Lab Session: Classification of Satellite Image Time Series with Orfeo ToolBox (OTB)

Objectives:
The main goal of this lab session is to introduce pixel-based supervised classification applications in Orfeo ToolBox. We will use a Sentinel-2 image time series and a crop type dataset. This lab session has the following main objectives:

- understand the preprocessing steps required for the supervised classification procedure of SITS
- understand the OTB applications needed for this task
- use different learning algorithms
- measure the quality of classification results

It is based on a tutorial proposed by the OTB team[1].

The lab session is divided into two main sections. The first section deals with single-date classification, whereas the second one focuses on multi-date classification.

Orfeo ToolBox, Monteverdi and Mapla

Orfeo Toolbox[2] (OTB) is distributed as an open source library of image processing algorithms. OTB encourages full access to the details of all the algorithms implemented: this is not a black-box. Mapla is a launcher of OTB applications, whereas Monteverdi[3] is a satellite image viewer developed to support OTB applications.
Open Mapla and Monteverdi applications (from the `OTB-7.1.0-Win64` folder).

1 Single-date classification

In this section, we will train a classification model from a single (potentially cloudy) image. The process is divided into three main tasks:

1. selecting and extracting samples for training and testing the trained classification models,
2. training the classification model,
3. evaluating the classification model.

[1] `https://www.orfeo-toolbox.org/packages/doc/SummerSchoolSFPT2016/workshopGuide-en.pdf`
[2] `https://www.orfeo-toolbox.org/`
[3] `https://www.orfeo-toolbox.org/CookBook/Monteverdi.html`

1.1 Data

Let us first discover the dataset and the Sentinel-2 image time series we will use during this lab session. The data are available in the `Data/` directory, with the following subdirectories:

- `T30TVT_2017XXXX_Image.tif`: the Sentinel-2 images
- `T30TVT_Mask.tif`: the concatenation of all the masks (**it will be used only on Section 2**)
- `original_dates.txt`: the original dates
- `color_map.txt`: a look up table to display a coloured version of the produced land cover map
- `crop_labels_train`: the training data in the shapefile format
- `crop_labels_test`: the testing data in the shapefile format

The Sentinel-2 image time series comprises of seven images acquired during year 2017 at the following dates:

- 13/03/2017
- 22/05/2017
- 21/07/2017
- 25/08/2017
- 14/10/2017
- 13/11/2017
- 12/08/2017

The Sentinel-2 image time series has been preprocessed by

1. resampling the six spectral bands at a spatial resolution of 20 meters to meters
2. forming a multispectral images for each acquisition date by concatenating the 10 spectral bands (following the order of band names)
3. forminf the mask image `T30TVT_Mask.tif` by concatenating the masks of all Sentinel-2 images (it will be used for the gapfilling operation)

Step 1.1: Select one Sentinel-2 image and open it with QGIS or Monteverdi.
Step 1.2: Display this image in false color (NIR, R, G) and in true color (R,G,B). What are the band numbers that you used?
Step 1.3: Use the `ReadImageInfo` OTB application to determine the spatial resolution and the extent of the image.

1.2 Sample selection and extraction

Let us continue by selecting the samples that will be used to train a supervised classification algorithm and to measure the performance of our framework.

Estimation of sample statistics

The first step of the framework is to know how many samples are available for each class in the Sentinel-2 image. The `PolygonClassStatistics` will do this job for you. This application processes a set of training geometries and an image, and it outputs statistics about available samples (*i.e.*, pixels covered by the image), in the form of a XML file that includes the number of samples per class and the number of samples per geometry.
This application supports the following geometries: polygons, lines and points. Depending on the geometry type, this application behaves differently:

- polygon: select pixels whose center falls inside the polygon
- lines: select pixels intersecting the line
- points: select closest pixel to the provided point

We traditionally use **polygon geometries** when dealing with crop type mapping.
The application requires the input image, but it is only used to define the footprint in which samples will be selected. You can also provide a raster mask, that will be used to discard pixel positions, using the parameter `mask`.
The `field` parameter is the name of the field that corresponds to class labels in the input geometries.
Step 1.4: Open the available shapefile and identify the name of the field that will be used as class labels?
Step 1.5: Run the `PolygonClassStatistics` application. Have a look to the XML file. What is the number of classes? What is the majority class? How many instances of buckwheat is there in the selected area?

Sample selection

We now know exactly how many samples are available in the image for each class and each geometry in the training set. From these statistics, we can compute the sampling rates to apply for each class, and perform the sample selection. This will be done by the `SampleSelection` application.
There are several strategies to compute those sampling rates:

- **constant**: all classes will be sampled with the same number of samples, which is user-defined.
- **smallest class**: the class with the least number of samples will be fully sampled. All other classes will be sampled with the same number of samples.
- **percent**: each class will be sampled with a user-defined percentage (same value for all classes) of samples available in this class.
- **total**: a global number of samples to select is divided proportionally among each class (class proportions are enforced).
- **take all**: take all the available samples.
- **by class**: set a target number of samples for each class. The number of samples for each class is read from a CSV file.

To actually select the sample positions, there are two available sampling techniques:

- **random**: randomly select samples.
- **periodic**: sample periodically.

The application will make sure that samples span the whole training set extent by adjusting the sampling rate. It accepts as input the input image and training geometries, as well class statistics XML file computed during the previous step. It will output a vector file in the **sqlite** format containing point geometries, which indicate the location of the samples. The csv file written by the optional `Output rates` parameter sums-up what has been done during sample selection. Try it!
Step 1.6: In your opinion, what is the best choice for the sampling configuration (sampling type and sampling strategy)?
Step 1.7: Run the `SampleSelection` application. Have a look to the produced file on QGIS.

Sample extraction
Now that the locations of the samples are selected, we will attach measurements to them. This is the purpose of the `SampleExtraction` application. It will walk through the list of samples selected at the previous step and extract the underlying pixel values (in machine learning, this represents the feature vector X). It will also associate each sample with a label (the label vector Y).
Features will be stored in fields attached to each sample in a **sqlite** file. Field names can be generated from a prefix sequence of numbers (*i.e.*, if prefix is `feature_` then features will be named `feature_0`, `feature_1`, ...). This can be achieved with the `Output field prefix (Use a prefix and an incremental counter)` option. Alternatively, one can set explicit names for all features using the `Output field prefix (Use the given name list)` option.
Step 1.8: Run the `SampleExtraction` application. This operation might take few seconds. Have a look to the results, especially the attribute table.
Step 1.9: You have now built your training set. Repeat the three last operations (`PolygonClassStatistics`, `SampleSelection`, and `SampleExtraction`) to create the testing set. Do you think that you should apply the same sampling configuration, which you previously used for the training set?

1.3 Training a model

We will now train a classification algorithm by feeding it with the training set (features X and labels Y). The testing set will be used to quantitatively evaluate the performance of the classifier. Let us follow the following steps:

1. Compute the image statistics
2. Train the model

As we will perform several classification, it is recommended to work in a new subdirectory for each classification task, *e.g.* `sdate_20171208_svm`.
Image statistics
Some machine learning algorithms converge faster if the range of features is $[-1, 1]$ or $[0, 1]$. Other will be sensitive to relative ranges between feature, *e.g.* a feature with a larger range might have more weight in the final decision. This is for instance the case for machine learning algorithm using the Euclidean distance at some point to compare features (*e.g.*, the k-Nearest Neighbour algorithm). In those cases, it is advised to normalize all features to the range $[-1, 1]$ before performing the learning. For this purpose, the `ComputeImageStatistics` application allows to compute and output to an XML file the

mean and standard deviation based on pooled variance of each band for one or several images.
The output file, which contains the statistics, can then be fed to training and classification OTB applications.
Step 1.10: Run the `ComputeImageStatistics` application. What is the mean and the standard deviation for the red spectral band of your image?
Training the model
Now that the training samples are ready, we can perform the learning using the `TrainVectorClassifier` application.
The `Classifier to use for training` tab allows you to choose which machine learning model algorithm to train. You can test here one of the following classifier:

- the k Nearest Neighbors (kNN) classifier. (You can use the default value for the number of neighbors k.)
- the LibSVM classifier with a Linear kernel. You need ticking `ON` the parameters optimization – it will run a cross-validation procedure to estimate the best hyperparameter setting.
- the LibSVM classifier with a Gaussian radial basis function (RBF) kernel. You also need tuning hyperparameters.
- the Random Forest classifier. (Change the maximum depth from 5 to 25.)

Step 1.11: Run the `TrainVectorClassifier` application with the classifier of your choice. This operation might take a couple of minutes.
Step 1.12: Look at the `log` tab. What is your train Overall Accuracy? What do you think of this training?

1.4 Testing phase

Now that we have trained our first classification model, we will evaluate its performance by performing visual and quantitative analysis.
Using the classification model
Let us first apply the trained model to classify all the pixels on the Sentinel-2 image using the `ImageClassifier` application.
Step 1.13: Run the `ImageClassifier` application. According to you, what are the outputs of `confidence map` and `probability map` options?
The output of this step is a GeoTiff image, which associated with each pixel a class. To view this image, the `ColorMapping` application allows us to map a RGB color and generate a new "color mapped" visualisation image. Use the `ColorMapping` application with the proposed color map: `color_map.txt`.
Step 1.14: Run the `ColorMapping` application. Open the resulting tiff image in QGIS / Monteverdi. What do you think of this result?

Evaluating the classification model
The OTB training applications provides information about the performance of the generated model.
With the `ComputeConfusionMatrix` application, it is also possible to estimate the performance of a model from a classification map generated with the `ImageClassifier` application. This labelled image is compared to positive reference samples (either represented as a raster labelled image or as a vector data containing the reference classes). It will compute the confusion matrix and precision, recall and F-score of each class too.
Step 1.15: Run the `ComputeConfusionMatrix` application. What is your test Overall Accuracy?

2 Multidate classification

The quality of the first classification on a single image is not very good. We will try to improve the discrimination of vegetation classes by using the time series. The principle behind multidate classification is to use for each pixel, all the available dates as more spectral bands. With the provided Sentinel-2 data, we will therefore have 70 bands per pixel. The dates have been chosen to be spread over a year, taking into account seasonal variations which has a lot of information to discriminate some classes.

We will first end the preprocessing steps on the Sentinel-2 images by preparing the stack of Sentinel-2 images and by applying the gapfilling operation. We will also compute the Normalized Difference Vegetation Index (NDVI) for each image in the series. Finally, we will repeat previous operations to perform a multi-date classification.

2.1 Preprocessing of Sentinel-2 data

Concatenation

The steps are the same as before, but we will first create a new image, which concatenates the spectral information of the seven Sentinel-2 images.
Step 2.1: Run the `ConcatenateImages` application to create this image.
Step 2.2: Run the `ReadImageInfo` application to check the image size (extent and number of channels) and the image spatial resolution.

Gapfilling

We will now run a linear temporal interpolation to reconstruct missing values that occur due to the presence of clouds or edges in the image.
Step 2.3: Run the `ImageTimeSeriesGapfilling` application. It takes as inputs the cube of images, the masks and the input date file. It ouputs the gapfilled image. It requires the setting of the option `Number of components per date`.

NDVI temporal profiles

We will also compute the NDVI temporal profiles for each pixel. NDVI profiles can be given as additional information to train the classifier model in order to improve the quality of the classification.
Step 2.4: Compute an NDVI image for each Sentinel-2 image by running the `BandMath` application or the `RadiometricIndices` application. (Beware of the output type.) Read the OTB documentation to figure out the format of the expression.
Step 2.5: Stack the NDVI images to the gapfilled image. You should now have an image, which is composed of 77 bands (70 bands for the gapfilled image + 7 bands for NDVI temporal profiles).

2.2 Classification

Classifying the gapfilled SITS data

Step 2.6: Repeat operations of Section 1 in order to classify the gapfilled SITS data.

Classifying NDVI alone

Step 2.7: Repeat operations of Section 1 in order to classify the NDVI temporal profiles alone.

Classifying gapfilled SITS enriched with NDVI profiles.

Step 2.8: Repeat operations of Section 1 in order to classify the gapfilled SITS enriched with NDVI profiles.

3 Assignment

You will write a report, which presents the main findings of this lab session. You will then select one of the following topic:

- Is it better to classify raw SITS or gapfilled SITS?*[4]
- Show that shuffling the order of the bands in the datacube will not modify the classification performance.*
- What is the results of training a classifier on the three first dates and using it for classifying for the three next dates?**
- Can you evaluate the variance of the trained classification models?**
- What is the result of classifying SITS augmented by several spectral, spatial and/or temporal features?***
- Can you find a subset of spectral bands and/or NDVI images that improves the classification performance compare to the use of the SITS augmented with NDVI profiles?***

The objective is to present (1) the used framework to answer to one of these questions, and (2) results and some perspectives on these results.

[4]The number of stars corresponds to the level of difficulty: * medium, ** advanced, *** challenging.