Sentinel-2 Agriculture

Crop type DPM

Contents

1 Introduction

This document describes the proposed processing chain for the production of the crop type map for the Sentinel-2 Agriculture project.

The algorithm description and justification of choices have been documented in the Design Justification File[1]. The present document describes the processing chain and its subsystems.

Where possible, we use standard components available in the Orfeo Toolbox version 4.4[2]. When no equivalent component is available in the Orfeo Toolbox, the algorithm is described in pseudo-code.

An example implementation of the processing chain using Python as a glue for the different components is available at http://tully.ups-tlse.fr/jordi/croptype_bench/tree/master.

2 Processor components

2.1 Overview of the system

Figure 1 presents an overview of the processing chain. There are 3 main subsystems:

1. data preparation,
2. supervised learning with SVM [3] and Random Forests [4],
3. map production.

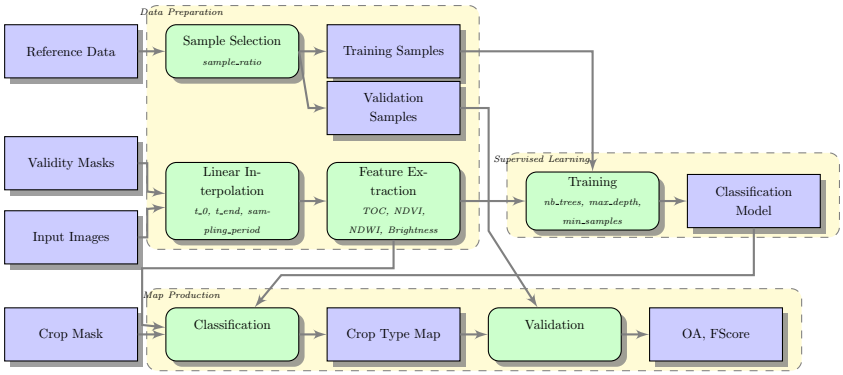

Figure 1: Block diagram of the crop type map production.

Figure 1 assumes that the same classifier is used for the whole mapped area. In order to take into account the variability of vegetation behavior accross a complete country, the use of different classifiers in different regions has to be possible in the system. The way of taking into account this segmentation of the mapped area is through spatial masks. The system will use a user-provided mask defining the spatial regions. Each region will be identified by a label. All regions with the same label will be used for training the same classifier and, of course, will be processed using the same classifier.

This design allows to:

1. Process the whole area with a single classifier: all the pixels of the mask belong to the same region.
2. Process the whole area with a classifier per tile: the mask has a different label for each tile covering the area.
3. Use an eco-climatic stratification: the mask has a different label for each stratum.

[1] Sentinel-2 for Agriculture Design Justification File (v1.1, 2015/04/15)
[2] This version corresponds to the changeset http://hg.orfeo-toolbox.org/OTB/file/baf740ee2113.
[3] C. Burges. A tutorial on support vector machines for pattern recognition, Knowledge Discovery and Data Mining 2(2), 1998.
[4] Breiman, Leo, Random forests, Machine learning, 45, 5–32, 2001.

This approach assumes that there is training data for each region of the mask. The user is responsible for ensuring that. She only has the knowledge needed to group together regions, in terms of landscape similarity for instance, in order to efficiently exploit the reference data.

2.2 Data preparation

The data preparation subsystem aims at generating the input data for the supervised learning and the map production subsystems. One processing line takes care of the preparation of the reference data. A second processing line prepares the image time series.

2.2.1 Sample selection

The sample selection consists in splitting the reference data into 2 disjoint sets, the training set and the validation set. These sets are composed of polygons, not individual pixels.

Table 1: Variables for the sample selection algorithm

Input variable	role	default value
`reference_polygons`	Vector file containing reference data	–
`sample_ratio`	Ratio between the number of training and validation polygons per class	0.75
Output variable	**role**	**–**
`training_polygons`	Vector file containing reference data for training	–
`validation_polygons`	Vector file containing reference data for validation	–

The algorithm is therefore a random sampling without replacement of the polygons of each class with probability $p =$ `sample_ratio` value for the training set and $1 - p$ for the validation set.

Algorithm 1: Sample selection algorithm

```
Data: reference_polygons, sample_ratio, list_of_class_labels
Result: training_polygons, validation_polygons
begin
    training_polygons ← ∅;
    validation_polygons ← ∅;
    for cl ∈ list_of_class_labels do
        for poly ∈ reference_polygons do
            if class_of(poly) = cl then
                p ← random(0, 1);
                if p ≤ sample_ratio then
                    add poly to training_polygons;
                end
                else
                    add poly to validation_polygons
                end
                remove poly from reference_polygons;
            end
        end
    end
end
```

2.2.2 Linear interpolation and gapfilling

The goal of the linear interpolation is to produce a reflectance image time series which is gapfilled with respect to missing data (the validity masks contain clouds, cloud shadows and saturated pixels) and temporally sampled on a regular grid. If several sensors are available (for instance Sentinel-2 and LANDSAT-8) their respective

reflectance time series are processed independently for the interpolation and the gapfilling steps. For each sensor, the temporal sampling rate will be equal to its revisit cycle (10 days for Sentinel-2 with one satellite, 5 days with 2 satellites, 16 days for Landsat8). The starting date for each interpolated time series can be the first available date if one singe scene (or tile) is being processed. In the case of a very large area where several satellite tracks are used, several choices are possible:

1. Use the earliest first date of all tracks.
2. Use the latest first date of all tracks.
3. Use the median of the first dates of all tracks.

Options 1 and 2 can lead to sub-optimal choices when there is much variability in the first dates for all tracks. The same rationale can be applied to the last date.

These time series are afterwards concatenated before providing them to the classification step and being processed as a single time series.

Algorithm 2 describes the procedure. In the case where the whole series is available, t_0 and t_end correspond to the first and last dates of the regular grid.

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

Table 2: Variables for the linear interpolation and gapfilling algorithm

Input variable	role	default value
`tocr`	S2 L2A surface reflectances	–
`mask`	Validity masks for each acquisition date	–
`input_dates`	Dates of each image acquisition	–
`sampling_period`	Temporal sampling rate	5
`t_0`	Starting sampling date	median of first dates
`t_end`	Last date	median of last dates
`radius`	radius of the temporal window	15
Output variable	**role**	
`rtocr`	Resampled S2 L2A surface reflectances	–

Algorithm 2: Temporal resampling

Data: $tocr$, $mask$, $input\_dates$, $sampling\_period$, t_0, t_end
Result: $rtocr$
begin
  for $pixel \in tocr, mpixel \in mask$ **do**
    $odc \leftarrow 0$;
    $od \leftarrow t\_0 + sampling\_period \times odc$;
    $outpix \leftarrow \emptyset$;
    while $od \leq t\_end$ **do**
      $pvd \leftarrow find\_previous\_valid\_date(od, mask, input\_dates)$;
      $nvd \leftarrow find\_next\_valid\_date(od, mask, input\_dates)$;
      $pweight \leftarrow od - pvd$;
      $nweight \leftarrow nvd - od$;
      $outpix[od] \leftarrow \frac{pixel[pvd] \times pweight + pixel[nvd] \times nweight}{pweight + nweight}$;
      $odc \leftarrow odc + 1$;
      $od \leftarrow t\_0 + sampling\_period \times odc$;
    end
    $rtocr[positon(pixel)] \leftarrow outpix$;
  end
end

2.2.3 Feature extraction

The feature extraction step produces the relevant features for the classification. The features are computed for each date of the resampled and gapfilled time series and concatenated together into a single multi-channel image file. The selected features are the surface reflectances, the NDVI, the NDWI and the brightness.

Algorithm 3 describes the procedure, which can easily be implemented using the ORFEO Toolbox `BandMath` application.

The NDVI computation will use the B8 band (not the B8a). The SWIR band will be resampled at 10 m.

Table 3: Variables for the feature extraction algorithm

Input variable	role
`rtocr`	Resampled S2 L2A surface reflectances
Output variable	**role**
`fts`	Feature time series

Algorithm 3: Feature extraction

Data: $rtocr$
Result: fts
begin
  for $pixel \in rtocr$ **do**
    $ndvi \leftarrow \frac{pixel[NIR] - pixel[R]}{pixel[NIR] + pixel[R]}$;
    $ndwi \leftarrow \frac{pixel[SWIR] - pixel[NIR]}{pixel[SWIR] + pixel[NIR]}$;
    $bright \leftarrow \sqrt{pixel[G]^2 + pixel[R]^2 + pixel[NIR]^2 + pixel[SWIR]^2}$;
    $concatenate(pixel, ndvi, ndwi, bright)$ **end**
    $fts[positon(pixel)] \leftarrow outpix$;
  end

2.3 Supervised learning

Two supervised classification algorithms are used: Support Vector Machine and Random Forests. They are used through the Orfeo Toolbox applications which relie on the OpenCV implementation.

Detailed documentation for these 2 algorithms is available online[5].

2.3.1 Compute image statistics

This step is optional for the random forest classifier, but recommended for the SVM. The goal here is to compute the mean and the standard deviation of each input feature of the classifier so that the samples can be normalized inside the training and classification steps.

Table 4: Variables for the statistics computation

Input variable	role
`fts`	Feature time series
Output variable	**role**
`statistics`	Mean and standard deviation for each input feature

The Orfeo Toolbox `ComputeImagesStatistics` application is used in order to produce an XML file containing the statistics for each channel of the image of features.

```
otbcli_ComputeImagesStatistics -il feature-time-series.tif -out statistics.xml
```

2.3.2 Training the classifier

The classifier is trained using the image of features and the training samples extracted from the reference data. For both algorithms (RF and SVM) there is a set of common variables, which are the input data (the time series, the training data, the optional statistics file), the seed for the random number generator (in order to be able to reproduce exactly the same results in different runs) and the ratio between training and validation samples.

It is worth noting that, even in the training step, a validation of the quality of the classification is performed. This allows to check if the training behaved correctly before producing the complete crop map for the final validation. This is the utility of the parameter giving the ration between training and validation samples. However, the sample splitting in this step does not garantee that a training and a validation sample will not be drawn from the same polygon. These may lead to very similar training and validation samples and, therefore, the obtained confusion matrix may give optimistic performances.

Table 5: Variables for the classifier training

Input variable	role	default value
`fts`	Feature time series	–
`statistics`	XML file containing mean and standard deviation for each input feature	–
`training-polygons`	Vector file containing reference data for training	–
`random-seed`	Seed for the random number generator	0
`classifier`	Classifier algorithm: `rf` or `svm`	rf
`sample.vtr`	Ratio between the number of training and validation samples	0.9
Output variable	**role**	
`confusion-matrix`	File containing the confusion matrix	–
`model`	File containing the classification model	–

[5] SVM: http://docs.opencv.org/modules/ml/doc/support_vector_machines.html; Random Forests: http://docs.opencv.org/modules/ml/doc/random_trees.html.

2.3.2.1 Random Forests

The specific parameters for the Random Forest classifier allow to set the complexity of the classifier and limit the risk of over-fitting.

Table 6: Specific input variables for the Random Forest classifier

Input variable	role	default value
`rf.nbtrees`	number of decision trees	100
`rf.min`	minimum number of samples per node	25
`rf.max`	maximum depth of each tree	25

The `TrainImagesClassifier` Orfeo Toolbox application is used as follows:

```
otbcli_TrainImagesClassifier -io.il feature-time-series.tif\
                             -io.vd training-polygons.shp\
                             -io.imstat statistics.xml\
                             -rand random-seed\
                             -sample.bm 0\
                             -io.confmatout confusion-matrix.csv\
                             -io.out model.txt\
                             -classifier rf\
                             -classifier.rf.nbtrees 100\
                             -classifier.rf.min 5\
                             -classifier.rf.max 25\
                             -sample.mt -1\
                             -sample.mv -1\
                             -sample.vtr 0.6
```

2.3.2.2 Support Vector Machines

The SVM classifier is used with a Gaussian (RBF) kernel, which allows good generalization properties. The 2 parameters of the algorithm (the C cost and the kernel width) are optimized by cross-validation, and therefore they do not need to be set.

Table 7: Specific input variables for the SVM classifier

Input variable	role	default value
`svm.k`	Type of kernel	rbf
`svm.opt`	Automatic optimization of the parameters	true

The `TrainImagesClassifier` Orfeo Toolbox application is used as follows:

```
otbcli_TrainImagesClassifier -io.il feature-time-series.tif\
                             -io.vd training-polygons.shp\
                             -io.imstat statistics.xml\
                             -rand random-seed\
                             -sample.bm 1\
                             -io.confmatout confusion-matrix.csv\
                             -io.out model.txt\
                             -classifier 'svm'\
                             -classifier.svm.k 'rbf'\
                             -classifier.svm.opt 1\
                             -sample.mt -1\
                             -sample.mv -1\
                             -sample.vtr 0.6\
```

2.4 Map production

The map production subsystem has 2 steps: the crop type map generation and its validation.

2.4.1 Crop type map generation

The crop type map generation applies the supervised classification to the feature image using the model. The statistics file is needed in order for the classifier to normalize the samples in the same way as in the training. The crop mask is used in order to limit the classification to the pixels inside the crop mask.

Table 8: Variables for the map generation algorithm

Input variable	role
`fts`	Feature time series
`statistics`	XML file containing mean and standard deviation for each input feature
`model`	File containing the classification model
`crop-mask`	Binary crop mask
Output variable	**role**
`crop-type-map`	Crop type map

The `ImageClassifier` Orfeo Toolbox application is used as follows:

```
otbcli_ImageClassifier -in feature-time-series.tif\
                       -imstat statistics.xml\
                       -mask crop-mask\
                       -model model.txt\
                       -out crop-type-map.tif
```

2.4.2 Crop type map validation

Once the crop type map has been generated, it can be validated using the validation polygons selected in the data preparation step. The output confusion matrix is stored into a file and the quality metrics (F-Score, precision, recall, κ coefficient and Overall Accuracy) are stored into a second file.

Table 9: Variables for the map validation algorithm

Input variable	role
`crop-type-map`	Crop type map
`validation_polygons`	Vector file containing reference data for validation
Output variable	**role**
`confusion-matrix_validation`	File containing the confusion matrix
`quality-metrics`	File containing the quality metrics

The `ComputeConfusionMatrix` Orfeo Toolbox application stores the confusion matrix into a file, but prints the quality metrics to standard output. Therefore, `stdout` is redirected to the file where the quality metrics will be stored:

```
otbcli_ComputeConfusionMatrix -in crop-type-map.tif\
                              -out confusion-matrix-validation.csv\
                              -ref vector\
                              -ref.vector.in validation_polygons.shp\
                              -ref.vector.field Class > quality-metrics.txt
```

2.4.3 Quality flags

The crop type map will be provided along with four quality flags indicating:

- the number of dates which are associated with the "land" status during the period used to generate the map;
- the number of dates which are associated with the "water" status during the period used to generate the map;
- the number of dates which are associated with the "snow" status during the period used to generate the map;

- the number of dates which are associated with the other statuses ("cloud", "cloud shadow", "no data") during the period used to generate the map.

These *number of dates* make reference to the time series before temporal gapfilling and resampling (i.e. L2A products).

2.5 Notes on large area mapping

2.5.1 Large region classification

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

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

2.5.2 Eco-climatic stratification

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

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

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

The stratum shapefile is used as follows:

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

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

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