Sentinel-2 Agriculture

Design Definition File

Algorithm Theoretical Basis Document for L4 crop type product

Milestone	Milestone 2
Version	1.0
Authors	CESBIO - Jordi Inglada, Marcela Arias

Contents

1 **Introduction** 2

2 **Processor components** 2

2.1 Overview of the system . . . 2

2.2 Data preparation . . . 3

2.2.1 Sample selection . . . 3

2.2.2 Linear interpolation and gapfilling . . . 3

2.2.3 Feature extraction . . . 4

2.3 Supervised learning . . . 5

2.3.1 Compute image statistics . . . 5

2.3.2 Training the classifier . . . 5

2.4 Map production . . . 7

2.4.1 Crop type map generation . . . 7

2.4.2 Crop type map validation . . . 7

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,
3. map production.

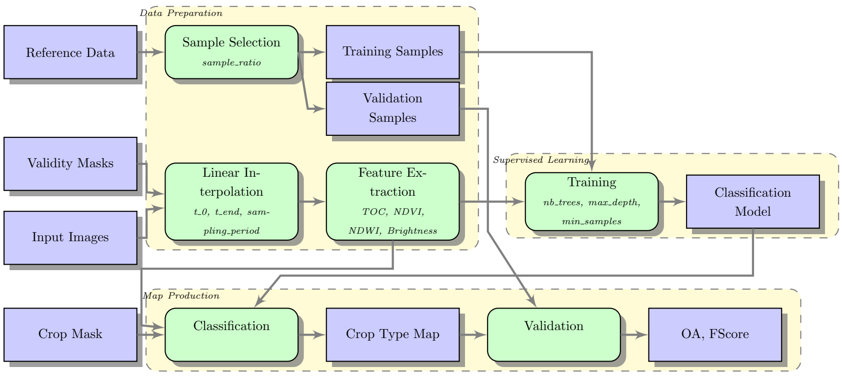

Figure 1: Block diagram of the crop type map production.

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.

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.

[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.

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
`reference_polygons`	Vector file containing reference data
`sample_ratio`	Ratio between the number of training and validation polygons per class
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
  $training\_polygons \leftarrow \emptyset$;
  $validation\_polygons \leftarrow \emptyset$;
  for $cl \in list\_of\_class\_labels$ **do**
    for $poly \in reference\_polygons$ **do**
      if $class\_of(poly) = cl$ **then**
        $p \leftarrow random(0, 1)$;
        if $p \leq sample\_ratio$ **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.

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
`tocr`	S2 L2A surface reflectances
`mask`	Validity masks for each acquisition date
`input_dates`	Dates of each image acquisition
`sampling_period`	Temporal sampling rate
`t_0`	Starting sampling date
`t_end`	Last date
`radius`	radius of the temporal window
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[3].

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.

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

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
`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
`classifier`	Classifier algorithm: `rf` or `svm`
`sample.vtr`	Ratio between the number of training and validation samples
Output variable	role
`confusion-matrix`	File containing the confusion matrix
`model`	File containing the classification model

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
`rf.nbtrees`	number of decision trees
`rf.min`	minimum number of samples per node
`rf.max`	maximum depth of each tree

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
`svm.k`	Type of kernel
`svm.opt`	Automatic optimization of the parameters

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
```