Sen4CAP - Sentinels for Common Agricultural Policy

Design Justification File

ATBD for Category Change detection

Milestone	CCN2 - Milestone 2
Authors	Maxime TROIANI, Diane HEYMANS, Sophie BONTEMPS, Pierre DEFOURNY, Laurentiu NICOLA, Cosmin UDROIU
Distribution	ESA - Zoltan SZANTOI

This page is intentionally left blank

Table of recorded changes

Version	Date	Reason
V0.1	19/07/2023	Internal version
v1.0	31/03/2024	First version released to ESA and published on the website

Table of contents

List of figures

List of tables

List of algorithms

References

ID	Title
RD.1	Sen4CAP Design Definition File - ATBD for the Subsidy Application Layer Preparation, version 1.1, 30 March 2021
RD.2	Sen4CAP Design Justification File: ATBD for Markers DataBase (July 2022) version 1.0
RD.3	Sen4CAP Design Justification File: ATBD for Bare Soil detection (March 2023) version 1.1
RD.4	Sen4CAP Design Justification File: ATBD for Crop Type mapping (April 2021) version 1.3

List of acronyms

Acronym	Definition
ATBD	Algorithm Theoretical Basis Document
AL	Arable Land
BS	Bare Soil
BSI	Bare Soil Index
EAA	Eligible Agricultural Area
GSAA	GeoSpatial Aid Application
LPIS	Land Parcel Identification System
LUT	Look-Up Table
NBS	Non-Bare Soil
NDTI	Normalized Difference Tillage Index
NDVI	Normalized Difference Vegetation Index
NDWI	Normalized Difference Water Index
MDB	Marker DataBase
ROI	Region Of Interest
RF	Random Forest
S1	Sentinel-1
S2	Sentinel-2
SAR	Synthetic Aperture Radar
Sen2-Agri	Sentinel-2 for Agriculture
SWIR	Short-Wave Infrared
UTM	Universal Transvere Mercator

1. Logical model – overview of the processor

The purpose of the ‘Category Change dectection’ is to detect changes in agricultural category from one agricultural season to the next. Agricultural categories are the following:

- Arable lands (temporary grasslands & annual croplands);
- Permanent grasslands;
- Permanent croplands.

Some changes are undetectable or simply impossible. Others are equally unlikely. The processor has been developed to determine the probability of different changes based on the calculation of vegetation markers derived from the processing of Sentinel-1 and Sentinel-2 images.

Based on these markers, an analysis is performed to give an indication of the likelihood of a change occurring. A large part of this processor is intended to be adapted by the user through the use of Jupyter Notebook. A reading key is also provided at the end of the document to help the user interpret the results of the processor. As the change detection is carried out over periods of two years (n-1 and n), two sites have to be created and therefore all consecutive steps will be performed for each site independentely. Enventually, the results for both periods are concatenated.

Figure presents the general workflow of the algorithm for detection category changes which is organized into 4 main components:

1. Input data preparation:
   a. Declaration data (hereafter referred to as “subsidy application layer”);
   b. Optical data;
   c. SAR data;

2. Markers generation (for two successive periods):
   a. Bare soil markers (based on the bare soil processor);
   b. Vegetation stability, outlier consecutivenses and growing markers;

3. Markers analysis (for two successive periods);

4. Results verification based on L4A crop type classification.

The algorithm relies both S1 and S2 time series. Optical and SAR data pre-processing is done in the Sen4CAP system. From the pre-processed data, S1 and S2 signal statistics extractions are performed at the parcel-level and stored into the markers database (MDB). The S2 time series are stored into the MDB1[1] that contains therefore the basic single-date markers of S2 bands and L3B variables. S2 time series are also stored into the MDB L4A[1], in the form of 10-day resampled values. The S1 time series, on the other hand, are generated using weekly resampled images extracted at the parcel level and stored in the MDB L4A[1].

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.

[1] In the actual system v3.1

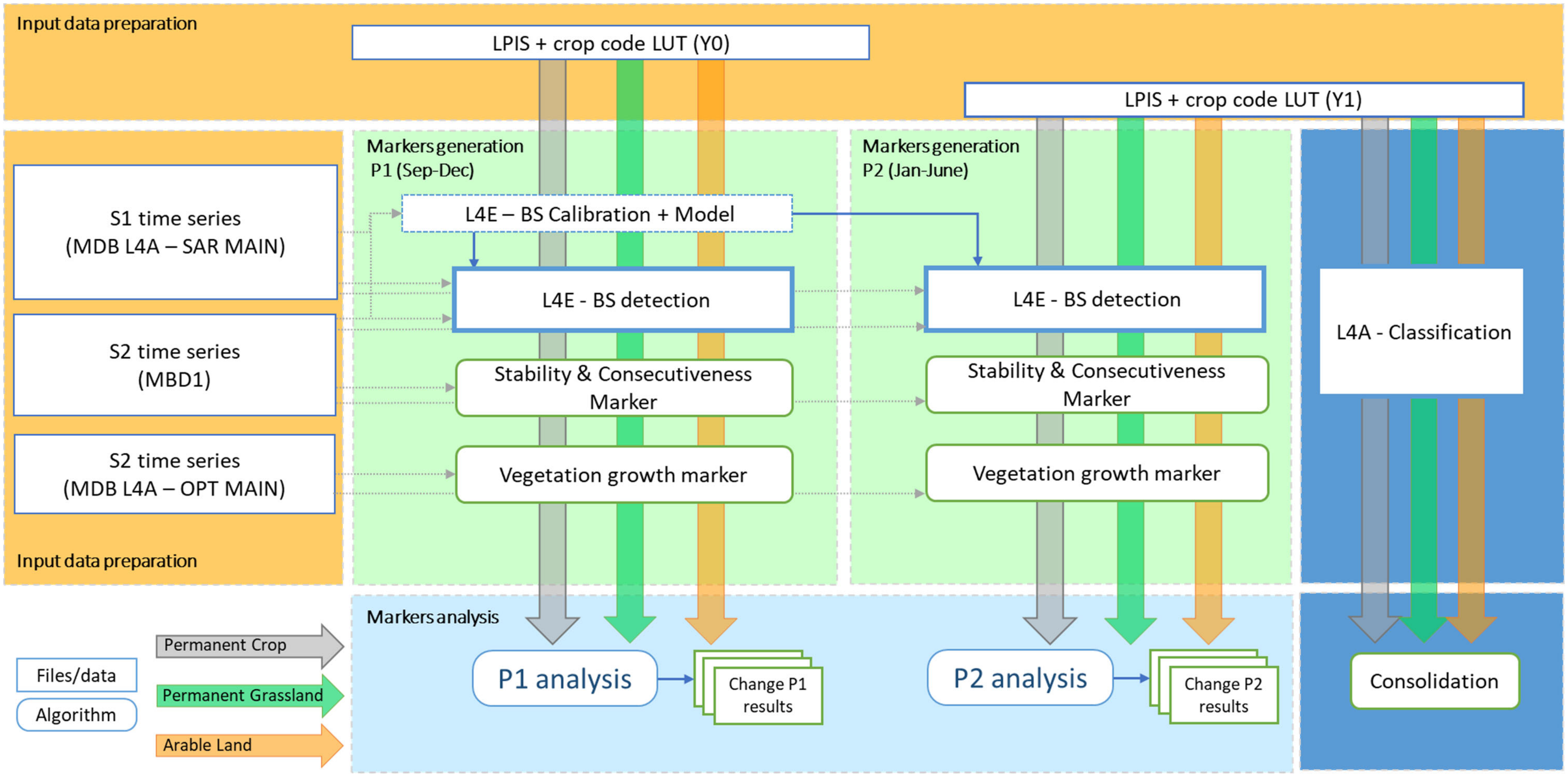


Figure 1-1. General workflow of the L4F category change algorithm

2. Input data preparation

2.1 Subsidy application layer

In order to ensure a certain level of consistency between the different Sen4CAP processors, the preparation of the subsidy application layer is performed prior to the execution of all processors. The subsidy application layer preparation is described in a dedicated ATBD (RD.1). The outputs of the subsidy application layer preparation used by the L4F category change detection processor are described below. As parcels may change in size and/or shape from a year to the next, only parcel of similar shape and size are retained for the change analysis. The script that allows this intersection is available in appendices (see Algorithm 7-1). It is used to make an inventory of the study plots and to establish the correspondence of the 'NewIDs' assigned by the system to the parcel from one year to the next.

2.1.1 Standardized subsidy application layer with quality flags

The standardized subsidy application layer with quality flags (Table 2-1):

- is stored as a PostGIS layer in the PostgreSQL database of the system;
- is projected in national projection;
- has the following name: decl_{site}_{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}_{year}	The standardized version of the subsidy application layer with the quality flags: geometry and spectral information	[GPKG] & [CSV]

It contains the attribute fields listed in Table 2-2 (fields in orange are already present in the original subsidy application layer). Attributes coming from the Look-Up Table (LUT) shown in Table 2-3 are also available in the layer at the parcel level. This layer is available as gpkg and csv files.

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]

2.1.2 Crop code LUT

If the original subsidy application layer contains 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 have a very similar phenology. It is done in the crop code LUT, which makes this grouping and defines new crop codes (CTnumL4A) and crop names (CTL4A).

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 LUT, which:

- is stored as a table in the PostgreSQL database of the system;
- is named lut_{site}_{year};
- contains the following information (Table 2-3).

These attributes are also stored at the parcel level in the standardized subsidy application (decl_{site}_{year}).

Table 2-3. 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]

2.2 Optical data from MDB1

The MDB1 contains basic single-date markers, which correspond to S1 and S2 signal statistics aggregated at the parcel-level [RD.2]. Only the following S2 signal statistics are used for this use case:

1. MDB1

- Blue B2: mean values by parcel;
- Green B3: mean values by parcel;
- NIR B8: mean values by parcel;
- SWIR1 B11: mean values by parcel;
- SWIR2 B12: mean values by parcel;
- NDVI: mean values by parcel;
- LAI: mean and standard deviation values by parcel;
- FAPAR: mean values by parcel;

FCOVER: mean values by parcel.

NDVI, LAI, FAPAR and FCover statistics are calculated for each L3B product generated from S2 time series by the Sen4CAP L3B processor.

Since the S2 single-dates bands values are extracted for each S2 tile, one parcel can have more than one value for a single-date and for single-bands (overlapping area). In those cases, only the mean of those values is kept.

These S2 signal statistics extracted will be used directly by the Category Change Detection processor or by by the Bare soil processor, which is part of it (see section 3.1). The access to this dataset in done using the API of the Sen4CAP system.

2.3 Optical and SAR data from MDB L4A

The MDB L4A contains the markers specifically used by the L4A crop type processor. They are the results of a 10-day temporal resampling in the case of S2 data and of a statistics calculation over different periods in the case of S1 data. The MDB L4A is devided into four sub-MDB [RD.2]. Only the following features are used in the category change detection processor and in the bare soil processor since it is part of the workflow:

- Optical S2 signal statistics: every 10 days, based on 10-meters resolution data:
  - NDVI: mean values by parcels.
- SAR S1 signal statistics: every week (7 days), based on 20-meters resolution data:
  - Mean Amplitude Ascending VV: mean values by parcel;
  - Mean Amplitude Ascending VH: mean values by parcel;
  - Mean Amplitude Descending VV: mean values by parcel;
  - Mean Amplitude Descending VH: mean by parcel;
  - Mean Amplitude VV/VH ratio: mean by parcel;
  - Mean Coherence Ascending VV: mean by parcel;
  - Mean Coherence Ascending VH: mean values by parcel;
  - Mean Coherence Descending VV: mean values by parcel;
  - Mean Coherence Descending VH: mean values by parcel.

These L4A S1 and S2 signal statistics extracted will be used directly by the Category Change Detection processor or by by the Bare soil processor, which is part of it (see section 3.1).

The access to this dataset in done using the API of the Sen4CAP system.

3. Markers generation

The rationale behind this module is to generate markers during two successive periods that will help detecting a change between agricultural category. The two successive periods, hereafter referred to as P1 and P2, are defined by default as: from the 1st of September to the 31st of December (01-09/31-12) for P1 and from the 1st of January to the 30th of May (01-01/30-05) for P2. These two periods are set as a default parameter. This can be adjusted in advanced parameters.

Fourkinds of markers are generated:

- Vegetation growth
- Bare soil markers (based on the L4E Bare Soil processor);
- Vegetation stability and;
- Outlier consecutiveness;

The way these markers are computed is described in the sub-sections below. The first step of the markers' generation consists in defining the site, the year and the IP address of the machine containing all the needed dataset and loading the different libraries. The standardized subsidy application layer is also imported.

The codes developed for this processor rely on a set of python libraries that need to be imported first (Algorithm 3-1).

Algorithm 3-1. Importing necessary python libraries

```
from osgeo import ogr, osr
import psycopg2
import datetime
from datetime import date, datetime, time, timedelta
from time import strftime
from json import JSONDecodeError
import requests
import random
import pandas as pd
import numpy as np
from tqdm import tqdm
```

Algorithm 3-2. Defining key variables to access the machine and specific site

```
ip_sen4cap = '000.11.222.33'
site = 'bel_2021'
year = '2021'
decl = pd.read_csv(file_path)
```

Once the key variables are set to access the machine and a specific site, a quick API verification is made (see Algorithm 3-3 & Algorithm 3-4) by requesting all the markers in the MDB 4 – Opt main.

Algorithm 3-3. Defining needed keys and variables to access the Sen4CAP machine API

```
payload = {"user": "sen4cap", "password": "sen4cap"}
result1 = requests.post(f'http://'+ip_sen4cap+':8080/login', data=payload).json()
api_token = result1["data"]["sessionToken"]
headers = {"X-Auth-Token": api_token}
```

Algorithm 3-4. Listing available S2 markers from MDB L4A

```
markers_all =
requests.get(f'http://'+ip_sen4cap+':8080/markers/names?site='+site+'&productType=s4c_mdb
_l4a_opt_main&year='+year, headers=headers).json()["data"]
print(markers_all)
```

3.1 Bare soil markers

A first set of markers used to detect the change between land cover categories are the bare soil markers generated by the L4E Bare Soil Processor. When launching the ‘Category change processor’, the bare soil processor will be launched automatically for the specific periods of analysis P1 and P2 (described above). Even if the user has already run the L4E Bare Soil Processor for his own monitoring, the probablilty of having the bare soil markers matching perfectly with the two periods P1 and P2 is too weak.

The complete description of the bare soil processor is described in [R.D.3]. The parameters used to in the bare soil processor are described in Table 3-1.

Table 3-1. Parameters of the Bare Soil Processor to be run in the Category Change Detection Processor

Parameters	Role	Default value [format]
Calibration period	Period on which the calibration dataset is built	01/09 - 31/12 for P1 & 01/01 – 30/06 for P2 [date]
Monitoring period	Period on which bare soil/ non-bare soil analysis is performed	01/09 - 31/12 for P1 & 01/01 – 30/06 for P2 [date]
Sentinel-2 minimun pixels	Minium number of S2 pixels in the parcel to allow a parcel being used in the calibration dataset.	50 [int]
Bare soil features	List of features to be tested to build the calibration dataset of “bare soil”. Only NDVI is mandatory.	‘NDVI’, ’NDWI’, ’NDTI’, ’BSI’ [string]
Non-bare soil features	List of features to be tested to build the calibration dataset of “non-bare soil”. Only NDVI is mandatory.	‘NDVI’, ’NDWI’, ’NDTI’, FCOVER’, ’BSI’ [string]
Treshold values for bare soil features	Features treshold values set to predict bare soil.	NDVI < 0.15 NDWI < 0 NDTI < 0.1 BSI > 0.15 [Float]

Treshold values for bare soil features	Features treshold values set to predict non-bare soil	NDVI > 0.45 NDWI > 0.3 NDTI > 0.25 FCOVER > 0.45 BSI < 0 [Float]
N estimators	Number of trees in the random forest.	30 [int]
P long	Stands for "Long Period". It is used as the duration to look for vegetation after the end of the bare soil period.	60 [days]
P short	Stands for "Short Period". It corresponds to the maximum number of days after the start of the bare soil period for which the algorithm will look for other bare soil detection or the end of the bare soil period. If no bare soil or non-bare soil occurred during that period, the End of the bare soil period is set as the Sart of the period.	30 [days]
Treshold values BS	Threshold of "bare soil" (BS) that indicates the minimum confidence level in the **BS** prediction (coming from the RF algorithm) to consider this detection as a **strong.**	S2: 0.8 S1: 0.65 [float]
Treshold values NBS	Threshold of "non-bare soil" (NBS) that indicates the minimum confidence level in the **NBS** prediction (coming from the RF algorithm) to consider this detection as a **strong.**	S2: 0.8 S1: 0.7 [float]

Among all 'Bare soil processor' outputs, only 'TTdaysS2' is used as markers in this processor [R.D.3]:

- 'TTdaysS2' stands for the number of days of bare soil observed with Sentinel-2 (i.e. the sum of all bare soil periods observed in the parcel);

This marker is contained in the csv file called: **L4E_BS_MarkersAll_{site}_{Period_M}.csv**. This table has to be opened at the beginning of the process. Then, the marker is extracted at the same time as the vegetation growth marker. The code is presented and described in the following sub section.

3.2 Vegetation growth marker

A second type of marker used to detect change between land cover category consists in the characterization of the vegetation growth. This marker is built by computing the area under the NDVI curve of each parcel during the two periods of analysis. It gives an idea of the growth dynamic of each crop type which helps to distinguish them.

The function created to compute this marker is described in the Algorithm 3-5. This function approximates the area under a curve defined by a list of values using a series of trapezoids. It calculates the area of each trapezoid formed by adjacent values in the list and accumulates these areas to determine the total approximate area under the curve. The function uses a combination of rectangles and triangles to estimate the areas of these trapezoids.

Algorithm 3-5. Function to compute the area under the curve

```
def area_curve(v_m):
    total_area = 0
    for j in range(len(v_m)-1):
        area_rec = v_m[j]*10
        area_j = area_rec + (v_m[j+1]-v_m[j]) * 0.5 * 10
        total_area+=area_j
    return total_area
```

Once the function has been defined, the following script can be used to extract both the vegetation growth and the bare soil markers (Algorithm 3-6). The inputs and ouputs of the script are presented below (Table 3-2).

Table 3-2. Inputs for the extraction of the vegetation growth and bare soil markers algorithm

Input names	Role	Default value [format]
marker	Marker used to compute the area under the curve.	's2_mean_ndvi' [string]
product_type	Refers to the marker database from which the marker will be extracted.	's4c_mdb_l4a_opt_main' [string]
fromdate_Curve	The date that defines the beginning of the period of interest	'yyyy-10-15' for P1 [string] (yyyy-mm-dd)
todate_Curve	List of features to be tested to build the calibration dataset of "bare soil". Only NDVI is mandatory.	'yyyy-11-30' [string] (yyyy-mm-dd)
Outputs names	**Role**	**Default value [format]**
veg_all	Dataframe containing the the outputs of the algorithm. The different columns are: • NewID: the ID of the parcel; • S2pix: the number of S2 pixels in the parcel; • LC: the agricultural class • Pgrass: indicates wether the parcel is a permanent grassland; • CTnum: the crop type number of the parcel • CTL4A: crop type of the parcel • TTdays: number of days with bare soil (Sentinel 2) • AreaVeg: area under the cuve	• NewID: [int] • S2pix: [int] • LC: [int] • Pgrass: [int] • CTnum: [int] • CTL4A: [str] • TTdaysS2: [int] • AreaVeg: [int]

Algorithm 3-6. Vegetation growth and Bare soil markers extraction

```
decl_csv.set_index('NewID', inplace=True)

val_all = pd.DataFrame()
marker = 's2_mean_ndvi'
veg_all = []

product_type = 's4c_mdb_l4a_opt_main'

# Setting the specific period to compute the vegetation growth marker
fromdate_Curve = '2021-10-15'
todate_Curve = '2021-11-30'

url =
f"http://{ip_sen4cap}:8080/markers?site={site}&productType={product_type}&year={year}&mar
kers={marker}&from={fromdate_Curve}&to={todate_Curve}"
print(url)

re = requests.get(url, headers=headers)
#print(re.json())
result = re.json()["data"]
parcels = result['parcels']
tt = len(parcels)

for p in tqdm(range(1,tt,2000)):
    re_p = parcels[p:p+2000]
    vall_all1 = pd.DataFrame()
    for i in re_p:
        newid = i['id']
        #print(i['id'])
        val_i = pd.DataFrame()
        val_i['dates'] = result["dates"]
        val_i[marker] = i['markers'][marker]

        tt_bs = bs_markers.loc[bs_markers['NewID']==newid]
        #print(tt_bs)
        if len(tt_bs) == 0:
            bs_count = np.nan
            count_P1 = np.nan
        else :
            bs_count = tt_bs['TTdaysS2'].iloc[:].squeeze()
            count_P1 = tt_bs['NbrTTS2'].iloc[:].squeeze()
        curve_i = area_curve(val_i[marker])

        if newid in decl.index:
            veg_all.append({
            'NewID' : newid,
            'S2Pix' : decl.S2Pix[newid],
            'LC' : decl.lc[newid],
            'PGrass' : decl.pgrass[newid],
            'CTnum' : decl.ctnum[newid],
            'CTL4A' : decl.ctl4a[newid],
            'TTdaysS2': bs_count,
            'AreaVeg' : curve_i
        })
veg_all = pd.DataFrame(veg_all)
```

First, an empty pandas DataFrame named `val_all` is initialized, which will be used to store data later. The marker to be extracted via Sen4CAP API is set as `s2_mean_ndvi`. An empty list named `veg_all` is created. This list is intended to accumulate processed data for further analysis. The variable `product_type` is defined as `'s4c_mdb_l4a_opt_main'. Two date variables, `fromdate_Curve` and `todate_Curve`, are set to `'2021-10-15'` and `'2021-11-30'` respectively. These dates define the time period for the computation of the vegetation growth marker. This period has been set after a sensititvity analysis. It appeared that this period was the more appropriate to underline differences between agricultural categories. A URL is constructed using various variables such as `ip_sen4cap`, `site`, `product_type`, `year`, `marker`, `fromdate_Curve`, and `todate_Curve`. This URL is designed to fetch data related to specific markers from the Sen4CAP platform.

An HTTP GET request is made to the constructed URL using the `requests` library, and the response is stored in the variable `re`. The JSON response is parsed, and the `"data"` field is extracted into the variable `result`. This data contains marker-related information. From the `result` data, the list of parcels is extracted and stored in the variable `parcels`. The length of the `parcels` list is calculated and stored in the variable `tt`. A loop is initiated to iterate through a range of indices. Within this loop:

- The `parcels` list is sliced into chunks of 2000 elements based on the current index `p`, and this sliced data is stored in the variable `re_p`.
- An empty pandas DataFrame named `vall_all1` is initialized.
- Another loop iterates through each parcel in the `re_p` list. Within this loop:
    1. The `id` field of the current parcel is extracted and stored in the variable `newid`.
    2. Additional data related to the current parcel is retrieved from a DataFrame named `bs_markers`.
    3. An area value is calculated using the `area_curve` function for the specific marker value.
    4. It's checked whether `newid` exists in the `decl_csv` DataFrame index. If it does, a dictionary containing relevant data is appended to the `veg_all` list.

Finally, the accumulated data in the `veg_all` list is converted into a pandas DataFrame named `veg_all`.

3.3 Vegetation stability and outlier consecutiveness

The third type of marker aims at assess the physiological behavior of a parcel by comparing it to its previous crop type belonging. To do so, two markers have been developed. The first one assesses the LAI behavior of the parcel during the period of monitoring by comparing its mean value to the mean LAI value of its previous crop type. If the crop type of the parcel has changed from the previous season (i.e., temporary grassland to winter wheat) its mean LAI value will diverge from the reference mean at a certain point. The second marker assesses the consecutiveness of the out-of-range observed values.

The two markers (stability and consecutiveness) are based on the mean and the standard deviation LAI of each main cultural class during the periods of analysis P1 and P2. The first step in the elaboration of these two markers consists in building a reference dataframe containing the mean and the standard deviation for each cultural class per acquisition. This dataframe will then be used to compare the LAI value of each parcel to its reference mean and standard deviation. Table 3-3 describes the function’s inputs and output and Algorithm 3-7 presents the function itself.

Table 3-3. Inputs and outputs of the LAI reference dataframe construction function

Input names	Role	Default value [format]
marker	Marker used for both vegetation stability and outlier consecutiveness markers computation	'mean_LAI' [string]
product_type	Refers to the marker database from which the marker will be extracted	's4c_mdb1' [string]
fromdate	The date that defines the beginning of the period of interest	'yyyy-09-01' for P1 [string] (yyyy-mm-dd)
todate	List of features to be tested to build the calibration dataset of "bare soil". Only NDVI is mandatory.	'yyyy-12-31' [string] (yyyy-mm-dd)
Outputs names	**Role**	**Default value [format]**
val_all	Dataframe containing LAI values for each acquisition date and for each parcel. The different columns are: • NewID: the ID of the parcel; • Date: acquisition date; • Marker: marker's value;	• NewID: [int] • Date: [date] • Marker: [int]

Algorithm 3-7. Mean LAI reference dataframe construction

```
val_all = pd.DataFrame()
marker = 'mean_LAI'

product_type = 's4c_mdb1'

fromdate = '2021-'+'09'+'-01'
todate = '2021-'+'12'+'-31'

url =
f"http://{ip_sen4cap}:8080/markers?site={site}&productType={product_type}&year={year}&mar
kers={marker}&from={fromdate}&to={todate}"
print(url)

re = requests.get(url, headers=headers)
result = re.json()["data"]
parcels = result['parcels']
tt = len(parcels)

for p in tqdm(range(0,tt,2000)):
    re_p = parcels[p:p+2000]
    vall_all1 = pd.DataFrame()
    for i in re_p:
        newid = i['id']
        val_i = pd.DataFrame()
        val_i['dates'] = result["dates"]
        val_i[marker] = i['markers'][marker]
        val_i['NewID'] = i['id']
        vall_all1 = pd.concat([vall_all1,val_i])
    val_all = pd.concat([val_all,vall_all1])
```

The script above is used to generate a first dataframe, containing the mean_LAI per parcel and for each S2 acquisition. It could be separated in several setps:

An empty pandas DataFrame named `val_all` is created, which will be used to store collected data. The variable `marker` is assigned the value `"mean_LAI"` since it is the metric on which the two markers are based on. The variable `product_type` is set to `"s4c_mdb1"`, indicating the marker database from which the mean_LAI will be retrieved from. Two date variables, `fromdate` and `todate`, are defined with specific date values. These variables specify the time period for which data will be retrieved.

A URL is constructed using various variables either created at the beginning of the script (`ip_sen4cap`, `site`, 'year') or specifically related to markers (`product_type`, `marker`, `fromdate`, and `todate`). This URL is designed to fetch data related to a specific marker type within the given time frame. An HTTP GET request is made to the constructed URL using the `requests` library, and the response is stored in the variable `re`. The JSON response is parsed, and the `"data"` field is extracted into the variable `result`. This data contains information related to the marker type and associated values. From the `result` data, the list of parcels is extracted and stored in the variable `parcels`. Then, the length of the `parcels` list is calculated and stored in the variable `tt`. A loop is initiated to iterate through a range of indices, starting from 0 up to the length of `parcels`, with a step size of 2000. Inside the loop:

- The `parcels` list is sliced into chunks of 2000 elements based on the current index `p`, and this sliced data is stored in the variable `re_p`;
- An empty pandas DataFrame named `vall_all1` is initialized to collect data within the current iteration;
- Another loop iterates through each parcel in the `re_p` list:
  - The `id` of the current parcel is extracted and stored in the variable `newid`;
  - A pandas DataFrame named `val_i` is created. This DataFrame includes date values and the specific marker value associated with the current parcel;
  - The `id` of the current parcel is added as a new column in the `val_i` DataFrame;
  - The `val_i` DataFrame is concatenated with the existing `vall_all1` DataFrame.

After the loop, the `vall_all1` DataFrame is concatenated with the `val_all` DataFrame to accumulate all the processed data.

Subsequently, a dictionnay of dataframes is created (Algorithm 3-8).

Overall, this code processes data by grouping and aggregating it based on the 'ctnuml4a' and 'dates' columns, calculating mean and standard deviation values, and then organizing this information into a dictionary where each 'ctnuml4a' value corresponds to a DataFrame containing 'dates', 'mean', and 'std' columns.

Algorithm 3-8. Reference dataframes dictionnary creation

```
dict_ref = {}
ctnuml4a = decl_csv.ctnuml4a.unique()

for ct in ctnuml4a:

    val_plot = val_all21.loc[val_all21['ctnuml4a']==ct]
    val_dates_ct = val_plot.groupby(['dates'],as_index=False)[marker].mean()
    val_dates_ct['std'] = 
val_plot.groupby(['dates'],as_index=False)[marker].std()[marker]
    df_ref = val_dates_ct.loc[:,('dates',marker,'std')]
    df_ref = df_ref.rename(columns={marker:'mean','std':'std'})
    dict_ref[ct] = df_ref
```

On the other hand, a function is created to compare the values of each parcel to the reference dataframe. The function below (Algorithm 3-9) has been designed to make this comparison and to analyze the number of outlier (number, consecutiveness, ratio, etc.).

Algorithm 3-9. LAI outliers consecutivensess function

```
def count_lai_outliers(df, df_ref, nb_stdev):

    df['ref_LAI - stdev'] = df_ref['mean'] - nb_stdev * df_ref['std']
    df['ref_LAI + stdev'] = df_ref['mean'] + nb_stdev * df_ref['std']

    # Create a new DataFrame to store the counts
    counts_df = pd.DataFrame(columns=["count","consec_count", "total_non_nan"])

    # Loop over each parcel in the DataFrame
        # Get the LAI values for the current parcel
    parcel_lai = df
    #print(parcel_lai)
    # Calculate the reference range for each date
    ref_lai_min = df['ref_LAI - stdev']
    ref_lai_max = df['ref_LAI + stdev']
    #print(ref_lai_max)
    # Calculate the counts for the current parcel
    count = 0
    consec_count = 0
    prev_out_of_range = False
    total_non_nan = 0
    for i in range(len(parcel_lai)):
        #print('i:',i)
        if pd.isna(parcel_lai['mean'][i]):
            continue
        elif (parcel_lai['mean'][i] < ref_lai_min[i]) or (parcel_lai['mean'][i] >
ref_lai_max[i]):
            count += 1
            if prev_out_of_range:
                consec_count += 1
            else:
                prev_out_of_range = True
        else:
            prev_out_of_range = False

        total_non_nan = parcel_lai['mean'].count()

        # Append the counts to the counts DataFrame
    parcel_counts_df = pd.DataFrame({ "count": [count],
                                    "consec_count": [consec_count],
                                    "total_non_nan": [total_non_nan],})
    counts_df = pd.concat([counts_df, parcel_counts_df], ignore_index=True)
    id_not_nan = counts_df['total_non_nan'] != 0

    counts_df = counts_df[id_not_nan]

    counts_df['ratio_obs'] = (counts_df['count']/counts_df['total_non_nan'])*100

    return counts_df
```

The function above can be described several steps:

Calculate Reference Range: The function receives three parameters: `df` (the input DataFrame of LAI values), `df_ref` (a reference DataFrame containing mean and standard deviation values), and `nb_stdev` (the number of standard deviations to consider for the reference range). The function calculates the upper and lower bounds of the reference range by adding and subtracting `nb_stdev` times the standard deviation from the mean. Initialize Counts DataFrame: A new DataFrame named `counts_df` is created with columns "count," "consec_count," and "total_non_nan." This DataFrame will store the calculated counts and ratios for each parcel. Then the function loops over each row (parcel) in the input DataFrame `df`. For each parcel:

- The LAI values for the current parcel are assigned to `parcel_lai`.

- The reference range bounds for each date are assigned to `ref_lai_min` and `ref_lai_max`.

- The loop iterates over each LAI value in the parcel's data:

  - If the value is NaN, it's skipped.

  - If the value is outside the reference range, the `count` is incremented, and if the previous value was also out of range, `consec_count` is incremented. The `prev_out_of_range` flag keeps track of consecutive out-of-range values.

  - If the value is within the reference range, `prev_out_of_range` is reset to `False`.

- `total_non_nan` is calculated as the total count of non-NaN LAI values in the parcel.

A new dataframe 'parcel_counts_df' is then created top store the counts and ratios for the current parcel. It includes the calculated `count`, `consec_count`, and `total_non_nan` values. Secondly, the `parcel_counts_df` is concatenated with the main `counts_df` using `pd.concat`, effectively adding the counts and ratios for the current parcel to the overall counts DataFrame. A filtering step removes the rows in `counts_df` where the `total_non_nan` count is zero. Finally, the function calculates the ratio of outliers for each parcel by dividing the `count` by the `total_non_nan` and multiplying by 100. This ratio is stored in a new column named `ratio_obs` in `counts_df`.

Overall, this function is used to analyze LAI data, identify outliers based on a reference range, and calculate the counts and ratios of outliers for each parcel in the dataset. Once this function is created, it is applied (see Algorithm 3-10). The input and output of the script are described in Table 3-4.

Table 3-4. Inputs and Outputs of the vegetation stability and outlier consecutiveness functions

Input names	Role	Default value [format]
marker	Marker used to compute the area under the curve.	'mean_LAI' [string]
product_type	Refers to the marker database from which the marker will be extracted.	's4c_mdb1' [string]
fromdate_Stability	The date that defines the beginning of the period of interest	'yyyy-09-01' for P1 [string] (yyyy-mm-dd)
Todate_Stability	List of features to be tested to build the calibration dataset of "bare soil". Only NDVI is mandatory.	'yyyy-12-31' [string] (yyyy-mm-dd)
Outputs names	**Role**	**Default value [format]**

veg_all	Dataframe containing the previous outputs from the vegetation growth and bare soil markers (see table 3-2) and the outputs of the algorithm. The new added columns are: • Ratio_stability; • Consec_stability;	• Ratio_stability [int] • Consec_stability [int]

Algorithm 3-10. Vegetation stability and consecutiveness markers computation

```
fromdate_Stability = '2021-09-01'
todate_Stability = '2021-12-31'
marker = 'mean_LAI'
#list_m = ','.join(markers_opt_main)

product_type = 's4c_mdb1'

url =
f"http://{ip_sen4cap}:8080/markers?site={site}&productType={product_type}&year={year}&mar
kers={marker}&from={fromdate_Stability}&to={todate_Stability}"
print(url)

re = requests.get(url, headers=headers)
#print(re.json())
result = re.json()["data"]
parcels = result['parcels']
tt = len(parcels)

for p in tqdm(range(0,tt,2000)):
    re_p = parcels[p:p+2000]
    for i in re_p:
        newid = i['id']
        if newid in decl_csv.index: ## As mentionned in the file Sen4CAP error, a
mismatch can happen between the produced declaration and the declaration stored in the
sql system
        #print(i['id'])
            val_i = pd.DataFrame()
            val_i['dates'] = result["dates"]
            val_i[marker] = i['markers'][marker]
            val_i = val_i.rename(columns={marker:'mean'})
            stab = count_lai_outliers(val_i,dict_ref[decl_csv.ctnum14a[newid]],1.5)
        #print(stab)
            if len(stab) !=0:
                veg_all.loc[veg_all.NewID==newid,'Ratio_stability'] =
stab['ratio_obs'][0]

                          veg_all.loc[veg_all.NewID==newid,'Consec_stability'] =
stab['consec_count'][0]
```

As for the other scripts, variables related to the period of analysis, to the needed marker and the Sen4CAP database it comes from are set at the beginning. The URL is then built as well as the connection to the plateform with the API. Then the script proceeds to iterate through the parcels in segments of 2000 parcels, displaying progress using the `tqdm` function. Within each iteration:

- A batch of parcel data is extracted and stored as `re_p`.

- For each individual parcel (`i`) in the current batch:

- The parcel's unique identifier, accessed through the 'id' key, is saved as `newid`.

- A check is performed to verify whether `newid` is present within the index of the `decl_csv` DataFrame.

- If the check passes:

  - An empty DataFrame called `val_i` is created.

  - The 'dates' and specific `marker` values are extracted from the result data and inserted into `val_i`.

  - The column name of `marker` in `val_i` is renamed to 'mean'.

  - The `count_lai_outliers` function is invoked, taking `val_i` as input, along with the corresponding reference DataFrame from `dict_ref` and a threshold value of 1.5.

  - If there are outcomes from the `count_lai_outliers` function:

    - The 'ratio_obs' and 'consec_count' values from the results are assigned to their respective columns in the `veg_all` DataFrame using the `loc` method.

4. Markers analysis

Once the markers have been computed, an analysis is performed for the two periods independently. The idea behind this separation comes from the fact that according the type of change, its occurrence might take place during the first period (i.e., from September to December) or during the second period (from January to June). Even if results of both P1 and P2 are used to provide the user with a prediction of change, keeping the information from both periods independently allows to keep a temporal insight.

For each of the two periods P1 and P2, a change score is given to each parcel. This score is calculated on the basis of the values of the various markers presented above. Logically, the higher the score, the higher the probability of an agricultural category change. More concretely, for each period, a trehsold is set on the change score. When the value exceeds the threshold, a change of agricultural category is predicted. The confidence of the prediction is intrinsically linked to the change score. If the change score is equal to the threshold, the prediction could be considered as **medium-high**. If the change score equals the highest reachable score, the prediction could be considered as **strong**. In between values are considered are associated with **good confidence** level. We **strongly recommend to code this part as a Jupyther notebook**. The scripts provided are given as an exemple and value should be tuned by the user. If for the same site, data from a previous year are available, an optimization could be performed first. A script is provided at the end of the section.

4.1 Change score computation

The script below (Algorithm 4-1) assigns a new column (‘ChangeP1’ or ‘ChangeP2’) to the dataframe ‘veg_all’ that contains for each parcel, the value of all computed markers. Default tresholds values for each agricultural category and each period are provided in Table 4-1 and Table 4-2 . These values should be adatpted according to the specificities of the region of interest.

Table 4-1. Change score computation : Tresholds values P1

	Marker	Tresholds	Change score value
Grassland	TTdaysS2	TTdaysS2 > 0	+2
	Ratio_stability	0 < Ratio_stability < 50 Ratio_stability > 50	+1 +1.5
	Consec_stability	Consec_stability > 0	+1
Permanent crop	TTdaysS2	TTdaysS2 > 0	+3
	AreaVeg	AreaVeg> 50	+1
	Ratio_stability	Ratio_stability > 20	+1
Annual crop	TTdaysS2	TTdaysS2 > 0	+1
	AreaVeg	AreaVeg > 50	+1,5

Table 4-2. Change score computation : Tresholds values P2

	Marker	Tresholds	Change score value
Grassland	TTdaysS2	TTdaysS2 > 0	+2
	Ratio_stability	0 < Ratio_stability < 25 Ratio_stability > 25	+1 +1.5
	Consec_stability	Consec_stability > 1	+1
Permanent crop	TTdaysS2	TTdaysS2 > 0	+3
	AreaVeg	AreaVeg> 25	+1
	Ratio_stability	Ratio_stability > 20	+1
Annual crop	TTdaysS2	TTdaysS2 > 0	+1
	AreaVeg	0 < AreaVeg < 50 AreaVeg > 50	+0,5 +1.5

Algorithm 4-1. Application of treshold values

```
P1

#1. Grassland changes
df['ChangeP1'] = np.nan
df.loc[(df['LC_20']==3),'ChangeP1'] = 0
df.loc[(df['LC_20']==3) & (df['TTdaysS2']>0),'ChangeP1'] += 2
df.loc[(df['LC_20']==3) & (df['Ratio_stability']>0) & (df['Ratio_stability']<50),
'ChangeP1'] += 1
df.loc[(df['LC_20']==3) & (df['Ratio_stability']>50), 'ChangeP1'] += 1.5
df.loc[(df['LC_20']==3) & (df['ConsecC_stability']>= 0) ,'ChangeP1'] += 1

#2. Permanent Crop changes
df.loc[(df['LC_20']==2),'ChangeP1'] = 0
df.loc[(df['LC_20']==2) & (df['TTdaysS2']>0),'ChangeP1'] += 3
df.loc[(df['LC_20']==2) & (df['AreaVeg']>50), 'ChangeP1'] += 1
df.loc[(df['LC_20']==2) & (df['Ratio_stability']>20), 'ChangeP1'] += 1

#3. Arable land changes
df.loc[(df['LC_20']==1),'ChangeP1'] = 0
df.loc[(df['LC_20']==1) & (df['TTdaysS2'] == 0),'ChangeP1'] += 1
df.loc[(df['LC_20']==1) & (df['AreaVeg']>50), 'ChangeP1'] += 1.5


 P2

#1. Grassland changes
df['ChangeP2'] = np.nan
df.loc[(df['LC_20']==3),'ChangeP2'] = 0
df.loc[(df['LC_20']==3) & (df['TTdaysS2_P2']>0),'ChangeP2'] += 2
df.loc[(df['LC_20']==3) & (df['Ratio_stability_P2']>0) &
(df['Ratio_stability_P2']<25), 'ChangeP2'] += 1
```

```
df.loc[(df['LC_20']==3) & (df['Ratio_stability_P2']>25), 'ChangeP2'] += 1.5
df.loc[(df['LC_20']==3) & (df['ConsecC_stability_P2']>= 1) ,'ChangeP2'] += 1

#2. Permanent Crop changes
df.loc[(df['LC_20']==2),'ChangeP1'] = 0
df.loc[(df['LC_20']==2) & (df['TTdaysS2_P2']>0),'ChangeP2'] += 3
df.loc[(df['LC_20']==2) & (df['AreaVeg_P2']>25), 'ChangeP2'] += 1
df.loc[(df['LC_20']==2) & (df['Ratio_stability_P2']>20), 'ChangeP2'] += 1

#3. Arable land changes
df.loc[(df['LC_20']==1),'ChangeP2'] = 0
df.loc[(df['LC_20']==1) & (df['TTdaysS2_P2'] == 0),'ChangeP2'] += 1
df.loc[(df['LC_20']==1) & (df['AreaVeg_P2']> 0) & (df['AreaVeg_P2']<50),
'ChangeP2'] += 0.5
df.loc[(df['LC_20']==1) & (df['AreaVeg_P2']>50), 'ChangeP1'] += 1.5
```

After having given a change score to each parcel, a prediction is made based on certain threshold. Again, this threshold should be adapted to fit the region of interet specificities. Two columns are added to the dataframe containing 1 or 0 if a prediction of change is made or not (Algorithm 4-2). Enventually a last column is added and contain the sum of ‘pred_changeP1’ and ‘pred_changeP2’.

Algorithm 4-2. Change prediction

```
veg_all ['pred_changeP1'] = 0
veg_all.loc[(df['ChangeP1'] >= 2.5),'pred_changeP1'] +=1


veg_all ['pred_changeP2'] = 0
veg_all.loc[(df['ChangeP2'] >= 3),'pred_changeP2'] +=1


veg_all ['pred_change_P1_P2'] = df['pred_changeP1'] + df['pred_changeP2']
```

4.2 Tresholds optimization

Algorithm 4-3 has been designed to optimize the detection of specific changes. As mentioned, this optimization phase is only possible if results from a previous year are available.

The optimization method consists in trying all markers combinations’ tresholds to find the combination that maximizes true change dections while minimizing false change detection. In this case, the script has been tunned to maximize the detection of change from grasslands to winter crops. Two subsets are defined: one for grasslands that become winter crops and another for grasses that stay grasses. The script evaluates the model's performance on each subset separately. After having defined a percentage of tolerated false positives, a performance metric is defined. This metric consists in a ratio between false negative for the winter crops (changes that occurred but are not detected) and true negative of grasses (grasses that are predicted to stay grasses). The best combination of parameters is the one that minimized the performance metric. Once it has been found, it is applied to the whole dataset.

Algorithm 4-3. Optimization algorithm

```
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)

from sklearn.metrics import accuracy_score, recall_score
```

```
# Define the threshold ranges to try
threshold_ranges = {
    'TTdaysS2': [0, 1, 2, 3, 4, 5],
    'Ratio_stability_min': [0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50],
    'Ratio_stability_max': [0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50],
    'ConsecC_stability': [0, 1]
}

# Define a list of threshold combinations to try
threshold_combinations = []
for tt in threshold_ranges['TTdaysS2']:
    for rsmin in threshold_ranges['Ratio_stability_min']:
        for rsmax in threshold_ranges['Ratio_stability_max']:
            for cs in threshold_ranges['ConsecC_stability']:
                threshold_combinations.append((tt, rsmin, rsmax, cs))

# Evaluate each combination and save the performance metrics
performance = []
for combination in threshold_combinations:
    tt_threshold, rsmin_threshold, rsmax_threshold, cs_threshold = combination

    # Apply the thresholds to the dataframe
    df['ChangeP1'] = np.nan
    df.loc[df['LC_20'] == 3, 'ChangeP1'] = 0
    df.loc[(df['LC_20'] == 3) & (df['TTdaysS2'] > tt_threshold), 'ChangeP1'] += 1.5
    df.loc[(df['LC_20'] == 3) & (df['Ratio_stability'] > rsmin_threshold) &
(df['Ratio_stability'] < rsmax_threshold), 'ChangeP1'] += 1
    df.loc[(df['LC_20'] == 3) & (df['Ratio_stability'] > rsmax_threshold), 'ChangeP1'] +=
1.5
    df.loc[(df['LC_20'] == 3) & (df['ConsecC_stability'] >= cs_threshold), 'ChangeP1'] +=
1

    df['pred_changeP1'] = 0
    df.loc[df['ChangeP1'] >= 2.5, 'pred_changeP1'] += 1

    # Select the subset for which we want to optimize the detection (in this case, winter
crops: Winter Cereal and Spelt)
    subset = df.loc[(df['LC_20'] == 3) & (df['LC_21'] == 1) & ((df['CTL4A_21'] == 'Winter
Cereal') | (df['CTL4A_21'] == 'Spelt'))]

    # Evaluate the performance of the model on the subset (winter crops & spelt)
    true_change = subset['true_change']
    predicted_change = subset['pred_changeP1']
    tp = sum((predicted_change == 1) & (true_change == 1))
    tn = sum((predicted_change == 0) & (true_change == 0))
    fn = sum((predicted_change == 0) & (true_change == 1))

    recall = recall_score(true_change, predicted_change)

    # Evaluate the performance of the model on the subset (grasses to grasses)
    subset_g = df.loc[(df['LC_20'] == 3) & (df['LC_21'] == 3)]
    true_change = subset_g['true_change']
    predicted_change = subset_g['pred_changeP1']
    tp_g = sum((predicted_change == 1) & (true_change == 1))
    tn_g = sum((predicted_change == 0) & (true_change == 0))
    fp_g = sum((predicted_change == 1) & (true_change == 0))

    recall_g = recall_score(true_change, predicted_change)

    # Calculate the performance metric that reflects the desired optimization goal
    if len(subset_g) > 0 and (fp_g / len(subset_g)) <= 0.05:
    performance_metric = fn / tn_g
    performance.append((tt_threshold, rsmin_threshold, rsmax_threshold, cs_threshold, 
recall, performance_metric))

    # Find the best performing combination of thresholds based on the performance metric
    best_performance = min(performance, key=lambda x: x[-1])
    best_thresholds = best_performance[:4]
    best_performance_metric = best_performance[-1]

    # Get the best threshold values
    tt_threshold, rsmin_threshold, rsmax_threshold, cs_threshold = best_thresholds

    # Apply the best thresholds to the dataframe
    df['ChangeP1'] = np.nan
    df.loc[df['LC_20'] == 3, 'ChangeP1'] = 0
    df.loc[(df['LC_20'] == 3) & (df['TTdaysS2'] > tt_threshold), 'ChangeP1'] += 1.5
    df.loc[(df['LC_20'] == 3) & (df['Ratio_stability'] > rsmin_threshold) & 
    (df['Ratio_stability'] < rsmax_threshold), 'ChangeP1'] += 1
    df.loc[(df['LC_20'] == 3) & (df['Ratio_stability'] > rsmax_threshold), 'ChangeP1'] += 1.5
    df.loc[(df['LC_20'] == 3) & (df['ConsecC_stability'] >= cs_threshold), 'ChangeP1'] += 1

    df['pred_changeP1'] = 0
    df.loc[df['ChangeP1'] >= 2.5, 'pred_changeP1'] += 1

    # Select the subset for which we want to optimize the detection (in this case, winter 
    crops: Winter Cereal and Spelt)
    subset = df.loc[(df['LC_20'] == 3) & (df['LC_21'] == 1) & ((df['CTL4A_21'] == 'Winter 
    Cereal') | (df['CTL4A_21'] == 'Spelt'))]

    # Evaluate the performance of the model on the subset (winter crops & spelt)
    true_change = subset['true_change']
    predicted_change = subset['pred_changeP1']
    tp = sum((predicted_change == 1) & (true_change == 1))
    tn = sum((predicted_change == 0) & (true_change == 0))
    fn = sum((predicted_change == 0) & (true_change == 1))

    # Evaluate the performance of the model on the subset (grasses to grasses)
    subset_g = df.loc[(df['LC_20'] == 3) & (df['LC_21'] == 3)]
    true_change = subset_g['true_change']
    predicted_change = subset_g['pred_changeP1']
    tp_g = sum((predicted_change == 1) & (true_change == 1))
    tn_g = sum((predicted_change == 0) & (true_change == 0))


    fp_g = sum((predicted_change == 1) & (true_change == 0))

    print("Best performance: Metric = {:.4f}, Recall = 
    {:.4f}".format(best_performance_metric, best_performance[-2]))

    print("Best thresholds: tt_threshold = {}, rsmin_threshold = {}, rsmax_threshold = {}, 
    cs_threshold = {}".format(tt_threshold, rsmin_threshold, rsmax_threshold, cs_threshold))

    print("Best performance: tp = {}, tn = {}, fn = {}, tp_g = {}, tn_g = {}, fp_g = 
    {}".format(tp, tn, fn, tp_g, tn_g, fp_g))
```

5. Results consolidation with L4A processor results

This last step consists in using the crop type prediction map to confirm or invalidate predictions made with the different markers. Based on the crop type prediction and its confidence level, the change score will be increased or decreased. At this point, it is important to note that this consolidation phase could only take place if the crop type algorithm have been excecuted (i.e., at the end of P2). To be performed, a declaration and a look-up table for the current year (year of P2) should be imported in the system. This will be first set as an option to allow the user to run the L4F processor without the use of the L4A processor. As for the change score computation, we **strongly recommend to code this last part in a Jyputer noterbook** and to adapt the tresholds and values that are given as an exemple here below.

As for the bare soil processor, the L4A crop type processor will be launched automatically when launching the L4F category change processor. This processor uses Sentinel-1 and Sentinel-2 data to extract timely resampled markers at the parcel level. These markers are then used by a Random Forest classification algorithm to predict a crop type map. A detailed explaination of the algorithm operation could be found in the L4A ATBD [RD.3]. Parameters (default) used to run the L4A processor are described in the same document. The period used to predict the crop type extends from January to June (i.e., P2). The way the change score is modified accordin the L4A processor results is described below. The example focuses on P1 but the same methodology is applied for P2 accordingly.

Algorithm 5-1. Consolition of change prediction with L4A results

```
## df is equal to previous veg_all

df['ChangeP1_CT'] = df['ChangeP1']
df.loc[(df['Pred_conf_1'] >= 0.85) & (df['ChangePred_CT'] == 1) , 'ChangeP1_CT'] += 2
df.loc[(df['Pred_conf_1'] >= 0.85) & (df['ChangePred_CT'] == 0) , 'ChangeP1_CT'] -= 2

df.loc[(df['Pred_conf_1'] > 0.2)  & (df['Pred_conf_1'] < 0.85) & (df['ChangePred_CT'] ==
1) , 'ChangeP1_CT'] += 1

df.loc[(df['Pred_conf_1'] > 0.2)  & (df['Pred_conf_1'] < 0.85) & (df['ChangePred_CT'] ==
0) , 'ChangeP1_CT'] -= 1
```

After having applied an increase or a decrease in the change score, a prediction is made based on certain threshold. This threshold is given as an example and should be adapted. Two columns new are added to the dataframe containing 1 or 0 if a prediction of change is made or not.

Algorithm 5-2. Consolidated change prediction

```
df['pred_changeP1_CT'] = 0
df.loc[(df['ChangeP1_CT'] >= 2.5), 'pred_changeP1_CT'] += 1

df['pred_changeP2_CT'] = 0
df.loc[(df['ChangeP2_CT'] >= 3), 'pred_changeP2_CT'] += 1
```

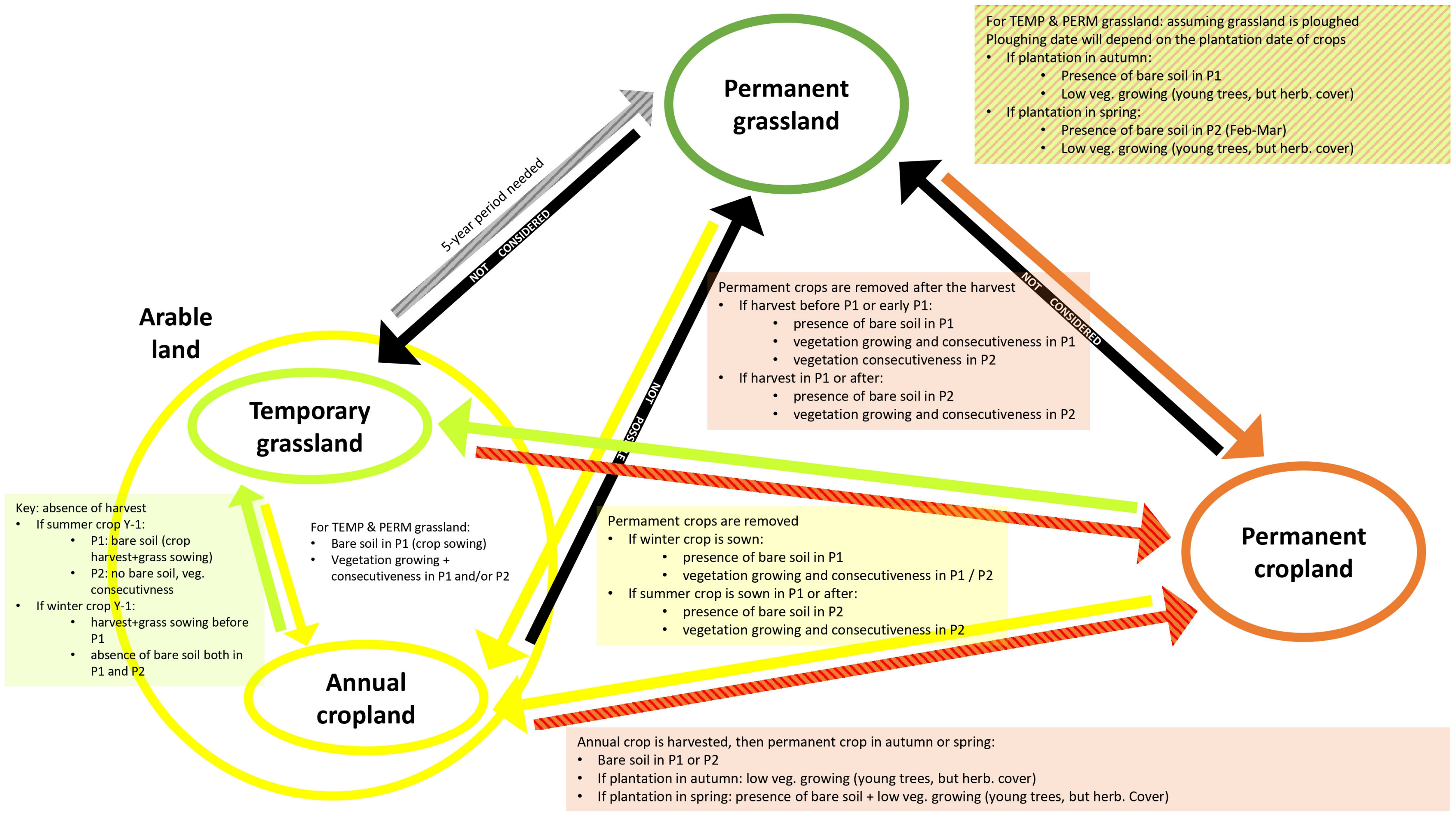
6. Output and interpretation

The ouput of the L4F category consists in a CSV file, **L4F_ACC_Markers_Pred_{site name}.csv**, gathering together all the markers for the two periods, the results of the crop type algorithm, the change score with and without the use of the crop ype change detection and the final prediction with and without the use of the crop type change detection. The columns of the file are described in Table 6-1. Y0 indicates the year of P1 and Y1 indicates the year of P2. To make it easier to read the results, an interpretation grid is also provided (see Figure 6-1).

Table 6-1. Csv Output content

Column name	Description	Value format
NewID_Y0	Parcel ID generated by the system (year 0)	[integer]
NewID_Y1[1]	Parcel ID generated by the system (year 1)	[integer]
S2_Pix	Number of Sentinel 2 pixels used in the parcel	[integer]
LC_Y0	Land cover type (see Table 2-3) (year 0)	[integer]
PGrass_Y0	Permanent grassland: 1 if the crop type belongs to this category, 0 otherwise (year 0)	[integer, binary]
CTnum_Y0	The new crop type code (each Ori_crop being associated to a unique CTnum) (year 0)	[integer]
CTL4A_Y0	The crop type name associated to CTnumL4A (year 0)	[string]
TTdays_S2_P1	The number of days of bare soil observed with Sentinel-2 (i.e. the sum of all bare soil periods observed in the parcel) for P1	[integer]
AreaVeg_P1	The area under the curve for P1	[integer]
Ratio_Stability_P1	The stability ratio for P1	[integer]
Consec_Stability_P1	The outlier consecutiveness for P1	[integer]
LC_Y1	Land cover type (see Table 2-3) (year 1)	[integer]
PGrass_Y1	Permanent grassland: 1 if the crop type belongs to this category, 0 otherwise (year 1)	[integer, binary]
CTnum_Y1	The new crop type code (each Ori_crop being associated to a unique CTnum) (year 1)	[integer]
CTL4A_Y1	The crop type name associated to CTnumL4A (year 1)	[string]

Column name	Description	Value format
TTdays_S2_P2	The number of days of bare soil observed with Sentinel-2 (i.e. the sum of all bare soil periods observed in the parcel) for P2	[integer]
AreaVeg_P2	The area under the curve for P2	[integer]
Ratio_Stability_P2	The stability ratio for P2	[integer]
Consec_Stability_P2	The outlier consecutiveness for P2	[integer]
ChangeP1	Change score for P1	[integer]
Pred_changeP1	Change prediction for P1, 1 if a change is predicted, 0 otherwise	[integer, binary]
ChangeP2	Change score for P2	[integer]
Pred_changeP2	Change prediction for P2, 1 if a change is predicted, 0 otherwise	[integrer, binary]
Pred_change_P1_P2	Sum of Pred_changeP1 & Pred_changeP2	[integer]
ChangePred_CT	Change prediction according to L4A processor results	[integer, binary]
Pred_conf_1	Confidence level of L4A crop type prediction	[float]
ChangeP1_CT	Change score for P1 adapted with ChangePred_CT	[integer]
Pred_changeP1_CT	Change prediction for P1 according to L4A prediction, 1 if a change is predicted, 0 otherwise	[integer, binary]
ChangeP2_CT	Change score for P2 adapted with ChangePred_CT	[integer]
Pred_changeP2_CT	Change prediction for P2 according to L4A prediction, 1 if a change is predicted, 0 otherwise	[integer, binary]



For TEMP & PERM grassland: assuming grassland is ploughed
Ploughing date will depend on the plantation date of crops
- If plantation in autumn:
  - Presence of bare soil in P1
  - Low veg. growing (young trees, but herb. cover)
- If plantation in spring:
  - Presence of bare soil in P2 (Feb-Mar)
  - Low veg. growing (young trees, but herb. cover)

Permanent crops are removed after the harvest
- If harvest before P1 or early P1:
  - presence of bare soil in P1
  - vegetation growing and consecutiveness in P1
  - vegetation consecutiveness in P2
- If harvest in P1 or after:
  - presence of bare soil in P2
  - vegetation growing and consecutiveness in P2

Key: absence of harvest
- If summer crop Y-1:
  - P1: bare soil (crop harvest+grass sowing)
  - P2: no bare soil, veg. consecutivness
- If winter crop Y-1:
  - harvest+grass sowing before P1
  - absence of bare soil both in P1 and P2

For TEMP & PERM grassland:
- Bare soil in P1 (crop sowing)
- Vegetation growing + consecutiveness in P1 and/or P2

Permanent crops are removed
- If winter crop is sown:
  - presence of bare soil in P1
  - vegetation growing and consecutiveness in P1 / P2
- If summer crop is sown in P1 or after:
  - presence of bare soil in P2
  - vegetation growing and consecutiveness in P2

Annual crop is harvested, then permanent crop in autumn or spring:
- Bare soil in P1 or P2
- If plantation in autumn: low veg. growing (young trees, but herb. cover)
- If plantation in spring: presence of bare soil + low veg. growing (young trees, but herb. Cover)

Figure 6-1. Interpretation grid

In order to break down the different possibilities for change, we have decided to distinguish between temporary grassland and annual crops, even though they are part of the same agricultural category.

Arable land - Permanent grassland

The first important thing to note is that temporary grassland is considered as arable land. It is only after 5 years that these grasslands are considered as permanent grasslands. It is therefore concretely impossible to detect a change in category from "temporary grassland" to "permanent grassland".

A change of category from "permanent grassland" to "temporary grassland" is not considered because it is impossible to detect and, in fact, virtually improbable. Similarly, the change from "annual crop" to permanent grassland" is not considered because it is simply impossible.

A return from "permanent grassland" to an "annual crop", on the other hand, can be observed. If a winter crop is sown, the grassland will probably be turned over and bare soil will appear in P1. If it is a summer crop that is sown, then this reversal could be observed in P2.

Arable land - Permanent cropland

A change from "permanent cropland" to "arable land" temporary grassland and annual crop) can be observed after harvesting. Harvesting either takes place before or at the beginning of P1, or during or after P1. In the first case, bare soil will most likely be observed in P1. In addition to bare soil, the markers 'vegetation growth' as well as 'stability' and 'consecutivity' should indicate a change. If harvesting takes place during or after P1, a period of bare soil period could be observed in P2. The other markers may also indicate a change.

As for the opposite change, it may take place in the first or second period. This change will be accompanied by a period of bare soil and sparse vegetation (young trees).

Permanent crop - Permanent grassland

The change "permanent cropland" to "permanent grassland" is not considered because it is impossible. The reverse change depends on when the crop is planted. If the crop is planted in autumn, it is likely to be accompanied by a period of bare soil and poor vegetation growth. If the crop is planted in spring, it is likely to be accompanied by a period of bare soil and a resumption of vegetation growth.

Change inside Arable Land

Although not considered to be a change in agricultural category, the change from temporary grassland to annual crop is relatively likely. In this case, the grassland will be replaced by either a summer crop or a winter crop. In the case of a winter crop, the grassland will most likely be turned over to P1, implying a period of bare soil. P2 will be characterised by an increase in vegetation growth and by the activation of stability and consecutive markers. If, on the other hand, a summer crop is sown, bare soil is unlikely to be observed in the first period. In the second period, however, bare soil is very likely to be observed.

7. Appendices

Algorithm 7-1. LPIS standardization

```
import geopandas as gpd
import pandas as pd

lpis = gpd.read_file(lpis_file)
lpis['idx_lpis'] = lpis.index
lpis['area_lpis'] = lpis.area

old_lpis = gpd.read_file(old_lpis_file)
old_lpis['idx_old_lpis'] = old_lpis.index
old_lpis['area_old_lpis'] = old_lpis.area

inter_lpis = gpd.overlay(old_lpis,lpis, how='intersection', keep_geom_type=True)

inter_lpis['area_inter'] = inter_lpis.area
inter_lpis['area_union'] = inter_lpis['area_old_lpis'] + inter_lpis['area_lpis'] -
inter_lpis['area_inter']
inter_lpis['area_sym_dif'] = inter_lpis['area_union'] - inter_lpis['area_inter']
inter_lpis['change_ratio'] = inter_lpis['area_sym_dif']/inter_lpis['area_union']

change_threshold = 0.05
area_threshold = 100000


same_lpis = inter_lpis.loc[(inter_lpis['area_sym_dif'] < area_threshold) &
(inter_lpis['change_ratio'] < change_threshold)]



id_stdlpis_lux_2020 = pd.unique(same_lpis['idx_old_lpis'])

id_stdlpis_lux_2021 = pd.unique(same_lpis['idx_lpis'])


all_indexes = same_lpis[['fid','NewID','fid_1','NewID_1']]
all_indexes = all_indexes.rename(columns = {'fid_1' : 'fid_old','NewID_1':
'NewID_20',  'fid_2' : 'fid', 'NewID_2': 'NewID_21'})

all_indexes['global_id'] = range(1, len(same_lpis)+1)

stdlpis_lux_2020 = old_lpis.loc[id_stdlpis_lux_2020]
stdlpis_lux_2020 = stdlpis_lux_2020.rename(columns = {'fid': 'fid_old'})
stdlpis_lux_2021 = lpis.loc[id_stdlpis_lux_2021]
```