specific users have demanded the possibility to use the first prediction of the classification; in this case, this parameter has to be defined between 0 and 1. If the confidence level in the first prediction of the model is above this threshold and the parcel is defined as not conform, the crop diversification class corresponding to the first prediction is used in the crop diversification process.

Table 6-3 Input variables of the crop diversification process

Input variable	Role	Default value [format]
input_csv	Path to the input csv file with the data {country}_{year}_DeclSTD_quality_indic_classif_results.csv	[character]
input_lut	Path to the L4A LUT csv file {country}_{year}_L4A_crop_code_LUT.csv	[character]
output_csv	Path to the output csv file with the crop diversification results {country}_{year}_crop_diversification_results.csv	[character]
parcel_id_field	Field in the input csv file with the data with the parcel id	'NewID' [character]
holding_id_field	Field in the input csv file with the data with the holding id	'Ori_hold' [character]
crop_divers_field_decl	Field in the input csv file with the data with the crop diversification class corresponding to the declaration of the farmer	'CTnumDIV' [character]
crop_divers_field_pred_1	Field in the input csv file with the data with the crop diversification class corresponding to the first prediction of the classification	'CTnumDIV_pred_1' [character]
area_field	Field in the input csv file with the data with the area of the parcels	'Area_meters' [character]
conf_threshold	Minimum confidence level to use the first prediction of the classification (from 0 to 1)	'2.0' [float]

6.3.2 Conformity assessment at the parcel level

A parcel is assessed as "conform" if the crop type declared by the farmer corresponds to one of the two outputs of the classification, i.e. with one of the two predictions associated with the two highest degrees of confidence. The parcel is assessed as "not conform" when the two outputs of the classification are different from the farmer declaration.

In the case of a not conform parcel, the user can decide to use the predicted crop type for the assessment at the holding level if the confidence level corresponding to the first prediction is high enough (above a defined threshold). In this case, the information is also given in the assessment at the parcel level.

A dedicated python script has been developed for the crop diversification use case. Algorithm 6-3 presents the first part of it, which includes the input data import, the definition of the parameters and the analysis until the conformity assessment at the parcel level.

Algorithm 6-3. Conformity assessment at the parcel level (crop diversification use case)

```
#!/usr/bin/python

import argparse,csv,os,sys
from collections import defaultdict

input_csv='\path\{country}_{year}_DeclSTD_quality_indic_classif_results.csv'
input_lut='\path\{country}_{year}_L4A_crop_code_LUT'
output_csv='\path\-         {country}_{year}_crop_diversification_results.csv’
parcel_id_field='NewID'
holding_id_field='ori_hold'
crop_divers_field_decl='CTnumDIV'
```

```
crop_divers_field_pred_1='CTnumDIV_pred_1'
area_field='Area_meters'
conf_threshold=2.0

aggDict = {}
cat = {}
cropdiv = {}
debug = {}

output_fields_int =
['nb_types_c','NewID','CT_decl','CT_pred_1','CT_pred_2','Ori_crop',crop_divers_field_decl,crop_divers_field_pred_1,'n
b_parcels_nc','LC','S2pix','S1pix']
output_fields_float =
[area_field,'area_eaa_c','area_tal_c','area_tempGrass_c','area_permGrass_c','area_llf_c','area_cwater_c','area_remAl_
ex2_c','area_remAl_ex3_c','area_mainCrop_c','area_2mainCrop_c','area_nc','CT_conf_1','CT_conf_2']
output_fields_str = ['Ori_id','ori_hold']
output_fields =  ['NewID','Classif_r','CD_cat','CD_diagn','Area_meters']
db_fields =
['nb_types_c','area_eaa_c','area_tal_c','area_tempGrass_c','area_permGrass_c','area_llf_c','area_cwater_c','area_remA
l_ex2_c','area_remAl_ex3_c','area_mainCrop_c','area_2mainCrop_c','nb_parcels_nc','area_nc']
agri_fields = ['ori_hold','CD_cat','CD_diagn']

##-------------------------------------------------------------------------------
## Create the lists of CTnumDIV in the different categories: Eligible Agriculture Area, Arable Land, Permanent
Grassland, Temporary Grassland, Land Lying Fallow and Crop Under Water

eaa_codes = list()
tal_codes = list()
permGrass_codes = list()
tempGrass_codes = list()
llf_codes = list()
cwater_codes = list()

with open(input_lut) as l_in:
  lut_in = csv.DictReader(l_in, delimiter=',')

  for row in lut_in:
    if row['EAA'] == '1':
      eaa_codes.append(str(row['CTnumDIV']))
    if row['AL'] == '1':
      tal_codes.append(str(row['CTnumDIV']))
    if row['PGrass'] == '1':
      permGrass_codes.append(str(row['CTnumDIV']))
    if row['TGrass'] == '1':
      tempGrass_codes.append(str(row['CTnumDIV']))
    if row['Fallow'] == '1':
      llf_codes.append(str(row['CTnumDIV']))
    if row['Cwater'] == '1':
      cwater_codes.append(str(row['CTnumDIV']))

  eaa_codes=list(set(eaa_codes))
  tal_codes=list(set(tal_codes))
  permGrass_codes=list(set(permGrass_codes))
  tempGrass_codes=list(set(tempGrass_codes))
  llf_codes=list(set(llf_codes))
  cwater_codes=list(set(cwater_codes))

##-------------------------------------------------------------------------------
## Import data file

with open(input_csv) as f_in:
  csv_in = csv.DictReader(f_in, delimiter=',')

##-------------------------------------------------------------------------------
## Create column with results of the classif "Classif_r"

  for row in csv_in:
    agri  = row[holding_id_field]
    fid  = row[parcel_id_field]
    out = dict(row);
    if agri not in aggDict:
      aggDict[agri] = {}
      debug[agri] = {}
    if row['CT_pred_1'] != 'NA' and row['CT_pred_1'] != '':
      if row['CT_decl'] == row['CT_pred_1'] or row['CT_decl'] == row['CT_pred_2']:
        out['Classif_r'] = 'Classified_conform'
      elif float(row['CT_conf_1']) >= conf_threshold:
        out['Classif_r'] = 'Classified_not_conform_prediction_used'
      else:
        out['Classif_r'] = 'Classified_not_conform'
    elif row['GeomValid'] == '0' or row['Duplic'] == '0' or row['Overlap'] == '1':
      out['Classif_r'] = 'Not_classified_geometry'
    elif row['LC'] == '' or row['LC'] == '0' or row['LC'] == '5' or row['LC'] == 'NA':
      out['Classif_r'] = 'Not_classified_land_cover'
    elif row['S2pix'] == '' or int(row['S2pix']) <= 2:
      out['Classif_r'] = 'Not_classified_minS2pix'
    elif row['S1pix'] == '' or int(row['S1pix']) == 0:
      out['Classif_r'] = 'Not_classified_noS1pix'
    else:
      out['Classif_r'] = 'Not_classified_undefined'
    aggDict[agri][fid] = out;
```

6.3.3 Summarized factors by holding

A series of factors are then summarized by holding, using the information given in the L4A LUT concerning the categories of each crop type (EAA, AL, PGrass, etc.). These summarized factors are described in Table 6-4. Only the parcels that are classified and conform (and the classified and not conform parcels with a high level of confidence in the first prediction, if it is activated by the user) are used for the calculation of the *_c factors (for "conform" parcels). The rest of the parcels are used for the calculation of the *_nc factors (for "not classified" and "not conform" parcels).

In the case of the use of classified and not conform parcels with a high level of confidence in the first prediction, the crop diversification class corresponding to the crop type predicted by the classification is used.

Table 6-4. Summarized factors by holding

Factor	Description
area_eaa_c	Classified and conform* Eligible Agriculture Area (EAA)
area_tal_c	Classified and conform* Total Arable Land (TAL) area
area_tempGrass_c	Classified and conform* Temporary Grassland (TGrass) area
area_permGrass_c	Classified and conform* Permanent Grassland (PGrass) area
area_llf_c	Classified and conform* Land Lying Fallow (Fallow) area
area_cwater_c	Classified and conform* Crop Under Water (Cwater) area
area_remAl_ex2_c	Classified and conform* remaining area of AL (in the case of exemption 2)
area_remAl_ex3_c	Classified and conform* remaining area of AL (in the case of exemption 3)
nb_types_c	Number of classified and conform* crop types in AL
area_mainCrop_c	Area of the main classified and conform* crop type in AL
area_2mainCrop_c	Area of the second main classified and conform* crop type in AL
nb_parcels_nc	Number of not classified or classified and not conform** parcels (all remaining parcels in EAA)
area_nc	Not classified parcels or classified or not conform** (all remaining parcels in EAA)

* and the prediction of the model in the case of the classified and not conform parcels with a high level of confidence in the first prediction, if it is activated by the user

** except the classified and not conform parcels with a high level of confidence in the first prediction, if it is activated by the user

Algorithm 6-4 shows the python script that generates the summarized factors by holding.

Algorithm 6-4. Summarized factors by holding (crop diversification use case)

```
##-------------------------------------------------------------------------------
## Factors summary

  for agri in aggDict:
    area_eaa_c = 0.
    area_tal_c = 0.
    area_tempGrass_c = 0.
    area_permGrass_c = 0.
    area_llf_c = 0.
    area_cwater_c = 0.
    area_remAl_ex2_c = 0.
    area_remAl_ex3_c = 0.
    types_c = set()
    nb_parcels_nc = 0.
    area_mainCrop_c = 0.
    area_2mainCrop_c = 0.
    area_nc = 0.
    areas_dict = {}

    for fid in aggDict[agri]:
      row = aggDict[agri][fid]
      if row['Classif_r'] == 'Classified_conform':
        if row[crop_divers_field_decl] == '':
          cType = str(row[crop_divers_field_decl])
        else:
          cType = str(int(row[crop_divers_field_decl]))
      elif row['Classif_r'] == 'Classified_not_conform_prediction_used':
        if row[crop_divers_field_decl] == '':
          cType = str(row[crop_divers_field_pred_1])
        else:
          cType = str(int(row[crop_divers_field_pred_1]))
      else:
        if row[crop_divers_field_decl] == '':
          cType = str(row[crop_divers_field_decl])
        else:
          cType = str(int(row[crop_divers_field_decl]))
      area = float(row[area_field])
      if row['Classif_r'] == 'Classified_conform' or row['Classif_r'] == 'Classified_not_conform_prediction_used':
        if cType in eaa_codes:
          area_eaa_c += area
        if cType in tal_codes:
          area_tal_c += area
          types_c.add(cType)
          areas_dict[cType] = area if cType not in areas_dict else area + areas_dict[cType]
        if cType in tempGrass_codes:
          area_tempGrass_c += area
        if cType in permGrass_codes:
          area_permGrass_c += area
        if cType in llf_codes:
          area_llf_c += area
        if cType in cwater_codes:
          area_cwater_c += area
      elif cType in eaa_codes:
        area_nc += area
        nb_parcels_nc += 1

    nb_types_c = len(types_c)
    area_remAl_ex2_c = area_tal_c - area_tempGrass_c - area_llf_c
    area_remAl_ex3_c = area_tal_c - area_tempGrass_c - area_cwater_c

    areas = list(areas_dict.values())
    areas.sort(reverse=True)

    area_mainCrop_c = 0. if len(areas) == 0 else areas[0]
    area_2mainCrop_c = 0. if len(areas) == 0 or len(areas) == 1 else areas[1]
```

6.3.4 Category assessment at the holding level

Using the summarized factors by holding, the first part of the assessment is to define to which crop diversification category the holding belongs. The *_c factors are used to predefine to which of these 7 categories each holding belongs. Then, "worst case scenarios" are applied to check if the *_nc factors can have an impact on the definition of the category or not. If it is not the case, the category is confirmed; it corresponds to one of the 7 defined categories. If the *_nc factors can have an impact on the definition of the holding category, "worst case scenarios" are again applied to check all the possible categories of the holding. The field "CD_cat" (for crop diversification category) gives the results of the crop diversification category assessment (Table 6-5).

Table 6-5. Crop diversification category assessment

CD_cat	Description
Exemption1	TAL less than 10 ha
Exemption2	TGrass and Fallow greater than 75% of TAL and remaining AL less than 30 ha
Exemption3	PGrass, TGrass and Cwater greater than 75% of EAA and remaining AL less than 30 ha
Exemption4	Cwater = TAL
Category1	TAL between 10 and 30 ha
Category2	TAL greater than 30 ha
Category3	TGrass and Fallow greater than 75% of TAL
Category1_or_2	Holding belongs to Category1 or Category2 (see above)
Category1_or_3	Holding belongs to Category1 or Category3 (see above)
Category2_or_3	Holding belongs to Category2 or Category3 (see above)
Category1_2_or_3	Holding belongs to Category1, Category2 or Category3 (see above)
Exemption_or_Category1	Holding belongs to at least one of the Exemption or Category1 (see above)
Exemption_or_Category2	Holding belongs to at least one of the Exemption or Category2 (see above)
Exemption_or_Category3	Holding belongs to at least one of the Exemption or Category3 (see above)
Exemption_or_Category1_or_2	Holding belongs to at least one of the Exemption or Category1 or Category2 (see above)
Exemption_or_Category1_or_3	Holding belongs to at least one of the Exemption or Category1 or Category3 (see above)
Exemption_or_Category2_or_3	Holding belongs to at least one of the Exemption or Category2 or Category3 (see above)
Exemption_or_Category1_2_or_3	Holding belongs to at least one of the Exemption or Category1 or Category2 or Category3 (see above)

Algorithm 6-5 presents the python script that performs the category assessment.

Algorithm 6-5. Crop diversification category assessment (crop diversification use case)

```
##-------------------------------------------------------------------------------
## Define the crop diversification category

    cat[agri] = 'Exemption_or_Category1_2_or_3'

    if area_nc == 0:

      if area_tal_c < 100000:
        cat[agri] = 'Exemption1'

      elif area_tempGrass_c + area_llf_c > 0.75 * area_tal_c and area_remAl_ex2_c <= 300000:
        cat[agri] = 'Exemption2'

      elif area_permGrass_c + area_tempGrass_c + area_cwater_c > 0.75 * area_eaa_c and area_remAl_ex3_c <= 300000:
        cat[agri] = 'Exemption3'

      elif area_cwater_c == area_tal_c:
        cat[agri] = 'Exemption4'

      elif area_tempGrass_c + area_llf_c > 0.75 * area_tal_c:
        cat[agri] = 'Category3'

      elif area_tal_c > 100000 and area_tal_c <= 300000:
        cat[agri] = 'Category1'

      elif area_tal_c > 300000:
        cat[agri] = 'Category2'

    elif area_nc != 0:

      if area_tal_c + area_nc < 100000:
        cat[agri] = 'Exemption1'

      elif area_tempGrass_c + area_llf_c > 0.75 * (area_tal_c + area_nc) and area_remAl_ex2_c + area_nc <= 300000:
        cat[agri] = 'Exemption2'

      elif area_permGrass_c + area_tempGrass_c + area_cwater_c > 0.75 *(area_eaa_c + area_nc) and area_remAl_ex3_c +
area_nc <= 300000:
        cat[agri] = 'Exemption3'

      elif area_tal_c < 100000 or (area_tempGrass_c + area_llf_c > 0.75 * area_tal_c and area_remAl_ex2_c <= 300000)
or (area_permGrass_c + area_tempGrass_c + area_cwater_c > 0.75 * area_eaa_c and area_remAl_ex3_c <= 300000) or
(area_cwater_c == area_tal_c):

        if area_tempGrass_c + area_llf_c > 0.75 * area_tal_c:

          if area_tempGrass_c + area_llf_c > 0.75 * (area_tal_c + area_nc):
            cat[agri] = 'Exemption_or_Category3'

          elif area_tal_c + area_nc >= 100000 and area_tal_c + area_nc < 300000:
            cat[agri] = 'Exemption_or_Category1_or_3'

          elif area_tal_c >= 300000:
            cat[agri] = 'Exemption_or_Category2_or_3'

          elif area_tal_c + area_nc >= 300000:
            cat[agri] = 'Exemption_or_Category1_2_or_3'

        elif area_tempGrass_c + area_llf_c + area_nc <= 0.75 * (area_tal_c + area_nc):

          if area_tal_c + area_nc >= 100000 and area_tal_c + area_nc < 300000:
            cat[agri] = 'Exemption_or_Category1'

          elif area_tal_c >= 300000:
            cat[agri] = 'Exemption_or_Category2'

          elif area_tal_c + area_nc >= 300000:
            cat[agri] = 'Exemption_or_Category1_or_2'

        elif area_tal_c + area_nc >= 100000 and area_tal_c + area_nc < 300000:
          cat[agri] = 'Exemption_or_Category1_or_3'

        elif area_tal_c >= 300000:
          cat[agri] = 'Exemption_or_Category2_or_3'
        elif area_tal_c + area_nc >= 300000:
          cat[agri] = 'Exemption_or_Category1_2_or_3'

      elif area_tal_c > 100000 and ((area_tempGrass_c + area_llf_c + area_nc <= 0.75 * (area_tal_c + area_nc)) or
(area_tempGrass_c + area_llf_c + area_nc > 0.75 * (area_tal_c + area_nc) and area_remAl_ex2_c > 300000) or
(area_tempGrass_c + area_llf_c > 0.75 * (area_tal_c + area_nc) and area_remAl_ex2_c + area_nc > 300000) or
(area_tempGrass_c + area_llf_c > 0.75 * area_tal_c and area_remAl_ex2_c > 300000)) and ((area_permGrass_c +
area_tempGrass_c + area_cwater_c + area_nc <= 0.75 * (area_eaa_c + area_nc)) or (area_permGrass_c + area_tempGrass_c
+ area_cwater_c > 0.75 * area_eaa_c and area_remAl_ex3_c > 300000) or (area_permGrass_c + area_tempGrass_c +
area_cwater_c + area_nc > 0.75 * (area_eaa_c + area_nc) and area_remAl_ex3_c > 300000)) and (area_cwater_c !=
area_tal_c):

        if area_tempGrass_c + area_llf_c > 0.75 * area_tal_c:

          if area_tempGrass_c + area_llf_c > 0.75 * (area_tal_c + area_nc):
            cat[agri] = 'Category3'

          elif area_tal_c + area_nc >= 100000 and area_tal_c + area_nc < 300000:
            cat[agri] = 'Category1_or_3'

          elif area_tal_c >= 300000:
            cat[agri] = 'Category2_or_3'

          elif area_tal_c + area_nc >= 300000:
            cat[agri] = 'Category1_2_or_3'

        elif area_tempGrass_c + area_llf_c + area_nc <= 0.75 * (area_tal_c + area_nc):

          if area_tal_c + area_nc >= 100000 and area_tal_c + area_nc < 300000:
            cat[agri] = 'Category1'

          elif area_tal_c >= 300000:
            cat[agri] = 'Category2'

          elif area_tal_c + area_nc >= 300000:
            cat[agri] = 'Category1_or_2'

        elif area_tal_c + area_nc >= 100000 and area_tal_c + area_nc < 300000:
          cat[agri] = 'Category1_or_3'

        elif area_tal_c >= 300000:
          cat[agri] = 'Category2_or_3'

        elif area_tal_c + area_nc >= 300000:
          cat[agri] = 'Category1_2_or_3'
```

6.3.5 Crop diversification assessment at the holding level

In the case of the exemption categories, no crop diversification is needed. For the other categories, different rules have to be respected to be compliant regarding crop diversification. Again, first the *_c factors are used to check the compliancy of the holding regarding crop diversification. Then, “worst case scenarios” are applied to check if the *_nc factors can impact or not this compliancy assessment. If it is not the case, the holding is defined as compliant or not compliant regarding crop diversification. If it is the case, there is not enough information to assess the holding compliancy regarding crop diversification. The field “CD_diagn” (for crop diversification diagnostic) gives the results of the crop diversification assessment (Table 6-6).

Table 6-6. Crop diversification compliancy assessment

CD_diagn	Description
Compliant	Holding compliant regarding crop diversification
Not_compliant	Holding not compliant regarding crop diversification
Not_required	Holding with no crop diversification required

Missing_info	Not enough information to assess the holding compliancy regarding crop diversification

When the category assessment defines different possible categories for a holding, the different corresponding rules are checked. The holding will be assessed as compliant (or not compliant) if it is compliant (or not compliant) in all possible categories. If it is not the case, not enough information are available to assess the holding compliancy regarding crop diversification.

Algorithm 6-6 shows the python script which performs the crop diversification assessment and creates the output files described in the next section.

Algorithm 6-6. Crop diversification assessment (crop diversification use case)

```
##--------------------------------------------------------------------------------
## Check crop diversification rules

    if cat[agri] == 'Exemption1' or cat[agri] == 'Exemption2' or cat[agri] == 'Exemption3' or cat[agri] == 
'Exemption4':
      cropdiv[agri] = 'Not_required'

    elif cat[agri] == 'Category1':
      if nb_types_c >= 2 and area_mainCrop_c + area_nc <= 0.75*(area_tal_c + area_nc):
        cropdiv[agri] = 'Compliant'
      elif nb_types_c + nb_parcels_nc < 2 or area_mainCrop_c > 0.75*(area_tal_c + area_nc):
        cropdiv[agri] = 'Not_compliant'
      else:
        cropdiv[agri] = 'Missing_info'

    elif cat[agri] == 'Category2':
      if nb_types_c >= 3 and area_mainCrop_c + area_nc <= 0.75*(area_tal_c + area_nc) and area_mainCrop_c + 
area_2mainCrop_c + area_nc <= 0.95 *(area_tal_c + area_nc):
        cropdiv[agri] = 'Compliant'
      elif nb_types_c + nb_parcels_nc < 3 or area_mainCrop_c > 0.75*(area_tal_c + area_nc) or area_mainCrop_c + 
area_2mainCrop_c > 0.95 *(area_tal_c + area_nc):
        cropdiv[agri] = 'Not_compliant'
      else:
        cropdiv[agri] = 'Missing_info'

    elif cat[agri] == 'Category3':
      if area_mainCrop_c + area_nc <= 0.75*(area_remAl_ex2_c + area_nc):
        cropdiv[agri] = 'Compliant'
      elif area_mainCrop_c > 0.75*(area_remAl_ex2_c + area_nc):
        cropdiv[agri] = 'Not_compliant'
      else:
        cropdiv[agri] = 'Missing_info'

    elif cat[agri] == 'Category1_or_2':
      if (nb_types_c >= 2 and area_mainCrop_c + area_nc <= 0.75*(area_tal_c + area_nc)) and (nb_types_c >= 3 and 
area_mainCrop_c + area_nc <= 0.75*(area_tal_c + area_nc) and area_mainCrop_c + area_2mainCrop_c + area_nc <= 0.95 
*(area_tal_c + area_nc)):
        cropdiv[agri] = 'Compliant'
      elif (nb_types_c + nb_parcels_nc < 2 or area_mainCrop_c > 0.75*(area_tal_c + area_nc)) and (nb_types_c + 
nb_parcels_nc < 3 or area_mainCrop_c > 0.75*(area_tal_c + area_nc) or area_mainCrop_c + area_2mainCrop_c > 0.95 
*(area_tal_c + area_nc)):
        cropdiv[agri] = 'Not_compliant'
      else:
        cropdiv[agri] = 'Missing_info'

    elif cat[agri] == 'Category1_or_3':
      if (nb_types_c >= 2 and area_mainCrop_c + area_nc <= 0.75*(area_tal_c + area_nc)) and (area_mainCrop_c + 
area_nc <= 0.75*(area_remAl_ex2_c + area_nc)):
        cropdiv[agri] = 'Compliant'
      elif (nb_types_c + nb_parcels_nc < 2 or area_mainCrop_c > 0.75*(area_tal_c + area_nc)) and (area_mainCrop_c > 
0.75*(area_remAl_ex2_c + area_nc)):
        cropdiv[agri] = 'Not_compliant'
      else:
        cropdiv[agri] = 'Missing_info'

    elif cat[agri] == 'Category2_or_3':
      if (nb_types_c >= 3 and area_mainCrop_c + area_nc <= 0.75*(area_tal_c + area_nc) and area_mainCrop_c + 
area_2mainCrop_c + area_nc <= 0.95 *(area_tal_c + area_nc)) and (area_mainCrop_c + area_nc <= 0.75*(area_remAl_ex2_c 
+ area_nc)):
        cropdiv[agri] = 'Compliant'
      elif (nb_types_c + nb_parcels_nc < 3 or area_mainCrop_c > 0.75*(area_tal_c + area_nc) or area_mainCrop_c + 
area_2mainCrop_c > 0.95 *(area_tal_c + area_nc)) and (area_mainCrop_c > 0.75*(area_remAl_ex2_c + area_nc)):
        cropdiv[agri] = 'Not_compliant'
      else:
        cropdiv[agri] = 'Missing_info'

    elif cat[agri] == 'Category1_2_or_3':
      if (nb_types_c >= 2 and area_mainCrop_c + area_nc <= 0.75*(area_tal_c + area_nc)) and (nb_types_c >= 3 and 
area_mainCrop_c + area_nc <= 0.75*(area_tal_c + area_nc) and area_mainCrop_c + area_2mainCrop_c + area_nc <= 0.95 
*(area_tal_c + area_nc)) and (area_mainCrop_c + area_nc <= 0.75*(area_remAl_ex2_c + area_nc)):
        cropdiv[agri] = 'Compliant'
      elif (nb_types_c + nb_parcels_nc < 2 or area_mainCrop_c > 0.75*(area_tal_c + area_nc)) and (nb_types_c + 
nb_parcels_nc < 3 or area_mainCrop_c > 0.75*(area_tal_c + area_nc) or area_mainCrop_c + area_2mainCrop_c > 0.95 
*(area_tal_c + area_nc)) and (area_mainCrop_c > 0.75*(area_remAl_ex2_c + area_nc)):
        cropdiv[agri] = 'Not_compliant'
      else:
        cropdiv[agri] = 'Missing_info'

    elif cat[agri] == 'Exemption_or_Category1_or_2':
      if (nb_types_c >= 2 and area_mainCrop_c + area_nc <= 0.75*(area_tal_c + area_nc)) and (nb_types_c >= 3 and 
area_mainCrop_c + area_nc <= 0.75*(area_tal_c + area_nc) and area_mainCrop_c + area_2mainCrop_c + area_nc <= 0.95 
*(area_tal_c + area_nc)):
        cropdiv[agri] = 'Compliant'
      else:
        cropdiv[agri] = 'Missing_info'

    elif cat[agri] == 'Exemption_or_Category1_or_3':
      if (nb_types_c >= 2 and area_mainCrop_c + area_nc <= 0.75*(area_tal_c + area_nc)) and (area_mainCrop_c + 
area_nc <= 0.75*(area_remAl_ex2_c + area_nc)):
        cropdiv[agri] = 'Compliant'
      else:
        cropdiv[agri] = 'Missing_info'

    elif cat[agri] == 'Exemption_or_Category2_or_3':
      if (nb_types_c >= 3 and area_mainCrop_c + area_nc <= 0.75*(area_tal_c + area_nc) and area_mainCrop_c + 
area_2mainCrop_c + area_nc <= 0.95 *(area_tal_c + area_nc)) and (area_mainCrop_c + area_nc <= 0.75*(area_remAl_ex2_c 
+ area_nc)):
        cropdiv[agri] = 'Compliant'
      else:
        cropdiv[agri] = 'Missing_info'

    elif cat[agri] == 'Exemption_or_Category1_2_or_3':
      if (nb_types_c >= 2 and area_mainCrop_c + area_nc <= 0.75*(area_tal_c + area_nc)) and (nb_types_c >= 3 and 
area_mainCrop_c + area_nc <= 0.75*(area_tal_c + area_nc) and area_mainCrop_c + area_2mainCrop_c + area_nc <= 0.95 
*(area_tal_c + area_nc)) and (area_mainCrop_c + area_nc <= 0.75*(area_remAl_ex2_c + area_nc)):
        cropdiv[agri] = 'Compliant'
      else:
        cropdiv[agri] = 'Missing_info'

    elif cat[agri] == 'Undefined':
      cropdiv[agri] = 'Missing_info'

    else:
      cropdiv[agri] = 'Missing_info'

##--------------------------------------------------------------------------------
## Debug factors

    debug[agri]['nb_types_c'] = nb_types_c
    debug[agri]['area_eaa_c'] = area_eaa_c
    debug[agri]['area_tal_c'] = area_tal_c
    debug[agri]['area_tempGrass_c'] = area_tempGrass_c
    debug[agri]['area_permGrass_c'] = area_permGrass_c
    debug[agri]['area_llf_c'] = area_llf_c
    debug[agri]['area_remAl_ex2_c'] = area_remAl_ex2_c
    debug[agri]['area_remAl_ex3_c'] = area_remAl_ex3_c
    debug[agri]['area_mainCrop_c'] = area_mainCrop_c
    debug[agri]['area_2mainCrop_c'] = area_2mainCrop_c
    debug[agri]['nb_parcels_nc'] = nb_parcels_nc
    debug[agri]['area_nc'] = area_nc
    debug[agri]['area_cwater_c'] = area_cwater_c

##--------------------------------------------------------------------------------
## Write outputs

  with open(output_csv,'w') as f_out, open(output_csv.replace('.csv','_holding.csv'),'w') as f_out_holding:
    csv_out = csv.DictWriter(f_out, fieldnames=output_fields)
    csv_out.writeheader()
    csv_out_holding = csv.DictWriter(f_out_holding, fieldnames=agri_fields + db_fields)
    csv_out_holding.writeheader()

    for agri in aggDict:
      for fid in aggDict[agri]:
        row = aggDict[agri][fid]
        newrow = {}
        for field in output_fields:
          if field in output_fields_int:
            if row[field] == '':
              newrow[field] = 0
            else:
              newrow[field] = int(float(row[field]))
          elif field in output_fields_float:
            if row[field] == '':
              newrow[field] = 0
            else:
              newrow[field] = float(row[field])
          elif field in output_fields_str:
            newrow[field] = str(row[field])
        newrow['Classif_r'] = row['Classif_r']
        newrow['CD_cat'] = cat[agri]
        newrow['CD_diagn'] = cropdiv[agri]
        csv_out.writerow(newrow)

        for field in db_fields:
          newrow[field] = debug[agri][field]

    for agri in aggDict:
      listFields = aggDict[agri]
      firstKey = aggDict[agri].keys()[0]
      row = aggDict[agri][firstKey]
      newrow = {}
      for field in agri_fields:
        if field in output_fields_int:
          if row[field] == '':
            newrow[field] = 0
          else:
            newrow[field] = int(float(row[field]))
        elif field in output_fields_float:
          if row[field] == '':
            newrow[field] = 0.0
          else:
            newrow[field] = float(row[field])
        elif field in output_fields_str:
          newrow[field] = str(row[field])
      newrow['CD_cat'] = cat[agri]
      newrow['CD_diagn'] = cropdiv[agri]
      for field in db_fields:
        newrow[field] = debug[agri][field]
      csv_out_holding.writerow(newrow)

print('done')
```

6.4 Output

Two crop diversification outputs are created:

- crop_div.csv: it contains the results by parcel;
- crop_div_holding.csv: it contains the results by holding.

The content of the two crop diversification outputs are detailed in Table 6-7 and Table 6-8.

Table 6-7. Content of the crop diversification output crop_div.csv

Output variable	Role	Default value [format]
NewID	New sequential ID of the parcel	[integer]
Classif_r	Results of the conformity assessment at the parcel level	[character]

CD_cat	Results of the category assessment at the holding level	[character]
CD_diagn	Results of the crop diversification rules assessment at the holding level	[character]

Table 6-8. Content of the crop diversification output crop_div_holding.csv

Output variable	Role	Default value [format]
Ori_hold	The initial holding id from the subsidy application layer	[integer or string]
CD_cat	Results of the category assessment at the holding level	[character]
CD_diagn	Results of the crop diversification rules assessment at the holding level	[character]
nb_types_c	Number of different crop types of AL confirmed by the classification, by holding	[integer]
area_eaa_c	Area of the EAA confirmed by the classification, by holding	[float]
area_tal_c	Area of the TAL confirmed by the classification, by holding	[float]
area_tempGrass_c	Area of the temporary grassland confirmed by the classification, by holding	[float]
area_permGrass_c	Area of the permanent grassland confirmed by the classification, by holding	[float]
area_llf_c	Area of the land lying fallow confirmed by the classification, by holding	[float]
area_cwater_c	Area of the crops under water confirmed by the classification, by holding	[float]
area_remAl_ex2_c	Area of the remaining AL in the case of exemption 2 confirmed by the classification, by holding	[float]
area_remAl_ex3_c	Area of the remaining AL in the case of exemption 3 confirmed by the classification, by holding	[float]
area_mainCrop_c	Area of the main crop confirmed by the classification, by holding	[float]
area_2mainCrop_c	Area of the second main crop confirmed by the classification, by holding	[float]
nb_parcels_nc	Number of remaining parcels not confirmed by the classification (declared as EAA)	[integer]
area_nc	Area of the remaining area not confirmed by the classification (declared as EAA)	[float]

7. Output

7.1 Results of the crop type mapping

The results are delivered in geopackage (CropType.gpkg) and shapefile (CropType.shp) formats, projected in the national projection, as well as in a csv (CropType.csv). The list of attribute fields in these outputs is detailed in Table 7-1.

Table 7-1. Content of the output shapefile export

Field name	Role	Default value [format]
Attribute fields from subsidy application layer	All the attribute fields from the standardized subsidy application layer with quality flags (detailed in Table 2-2)	[integer, float or string]
CT_decl	L4A crop type code declared by the farmer	[integer]
CT_pred_1	Predicted L4A crop type code from the model with the highest degree of confidence	[integer]
CT_conf_1	Degree of confidence of CT_pred1	[float, between 0 and 1]
CT_pred_2	Predicted L4A crop type code from the model with the second highest degree of confidence	[integer]
CT_conf_2	L4A crop type code declared by the farmer	[float, between 0 and 1]

7.2 Results of the crop diversification assessment

Two crop diversification outputs are created:

- crop_div.csv: it contains the results by parcel;
- crop_div_holding.csv: it contains the results by holding.

The content of the two crop diversification outputs are detailed in Table 7-2 and Table 7-3.

Table 7-2. Content of the crop diversification output crop_div.csv

Output variable	Role	Default value [format]
NewID	New sequential ID of the parcel	[integer]
Classif_r	Results of the conformity assessment at the parcel level	[character]
CD_cat	Results of the category assessment at the holding level	[character]
CD_diagn	Results of the crop diversification rules assessment at the holding level	[character]

Table 7-3. Content of the crop diversification output crop_div_holding.csv

Output variable	Role	Default value [format]
Ori_hold	The initial holding id from the subsidy application layer	[integer or string]
CD_cat	Results of the category assessment at the holding level	[character]
CD_diagn	Results of the crop diversification rules assessment at the holding level	[character]
nb_types_c	Number of different crop types of AL confirmed by the classification, by holding	[integer]
area_eaa_c	Area of the EAA confirmed by the classification, by holding	[float]
area_tal_c	Area of the TAL confirmed by the classification, by holding	[float]
area_tempGrass_c	Area of the temporary grassland confirmed by the classification, by holding	[float]
area_permGrass_c	Area of the permanent grassland confirmed by the classification, by holding	[float]
area_llf_c	Area of the land lying fallow confirmed by the classification, by holding	[float]
area_cwater_c	Area of the crops under water confirmed by the classification, by holding	[float]
area_remAl_ex2_c	Area of the remaining AL in the case of exemption 2 confirmed by the classification, by holding	[float]
area_remAl_ex3_c	Area of the remaining AL in the case of exemption 3 confirmed by the classification, by holding	[float]
area_mainCrop_c	Area of the main crop confirmed by the classification, by holding	[float]
area_2mainCrop_c	Area of the second main crop confirmed by the classification, by holding	[float]
nb_parcels_nc	Number of remaining parcels not confirmed by the classification (declared as EAA)	[integer]
area_nc	Area of the remaining area not confirmed by the classification (declared as EAA)	[float]

7.3 Validation results

The validation results of the classication are contained in a series of files:

- Accuracy_metrics_{processing_time}.csv: the Overall Accuracy and Kappa value of the classification;
- Confusion_matrix_{processing_time}.csv: the results of the confusion matrix, and the calculation of the producer's and user's accuracy for each classified crop type;

- Accuracy_metrics_plot_{processing_time}.csv: plot with the Overall Accuracy and Kappa value, as well as the F-score values of each classified crop type; the crop types are sorted by area;
- Confusion_producer_{processing_time}.csv: for all the classified crop types, the 3 crop types with which the parcels were the most confused, regaring producer's accuracy;
- Confusion_user_{processing_time}.csv: for all the classified crop types, the 3 crop types with which the parcels were the most confused, regaring user's accuracy.

7.4 Classification related data

A series of data are also provided related to the classification:

- Crop_types_summary_{processing_time}.csv: lists all the original crop types and reports if the crop type was classified or not, and if it is the case with which strategy the calibration dataset was defined;
- Parcels_classified_with_predictions_{processing_time}.csv: results of the classification only for the classified parcels (NewID, AreaDeclared, S1Pix, S2Pix, CTnumL4A, CTL4A, LC, CT_decl, CT_pred_1, CT_conf_1, CT_pred_2, CT_conf_2);
- Parcels_all_with_predictions_{processing_time}.csv: results of the classification for all parcels (NewID, AreaDeclared, S1Pix, S2Pix, CTnumL4A, CTL4A, LC, CT_decl, CT_pred_1, CT_conf_1, CT_pred_2, CT_conf_2);
- Data_calibration_final_before_smote_{processing_time}.csv: all the markers used for the calibration of the classification, before SMOTE;
- Data_calibration_final_after_smote_{processing_time}.csv: all the markers used for the calibration of the classification, after SMOTE;
- Data_validation_final_{processing_time}.csv: all the markers used for validation;
- Random_Forest_Model_{processing_time}.rds: Random Forest model.