

Labwork Remote Sensing

How to use and to extract information from remote sensing images for land management ?

Mathieu Fauvel

September 20, 2017

Contents

1	Introduction	2
1.1	Objectives of the labworks	2
1.2	Remote sensing software	3
1.3	Sequences	4
1.4	During the labworks	4
2	Data sets	4
2.1	Pleiades images	4
2.2	Formosat 2 Satellite image time series	5
2.3	Historical Maps	5
3	Visualization of remote sensing data	6
3.1	Vizualization of remote sensing image	6
3.2	Get data information	7
4	Spectral indices: <i>Normalized Difference Vegetation Index</i>	8
5	Segmentation of remote sensing images	9
5.1	Radiometric analysis	9
5.2	Segmentation of 1D histogram	9
5.3	Graphical Modeler	9
6	Change detection: <i>Detection of floods</i>	10
7	Classification of remote sensing images	11
7.1	Introduction	11
7.2	Getting started with OTB	12
7.3	Automatize the process with scripts (shell, python or the Graphical modeler)	13
7.4	Influence of the spatial distribution of the learning samples	14
8	Dynamic Habitat Index	15
8.1	Introduction	15
8.2	Construction of the SITS	16
8.3	Computation of the dynamic indices	16
8.4	Characterization of ecozones	17
9	Python for Remote Sensing data analysis	17
9.1	Template filters	17

9.2	Historical map processing	19
9.3	QGIS Script	20
10	Appendix	21
10.1	Short introduction to shell	21
10.2	Short introduction to Python	23

1 Introduction

1.1 Objectives of the labworks

The main objective of these labworks is to

be able to use and to extract information from remote sensing images for land management.

Information can be any knowledge of a given landscape (landcover, land-use, humidity, ...) that is used to understand the configuration and/or the evolution of landscape.

In terms of *competences*, you should be able to master at the end of the sessions the items listed in tables 1, 2, 3, 4 and 5. Each of them is organized as a set of tasks that should be mastered progressively.

Table 1: Choose images with properties adapted to your problematic.

<i>Remember</i>	Properties of remote sensing images.
<i>Understand</i>	Physical meaning of each sampling of a given image.
<i>Apply</i>	Open and visualize a remote sensing image, extract its properties.
<i>Analyze</i>	Describe a remote sensing image. Recognize specific object.
<i>Evaluate</i>	Choose the good image adapted to what you are looking for.
<i>Create</i>	Create a set of properties needed for your problematic.

Table 2: Compute and use spectral indices.

<i>Remember</i>	Definition of spectral indices in general and of the NDVI in particular.
<i>Understand</i>	What (and why) does the NDVI emphasize.
<i>Apply</i>	Perform the computation of spectral indices.
<i>Analyze</i>	Analysis of the vegetation cover using NDVI.
<i>Evaluate</i>	Choose the right spectral index.
<i>Create</i>	Select from the literature a set of possible indices.

Table 3: Define, identify and analyze radiometric behavior.

<i>Remember</i>	Spectral signature for <i>vegetation</i> and <i>water</i> object.
<i>Understand</i>	Histogram of images.
<i>Apply</i>	Compute an histogram.
<i>Analyze</i>	Extract radiometric properties of some classes.
<i>Evaluate</i>	Choose relevant spectral bands and/or indices for the segmentation of several classes.
<i>Create</i>	Perform a segmentation using radiometric statistics on one or many spectral variables and/or indices.

Table 4: Perform and discuss pixel-wise classification of image

<i>Remember</i>	Definition of pixel-wise supervised classification.
<i>Understand</i>	The parameters of several classification algorithm, how the spatial sampling of the ground truth data influence the training and validation steps.
<i>Apply</i>	Classification algorithms.
<i>Analyze</i>	Interpret the confusion matrix and the thematic map, the quality of a ground truth.
<i>Evaluate</i>	Compare two classification maps/results.
<i>Create</i>	Choose the most appropriate classifier for one given application, build good ground truth data.

Table 5: Define and implement a processing chain

<i>Remember</i>	How to combine several bands, apply a given function to train a classifier or to predict a thematic map.
<i>Understand</i>	The different inputs and outputs of OTB functions, how to use their corresponding documentation.
<i>Apply</i>	Apply a set of different functions in a pipeline.
<i>Analyze</i>	Define the different processing needed to perform a given task.
<i>Evaluate</i>	Evaluate the accuracy of the given processing, check for errors.
<i>Create</i>	Shell scripts that automatize most the processes, in order to apply them on a large set of images or to apply several embedded processes.

1.2 Remote sensing software

In these labworks, **free and open sources softwares** will be used to visualize remote sensing images, to process them and to implement processing chains. In the following, each software/tools will be briefly described. Interested reader can find more information on the associated website. In particular, the installation process is not detailed. However, they can be freely download and installed on many operating systems from their official website.

Students from the MASTER SIGMA - Specialization *Agrogéomatique* (A. Greco) has made a youtube channel to help you in using/installing the different softwares:

https://www.youtube.com/channel/UCcxj3jqQVu3w4y13971_jKQ

1.2.1 Orfeo ToolBox (OTB)

OTB is a C++ library for remote sensing images processing. It has been developed by the **CNES** (French space agency) during the ORFEO program to *prepare, accompany and promote the use and the exploitation of the images derived from Pleiades satellites (PHR)*. Processing tools from OTB are appropriated to big images. When possible, processes are paralyzed and tiled automatically for users. Many applications derived from OTB and called *OTB-Applications* are directly usable for most of the common processing, they are described [here](#). For advanced users, it is possible to develop program based on the OTB library (not considered in these labworks).

Monteverdiz is graphical user interface that allows users to visualize and process remote sensing images with *OTB-Applications*. It is also developed by the CNES during the ORFEO program.

1.2.2 QGIS

QGIS is a *Geographic Information System (GIS)*. It is used to open, visualize and process digital map. It includes several spatial analysis tools working mainly on vector data. QGIS can be extended by several plugin (<https://plugins.qgis.org/>) and modules, such as the OTB applications.

1.2.3 Geospatial Data Abstraction Library (GDAL)

GDAL is a library for the processing of raster and vector data. Similar to OTB, it has several applications that can be used directly. For advanced users, it is possible to develop program based on the GDAL library (not considered in these labworks).

1.2.4 Python

Pyhton is a programming language. It has several programming capabilities, such as *object-oriented*, *functional programming*, *dynamic type* and *memory management* that make it widely used in several applications:

- Web and internet development,
- Scientific and numeric computing,
- Software development.

It has a large number of available packages that can be used in many applications. For instance, it is possible to call *OTB-Applications* or *GDAL* from Python.

1.3 Sequences

Table 6: Sequences

Sequences	Type	Volume	Topics
[2017-09-20 Wed 13:30]–[2017-09-20 Wed 17:30]	CTD	04:00:00	Visualization of remote sensing data + Spec
[2017-09-22 Fri 13:30]–[2017-09-22 Fri 17:30]	CTD	04:00:00	Segmentation of RS images + Floods detec
[2017-09-25 Mon 13:30]–[2017-09-25 Mon 17:30]	CTD	04:00:00	Classification of RS images
[2017-09-27 Wed 13:30]–[2017-09-27 Wed 15:30]	CTD	02:00:00	Classification of RS images
[2017-09-27 Wed 15:30]–[2017-09-27 Wed 17:30]	Project	02:00:00	Spatial distribution of pixels
[2017-09-29 Fri 13:30]–[2017-09-29 Fri 16:30]	CTD	03:00:00	Dynamic Habitat Index
[2017-10-02 Mon 08:00]–[2017-10-02 Mon 10:00]	EXAM	02:00:00	EXAM Groupe 1 & 2
Total		21:00:00	

1.4 During the labworks

For the *presentiel* sequences, you won't have to do any report. But you will have to write your personal material on remote sensing. You are encouraged to write it progressively during the sessions. **It will be the only document approved for the exam** (with those on moodle). The length of each sequence should let you enough time to write the report.

For the *non presentiel* sequences, you will be asked to write a document that describe briefly the results and how you obtained them. Discussion between all groups will be done during the next session.

2 Data sets

2.1 Pleiades images

These images were acquired over the Fabas forest in 2013. Images were acquired the <2013-10-12 Sat> and the <2013-12-10 Tue>, respectively. A true color composition is given in Figure 1.



Figure 1: Fabas image acquired the [2013-10-12 Sat].

Images are stored using the [GeoTIFF](#) format. It is an extended version of the TIFF format, which allows to embed geospatial information within the file. GeoTIFF can be read by most of the remote sensing and GIS software. Table 7 gives the band order of the data.

Table 7: Bands and channels information for the Pleiades images

Band	Channel
1	Red
2	Green
3	Blue
4	Infra-red

2.2 Formosat 2 Satellite image time series



Figure 2: Formosat 2 image acquired the [2012-05-03 Thu].

This time series was acquired in 2012 over the region of *Saint Lys*. It consists in a set of [Formosat-2](#) images along the year 2012. Figure 3 provide information about the acquisition date and the Figure 2 shows a false colors composition for the date [2012-05-03 Thu]. Table 8 gives the band order of the data.

2.3 Historical Maps

The figure 4 shows an historical map. This is a scan performed by the [IGN](#) of an old manually drawn map.

Table 8: Bands and channels information for the Formosat images

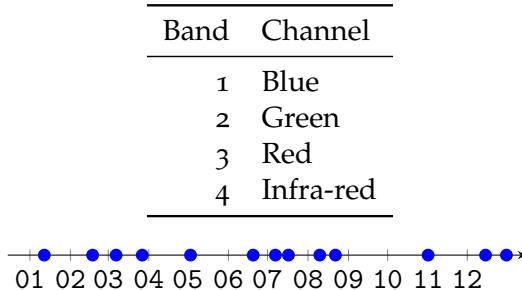


Figure 3: Acquisition dates for the SITS 2012.

3 Visualization of remote sensing data

3.1 Vizualization of remote sensing image

The vizualisation of remote sensing images can be done either with Monteverdi2 or QGIS¹. QGIS might be a more efficient when it comes to visualize several images, or for the vizualisation of vector layers. It will be used in these labworks.

Most of the information regarding the vizualisation of raster data with QGIS can be found online http://docs.qgis.org/2.14/en/docs/user_manual/working_with_raster/raster_properties.html.

More generally, to use raster data with QGIS is described here http://docs.qgis.org/2.14/en/docs/user_manual/working_with_raster/index.html.

In this labwork, a few properties will be reviewed and you are encouraged to check (at least) the given references.

3.1.1 Vizualization of grayscale image

Open the image *fabas_10_12_2013.tif* with QGIS. The default view is a colour composition, with the bands/channels association given in Table 9. To start easy, we just open one band at a time: right click on the name of the opened image in the *Layer* pane et select *Properties*. Then select the tab *Style* and *Band rendering*. In the *render type*, select *Singleband gray* and the band you want to display.

You surely have to do *Contrast enhancement*. Check the doc for that.

¹The library `matplotlib` of python is not adapted to visualize remote sensing image and should be avoided.

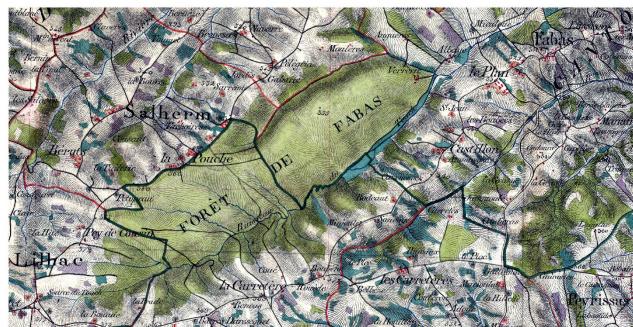


Figure 4: Historical Maps

Table 9: Bands and channels default association in QGIS (if there is not a set of specified spectral bands in the metadata).

Band	Channel
1	Red
2	Green
3	Blue

Work 3.1:

1. Visualize each spectral band of the data, and look at the differences in terms of graylevel between spectral bands.
2. Zoom in/out: use the mouse's wheel to zoom into the image. What do you observe ?

3.1.2 Visualization of colour image

Now you can visualize a colour images, by selecting three spectral bands among those available from the data. Again, *Contrast enhancement* should be done.

Work 3.2:

1. Do a "true colours" and "false colours" compositions and compare what is easily seen on each of them.
2. Get spectral values for several pixels corresponding to different materials (water, grassland, forest and bare soil). For that, use the tool *Identify features*, see http://docs.qgis.org/2.14/en/docs/user_manual/introduction/general_tools.html for detail.
3. Fill the *collaborative spreadsheet* with your pixel values:
 - https://framacalc.org/fauvel_rs_water
 - https://framacalc.org/fauvel_rs_grassland
 - https://framacalc.org/fauvel_rs_forest
 - https://framacalc.org/fauvel_rs_baresoil

3.2 Get data information

Before opening a remote sensing data, it is possible to get some information about its properties. For instance, using `gdalinfo` it is possible to extract several information. It can be used as

```
gdalinfo fabas_10_12_2013.tif
```

Help on the function can be obtained using the command alone or by doing :

```
man gdalinfo
```

Equivalently, it is possible to get the same information using the function `otbcli_ReadImageInfo` from the *OTB-Applications*:

```
otbcli_ReadImageInfo -in fabas_10_12_2013.tif
```

Work 3.3:

On the *Fabas* data set, get the following information.

1. Number of lines, columns and bands,
2. Size of each pixel,
3. Numerical types for coding pixel values,
4. Position of the upper left pixel,
5. Projection.

4 Spectral indices: *Normalized Difference Vegetation Index*

Among the available radiometric indices, only the NDVI is considered in this labwork. NDVI is widely used for vegetation monitoring because it can be related to chlorophyll content and photosynthesis.

Work 4.1:

1. Compute the NDVI for each *Fabas* image. You can compute the NDVI using several ways, using either *OTB-Applications* or the *Raster Calculator* http://docs.qgis.org/2.14/en/docs/user_manual/working_with_raster/raster_analysis.html#raster-calculator. For a per band analysis, both methods are equivalent. Using QGIS provides the Graphical user interface, which can be convenient for processing few images, while *OTB-Applications* allow to process large number of images using *shell* programming.

Using the raster calculator, the following formula can be used (for the Fabas image):

```
("fabas_10_12_2013@4"- "fabas_10_12_2013@1")/("fabas_10_12_2013@4"+ "fabas_10_12_2013@1")
```

Using the *OTB-Applications*, it is possible to use *otbcli_BandMath*. The syntax is similar, since we need to define the image, the bands used and the expression of our processing:

```
otbcli_BandMath -il fabas_10_12_2013.tif -out ndvi_fabas.tif -exp "(im1b4-im1b1)/(im1b4+im1b1)"
```

2. Compare the NDVI obtained for each date and explain your results.

5 Segmentation of remote sensing images

5.1 Radiometric analysis

Work 5.1:

For the NDVI of the image [2013-10-12 Sat], do

1. Look at the histogram and identify the local maxima. For each local maximum, try to identify the corresponding pixels in the image,
2. Keep track of the characteristics of each identified maximum (position and width).

5.2 Segmentation of 1D histogram

In this part, the extraction of image's pixels sharing the same *radiometric behavior* is considered. The analysis of the histogram is used to estimate this *behavior*. When only one material is segmented, the output is a binary image (image with value 0 or 1), where pixels having value 1 are from the same material. Figure 5 gives an example of such outputs. When several material are considered, the output is an images with integer values (1, 2, 3 ...), depending on the number of materials.



Figure 5: Binary image for Water.

A usual work-flow is proposed in this part. First, QGIS is used to analyze the data and set-up the processing (parameters *etc*). Then, the *OTB-Applications* are used to automatize the processing.

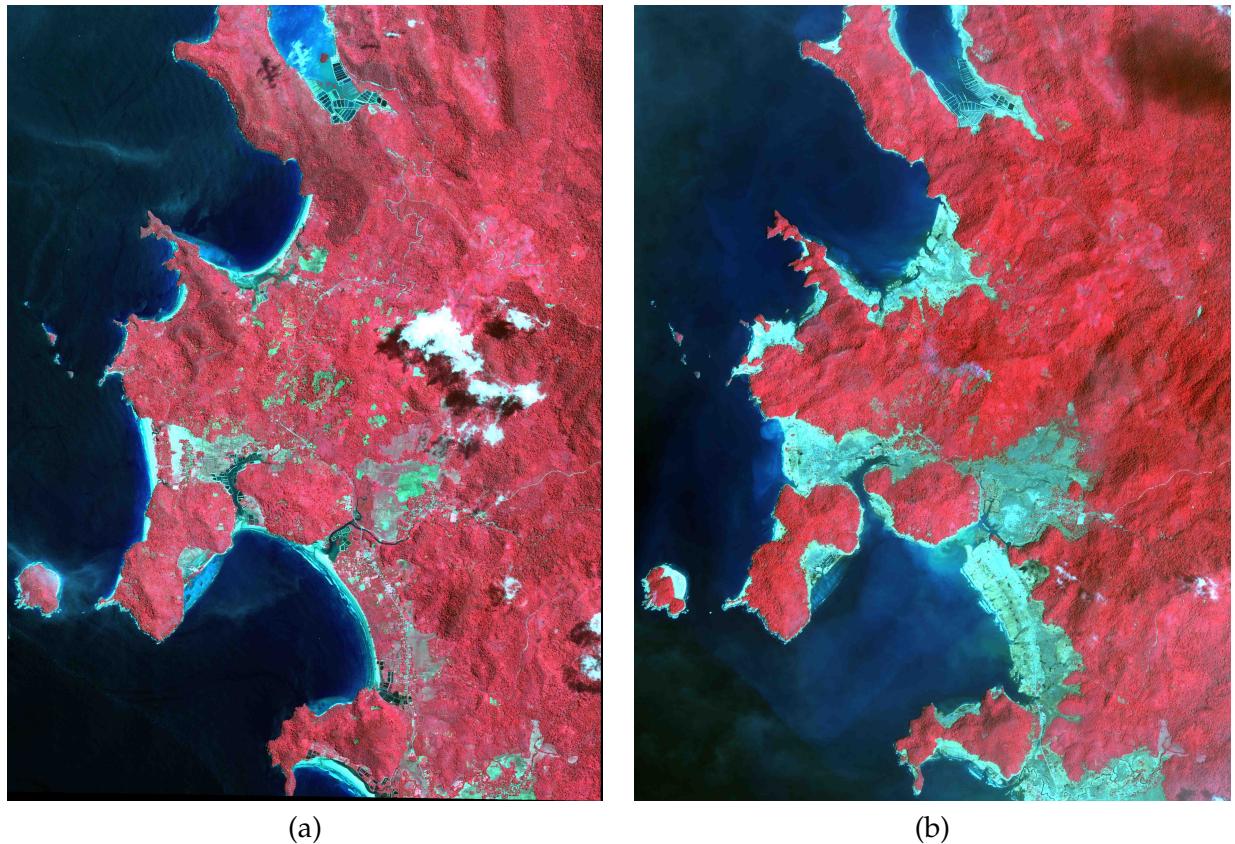
Work 5.2:

Segment the identified material on the NDVI. For that, you need to define interval of pixel values for which a specific action is done (*e.g.*, set the value to 0 or 1). Implement the processing using the BandMath application.

5.3 Graphical Modeler

For the segmentation of the NVDI, two processings are required

1. First, the computation of the NDVI from the original image,



(a)

(b)

Figure 6: False colours images of Lhonga Leupung area (a) before and (b) after flooding.

2. Second, the definition of the interval of values to extract the relevant pixels.

With the graphical modeler, it is possible to define your workflow, to automatize complex tasks. Take a look at http://docs.qgis.org/2.14/en/docs/user_manual/processing/modeler.html.

Work 5.3:

Define your model to perform the segmentation of intro three classes of the NDVI.

6 Change detection: *Detection of floods*

Change detection in remote sensing consists in detecting differences between two images, or a set of images. It can be used to detect changes in vegetation properties or in land cover. It is also used in disaster management, to detect impacted areas. In this labwork, we are dealing with floods. In Figure 6 is shown two quickbird images, before and after a flooding. The objective is to identify the impacted area to provide a map of these zones

Work 6.1:

1. Characterize the impacted zones in terms of radiometric behavior, *i.e.*, what is the variation in terms of spectral values. And why ?
2. Define the processing chain to extract these areas.
3. Implement the processing chain with the graphical modeler.
4. Optional: Implement the same processing chain with shell scripts, see [10.1](#).



Figure 7: Google view of the impacted area. The red square represents the area of Figure 6.

7 Classification of remote sensing images

7.1 Introduction

The aim of this labwork is to perform the classification of remote sensing images using supervised algorithms. The principle is the same than segmentation. But now the gray level intervals are not defined manually and the definition of a radiometric behavior is not limited to a rectangular area in the spectral domain. Furthermore, since all the computation are done by supervised algorithms, it is possible to use more information than one or two bands and the full multispectral image can be use. In fact, more than one image can be used. In this work, the two *Fabas* images will be classified: first separately and then conjointly.

The OTB proposes various classifiers, each one having different characteristics. In order to train (or learn) the classifier, some labeled pixels should be provided. It is possible to construct the ground-truth (set of labeled pixels) in different ways:

- Using GIS layer and extract the relevant information at the pixel level.
- Do field survey and use GPS to identify pixels.
- Do photo-interpretation when possible.

In this works, the ground-truth is provided as a vector file, see [8](#). Five classes are considered, they are given in Table [10](#).

During this labwork, it is proposed to compare in terms of classification accuracy and processing time some of the classifiers proposed in OTB and all the combination of input data, *i.e.*:

- K-nn, Bayes, SVM and Random Forest.

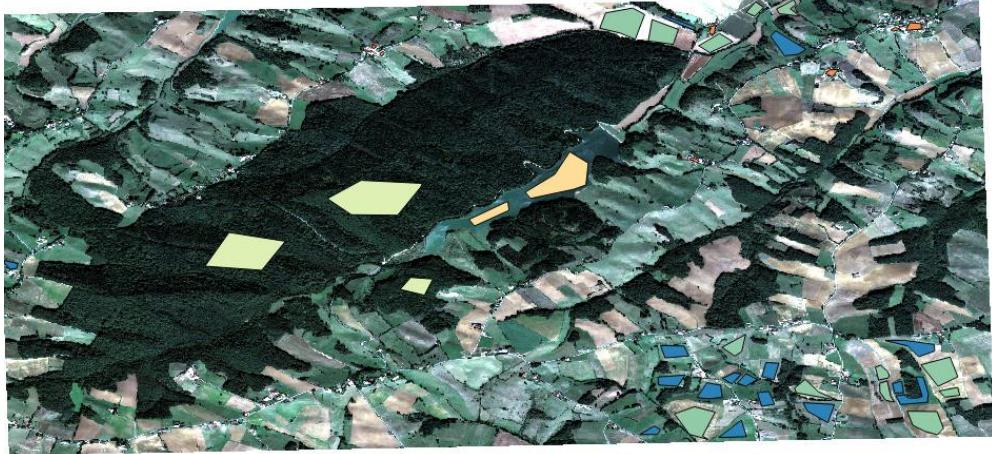


Figure 8: Ground truth for the *Fabas* image.

Table 10: Classes of interest. Numbers corresponding to the attribute in the GIS file is also given.

Classes	Sparse vegetation	Bare soil	Woody vegetation	Water	Built up
Attribute	1	2	3	4	5

- The ground-truth being composed of pixels from one date, and two *concatenated* dates.

7.2 Getting started with OTB

There are several steps to do a classification.

1. *Learn the classifier:* It is done with `TrainImagesClassifier`. It takes as inputs, the (set of) remote sensing image(s), the ground-truth (in vector format), and some parameters of the method. To learn the classifier, only the pixels inside the ground-truth are used. After this step, a *model* that contains the parameters is saved. If asked, a confusion matrix is computed.
2. *Classify the image:* Once the classifier is learned, it is possible to apply the model to all the pixels of the image. It can be done with `ImageClassifier`.
3. Compute the accuracy of the thematic map according to some groundtruth. **This groundtruth should not be spatially correlated with the one used for training.** The confusion matrix can be computed using the function `ComputeConfusionMatrix`.

Work 7.1:

This should be done for one image and one classifier only.

1. Learn the model,
2. Apply the model to classify the entire image,
3. Compute the confusion matrix and save it in a *csv* file.
4. Open the CSV using a spreadsheet. From the confusion matrix, compute the following indices:
 - Global accuracy,
 - Producer accuracy,
 - User accuracy.

7.3 Automatize the process with scripts (shell, python or the Graphical modeler)

It is possible to run directly the *OTB-Applications* from the command line. This way, it is possible to run several operations on one data set or on several data sets automatically. A brief introduction to command line tools is given in Appendix [10.1](#).

The three previous *OTB-Applications* are available from the command line interface (CLI), same name with the prefix `otbcli_`:

- `otbcli_TrainImagesClassifier`,
- `otbcli_ImageClassifier`,
- `otbcli_ComputeConfusionMatrix`.

The same inputs than in QGIS should be provided (*raster and vector file, algorithm parameters ...*). For instance, if you are in the directory where the data are, learning the KNN classifier with default parameters do the following, classifying the whole image and computing the confusion matrix reduce to

```
otbcli_TrainImagesClassifier \
    -io.il fabas_12_10_2013.tif \
    -io.vd train_fabas.shp \
    -classifier knn \
    -io.out model.mod
otbcli_ImageClassifier \
    -in fabas_12_10_2013.tif \
    -model model.mod \
    -out fabas_classif.tif
otbcli_ComputeConfusionMatrix \
    -in fabas_classif.tif \
    -out matconf.csv \
    -ref vector \
    -ref.vector.in valid_fabas.shp
```

This is nothing else than what you provide in QGIS ! In the following, we are going to combine Python scripts and the OTB Applications to define our processing chain. Two python modules will be used: `os` and `glob`. These modules are very convenient to manage files, folder and to launch applications. Also, we are going to benefit Python abilities to process strings.

Let's start with an example, to run the first application

```
# Load the module
import os

# Launch the application
os.system('otbcli_TrainImagesClassifier -io.il fabas_12_10_2013.tif -io.vd train_fabas.shp -classifier knn -io.out matconf.csv')
os.system('otbcli_ImageClassifier -in fabas_12_10_2013.tif -model model.mod -out fabas_classif.tif')
os.system('otbcli_ComputeConfusionMatrix -in fabas_classif.tif -out matconf.csv -ref vector -ref.vector.in valid_fabas.shp')
```

or equivalently:

```
# Load the module
import os

# Define processing
train = 'otbcli_TrainImagesClassifier -io.il fabas_12_10_2013.tif -io.vd train_fabas.shp -classifier knn -io.out matconf.csv'
classify = 'otbcli_ImageClassifier -in fabas_12_10_2013.tif -model model.mod -out fabas_classif.tif'
validate = 'otbcli_ComputeConfusionMatrix -in fabas_classif.tif -out matconf.csv -ref vector -ref.vector.in valid_fabas.shp'

# Launch the application
os.system(train)
os.system(classify)
os.system(validate)
```

Additional useful references are given section [10.2](#), take the time to read them.

Alternatively, it is possible to use the *batch processing interface* provided by the graphical modeler of QGIS. It allows the batch execution of a model over several inputs. Take a look at the following link for further details:

https://docs.qgis.org/2.14/en/docs/user_manual/processing/batch.html?highlight=batch

According to your preference, use one of these techniques to do the following tasks:

Work 7.2:

1. Write the script/modeler to learn the model for all the classification methods and with each date. Each time extract the confusion matrix and compute the global accuracy and the class average accuracy.
2. Report the results on the *collaborative spreadsheet*: https://framacalc.org/fauvel_res_classification
3. For the best method in terms of classification accuracy, discuss about the errors obtained with the confusion matrix.
4. Classify the whole image and compare by visual inspection the errors with what you have inferred from the confusion matrix.

7.4 Influence of the spatial distribution of the learning samples

In order to evaluate the influence of the validation samples, you will investigate several reference layers to compute the confusion matrix. Since OTB only select a few samples from all the available one (can be controlled with the options `samples.mt` and `samples.mv`), we need to repeat the experiment several times, to avoid bias.

Select one classifier for all the experiments. You are encouraged to define a python/shell script or a modeler.

```

import scipy as sp
import glob
# get all the csv files that match the pattern and order the list in increasing order
NAMES_TRAIN,NAMES_VALID = glob.glob('confu_train_*.csv'),glob.glob('confu_valid_*.csv')
NAMES_TRAIN.sort()
NAMES_VALID.sort()

oa_train,oa_valid = [],[]

for name_train,name_valid in zip(NAMES_TRAIN,NAMES_VALID):
    temp = sp.genfromtxt(name_train,delimiter=',',skip_header=2) # read the file, skip the two first lines (of comm
    oa = 100*sp.diag(temp).sum()/temp.sum() # Compute the overall accuracy
    oa_train.append(oa) # add the values to the list
    temp = sp.genfromtxt(name_valid,delimiter=',',skip_header=2)
    oa = 100*sp.diag(temp).sum()/temp.sum()
    oa_valid.append(oa)

# Compute mean accuracy and standard deviation and save the results
res = [[sp.mean(oa_train),sp.std(oa_train)],[sp.mean(oa_valid),sp.std(oa_valid)]]
sp.savetxt('acc.csv',res,delimiter=',',fmt='%1.3f')

```

Figure 9: Sample code to process a set of csv files.

Work 7.3:

Repeat 20 times the following test:

1. Learn with *train_fabas* and compute the confusion matrix with *train_fabas*. Save the confusion matrix for each repetition.
2. Learn with *train_fabas* and compute the confusion matrix with *valid_fabas*. Save the confusion matrix for each repetition.
3. Compute the average global accuracy and the mean class accuracy and their standard deviation. You can check the figure 9 to do it automatically.

Discuss about the results.

8 Dynamic Habitat Index

8.1 Introduction

In this labworks, we are going to compute several indices of habitat dynamic's in order to define several ecozones. It is bases on the following paper:

Nicholas C. Coops, Michael A. Wulder, Dennis C. Duro, Tian Han, Sandra Berry, The development of a Canadian dynamic habitat index using multi-temporal satellite estimates of canopy light absorbance, Ecological Indicators, Volume 8, Issue 5, September 2008, Pages 754-766, ISSN 1470-160X, <http://dx.doi.org/10.1016/j.ecolind.2008.01.007>. (<http://www.sciencedirect.com/science/article/pii/S1470160X08000071>)

These indicators underly vegetation dynamic, they are usually computed in the *fraction of photosynthetically active radiation (fPAR)* absorbed by the vegetation. However these data are not available. So in this lab, the NDVI will be used. The data is described in [2.2](#).

Work 8.1:

The first (easy) part is to convert NDVI values to fPAR like values. Since fPAR is a fraction, its values are between 0 and 1. You have to convert the interval range of NDVI to 0 and 1 using a simple linear function: $f(x) = ax + b$. You have to find a and b !

$$\begin{aligned}f : [-1, 1] &\rightarrow [0, 1] \\x &\mapsto f(x) = ax + b\end{aligned}$$

8.2 Construction of the SITS

Before analyzing the SITS, you need to built it.

Work 8.2:

1. Compute the NDVI for each date,
2. Convert to fPAR-like values,
3. Concatenate all the dates,
4. Using QGIS, plot the temporal profile for several objects.

8.3 Computation of the dynamic indices

The second part of the labwork concern the computation of the dynamic indices. Let us note the vector of fPAR values of pixel i $\mathbf{x}_i = [\mathbf{x}_i(t_1), \dots, \mathbf{x}_i(t_d)]$. Three indices have been defined:

1. The cumulative annual greenness,

$$CG = \sum_{j=1}^d \mathbf{x}_i(t_j)$$

1. The annual minimum cover,

$$MC = \min_j [\mathbf{x}_i(t_1), \dots, \mathbf{x}_i(t_j), \dots, \mathbf{x}_i(t_d)]$$

1. The greenness coefficient of variation.

$$GCV = \frac{\sigma_{\mathbf{x}_i}}{\mu_{\mathbf{x}_i}}$$

Work 8.3:

1. Write the python scripts to compute all indices.
2. Concatenate all the indices into one multiband image.

```

def TheFilter(imin,p1=p1,p2=p2, ...):
    """This function apply the filter TheFilter on image imin with
    parameters p1, p2, ...p2

    Input:
    -----
    imin = image to be processed, Scipy 2D-Array
    p1 = parameter 1 of the filter, default value p1
    p2 = parameter 1 of the filter, default value p2

    Output:
    -----
    imout = filtered image, Scipy 2D-Array, not necessraly of the same
    type as imin, depending of the filter
    """

    ## Some processing

    ## Other processing

    return imout

```

Figure 10: Skeleton of the filter

8.4 Characterization of ecozones

Perform a segmentation of the SITS using the three indices as input values. A primarily study suggests the number of ecozones is 4 for this area. Look at the function `otbcli_KMeansClassification` to perform the automatic segmentation of you data.

Work 8.4:

1. Performs the segmentation with 4 classes and save the values of the estimated centroid.
2. Extract the values of the centroid and interpret their values in terms of habitat.
3. Do a visual validation of your results on the thematic map.

9 Python for Remote Sensing data analysis

9.1 Template filters

9.1.1 Introduction

In this labwork, images will be provided under the Scipy format. How to open remote sensing images with GDAL will be addressed later. To load an image using Scipy just do

```

import scipy as sp
image = sp.load('dataname.npy')

```

In the following, you will have to write python functions (mainly image filters). In figure 10 is provided a skeleton of such function, using simple docstring convention. You are highly encouraged to put comment in your code !

Work 9.1:

1. With template filters, there are a list of pre-processing that needs to be done every time. Find them out. (Here we do not care about *edge effect*: pixels at the border of the image will not be processed)
 - check of inputs,
 - initialization of the output image,
 - ...
2. Complete the skeleton to have a ready-to-use function, where you will just have to implement the operation in the neighborhood. Few lines of codes to scan all the pixel of the images.

9.1.2 Template filters function

Using the previously defined skeleton, implement the following filter: `max`, `min`, `median`, `mean`. You can use methods of `scipy array` class describe below:

<https://docs.scipy.org/doc/numpy/reference/generated/numpy.amax.html>
<https://docs.scipy.org/doc/numpy/reference/generated/numpy.amin.html>
<https://docs.scipy.org/doc/numpy/reference/generated/numpy.mean.html>
<https://docs.scipy.org/doc/numpy/reference/generated/numpy.median.html>

Work 9.2:

1. Start with a simple template, a fixed size moving window of 3×3 pixels. Write your function.
2. Modify your function to include an input parameter that control the size of the moving window (only consider odd size).
3. When your function is working correctly, try to improve the processing time. You can use the module `time`, see <https://pymotw.com/2/time/>.
4. Apply your function on the Ikonos images and try to remove the noise.

9.1.3 To go further

- Extend your function for multidimensional images.
- Provide function with a rectangular template (with odd size).
- For the mean filter, add a new parameter that define the number of *cycle*, i.e., the number of times the filter is applied iteratively on the image.
- Implement the following filters and try to explain what are they action.
 - `max(im)-im`
 - `im-min(im)`
 - `max(im)-min(im)`

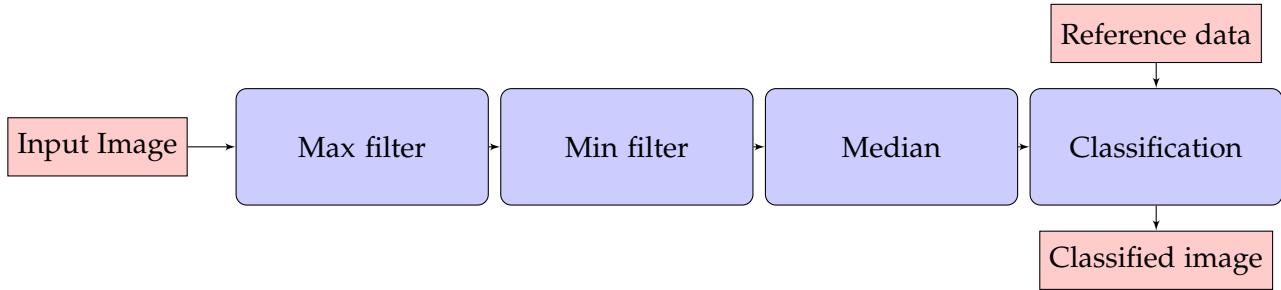


Figure 11: Processing chain.

9.2 Historical map processing

This section is dedicated to the implementation of the filtering part of the following paper

P-A Herrault, D Sheeren, M Fauvel, and M Paegelow. Automatic extraction of forests from historical maps based on unsupervised classification in the cielab color space. In Geographic Information Science at the Heart of Europe, pages 95–112. Springer International Publishing, 2013.

It concerns the filtering of historical maps, see 2.3 to see the data to be processed. The filtering consists in the application of the consecutive filters, on every bands of the color image:

1. Local max filter,
2. Local min filter,
3. Local median filter,

as illustrated in the (simplified) processing chain shown in 4. You should write a script that takes as arguments:

- The name of the image,
- the name of the output image,
- the size of the min/max filter,
- the size of the median filter,

an, off course, do the right processing.

Work 9.3:

1. Write the main steps of the filtering:
 - a) Load the image
 - b) Filter the image
 - c) Write the results
2. Add the necessary machinery to make a python application
 - a) Add command line arguments parser (see 10.2.4)
 - b) Make your function executable

```
chmod +x yourfunction.py
```
3. Play with it to find the best couple of parameters to remove black line in the historical map.

9.3 QGIS Script

QGIS offers facilities to call python scripts directly from QGIS. We are going to see how it is possible to use our previous script, which slight modifications, directly in QGIS. Doing this has two main advantages:

1. Let QGIS does the job for the selection and the visualization of the images,
2. Use QGIS pre-defined function to handle arguments of our functions.

9.3.1 Install *Script Runner*

Script Runner is a QGIS plugin that allows to integrate python script very easily: you don't need to worry about linking your script to QGIS, it is done automatically. To install it, just do *Extensions -> Manage and Install Plugins* and select the plugin.

Once the plugin is installed in QGIS, you can import your scripts.

9.3.2 Creating script for QGIS and *Script Runner*

Your script needs slight modification in order to be imported in *Script Runner*.

1. Add the following modules to interact with QGIS:

```
from PyQt4.QtCore import *
from PyQt4.QtGui import *
from qgis.core import *
from qgis.gui import *
```

2. First you should define a function `run_script(iface)` where the processing is done. `iface` is a python object that gives access to several QGIS objects and classes. It is used to communicate with QGIS. You can also add arguments to the function: when the script is run, a window will be automatically created to ask you to fill the value of each parameter. For instance, if you need to define bands corresponding to infrared and red and the name of the output file:

```
def run_script(iface,IR,R,data_name)
```

3. The second modification need is to get the name of the data of current layer in QGIS. It is done using `iface` object:

```
layer = iface.activeLayer() # Get the current layer
name = layer.dataProvider().dataSourceUri() # Get the path of the layer
```

Then, once the processed file has been saved on the hard drive, it is possible to load directly the file into QGIS, again by using `iface`:

```
raster_lyr = iface.addRasterLayer('name_of_the_data','name_of_the_layer')
```

4. The last modification is not mandatory, but it will simplify the use of your script. Don't use `import`, but directly put all your additional functions before the `run_script` function.

Work 9.4:

Make the your script and use it from QGIS.

10 Appendix

10.1 Short introduction to shell

This section provides an introduction to *shell* programming and *shell scripts*. A script is a set of commands, which allows to write a processing chain for a given image, or to apply one processing to a set of images. Of course, mixing these two situations is possible. You can find more information easily on the web, a good starting point can be the [Wikibook](#).

Shell is a programming language that is available on all GNU/Linux distributions. It can be used directly from the prompt (interactive mode), or by writing a file with a set of commands to be run. This file should start with the line

```
#!/bin/bash
```

In the following, it is assumed that we are working on the file `script.sh`. To insert comment inside the script, the symbol `#` has to be used.

```
# This is a comment
```

With Linux, a file can be *writable*, *readable* and/or *executable*. To be run as a script, it should be at least *executable* by the OS. It can be done by running the following command:

```
chmod +x script.sh
```

To run it, just do

```
./script.sh
```

10.1.1 Basic commands

- **cd**: Change directory. To enter a directory, do `cd Name_Of_Directory`.
- **ls**: List all the files in the current directory.
- **pwd**: Return the name of the current directory.
- **cp**: Copy a file/directory, for instance `cp A B`.
- **mv**: Move a file to another, for instance `mv A B`.
- **mkdir**: Create a directory, `mkdir Name_Of_Directory`.

For instance, to get all the `tif` files in the current folder:

```
ls *tif
fabas_10_12_2013.tif  fabas_12_10_2013.tif
```

10.1.2 Variables

In shell, a variable is a string (not a number). It can be defined as:

```
var1='Mathieu' # Store "Mathieu" in variables "var1"
var2='Fauvel'
var3='34'
```

Be careful to spaces: there are no spaces, otherwise an error is returned! A variable is displayed using the `echo` function and the variable is accessed with the command `$`.

```

echo $var1 $var2      # print "Mathieu Fauvel"
echo "$var3 ans"      # print "33 ans"
echo '$var3 ans'      # print "$var3 ans"

```

```

Mathieu Fauvel
34 ans
$var3 ans

```

Note the difference between the simple quote ' and the double quote ". The simple quote does not evaluate the variable while the double quote does.

It is possible to pass parameters to the script, solely by adding them when the script is called. They are accessible using the command \$ following by the order number of appearance when the script is called. Let define the script.sh file.

```

# ./script.sh Name FamilyName Age
echo $1 $2
echo "J ai (eu) $3 ans !"

```

When we do this, we have the following output:

```

chmod +x script.sh
./script.sh Mathieu Fauvel 33

```

```

Mathieu Fauvel
J ai (eu) 33 ans !

```

10.1.3 Loop

As in any programming language, loop are very useful to apply a series of processing to several elements of a sequence. The example below applies a processing on all *tif* files of the current directory:

```

for i in *.tif # For all tif file
do
    cp $i ndvi_${i} # create a new file and add ndvi_ at the beginning of the filename
done

```

10.1.4 Sequence

It is possible to define sequences of string like this:

```

for name in bayes libsvm knn rf
do
    echo $name
done

```

```

bayes
libsvm
knn
rf

```

Sequences of numbers can be defined like this:

```

for i in `seq 1 5`
do
echo $i
done

```

```
1  
2  
3  
4  
5
```

10.2 Short introduction to Python

A good starting point is the following link: <http://kitchingroup.cheme.cmu.edu/pycse/pycse.html>. Here, I just review few things that are usefull for the labwork. But python is far more than this short introduction.

10.2.1 String

Handling strings with python is very easy. It is possible to add strings together, as with number! Pay attention to spaces...

```
name="Mathieu"  
surname="Fauvel"  
print name + surname
```

MathieuFauvel

To use numbers in strings, it is necessary to convert them, using the function `str`

```
print "Bonjour j'ai eu " + str(33) + " ans"
```

Bonjour j'ai eu 33 ans

10.2.2 Loop

It is very easy to iterate over a list with python. The list can be made of numbers, strings etc ... Since a list is `iterable`, defining a `for` loop is just:

```
listeNumber = [1,2,3,4]  
print listeNumber  
for item in listeNumber:  
    print(item)  
  
listeString = ['knn', 'bayes', 'libsvm', 'rf']  
print listeString  
for item in listeString:  
    print(item)
```

```
[1, 2, 3, 4]  
1  
2  
3  
4  
['knn', 'bayes', 'libsvm', 'rf']  
knn  
bayes  
libsvm  
rf
```

10.2.3 Glob

The `glob` module finds all the path-names matching a given pattern. It uses standard Unix (shell) path expansion rules. However, results are returned in arbitrary order and therefore sometimes ordering operation is necessary. It returns a list of pathnames, or a iterator which can be useful for large processing. Below some examples to see how it works. First, check what is in my `figures` directory:

```
ls figures/
```

```
google_bridge.jpg
label_fabas.jpg
label_fabas.jpgw
old_map.jgw
old_map.jpg
pixel.pdf
quicklook_fabas_10_12_2013.jpg
quicklook_fabas_12_10_2013.jpg
quicklook_seg_eau.png
sits_f2.pgw
sits_f2.png
take5spot5.png
tsunami_after.jpg
tsunami_before.jpg
```

If we want to get all the files, we just need to do

```
import glob

files = glob.glob("figures/*")
for files_ in files:
    print files_

figures/sits_f2.png
figures/quicklook_fabas_12_10_2013.jpg
figures/tsunami_after.jpg
figures/take5spot5.png
figures/sits_f2.pgw
figures/quicklook_fabas_10_12_2013.jpg
figures/pixel.pdf
figures/label_fabas.jpg
figures/tsunami_before.jpg
figures/old_map.jpg
figures/quicklook_seg_eau.png
figures/label_fabas.jpgw
figures/old_map.jgw
figures/google_bridge.jpg
```

If we only want the `png` files:

```
import glob

files = glob.glob("figures/*.png")
for files_ in files:
    print files_
```

```
figures/sits_f2.png  
figures/take5spot5.png  
figures/quicklook_seg_eau.png
```

The iterator is `iglob`, it does the same job than `glob`, but without storing all the results simultaneously.

```
import glob  
  
for files_ in glob.iglob("figures/*.png"):  
    print files_
```

```
figures/sits_f2.png  
figures/take5spot5.png  
figures/quicklook_seg_eau.png
```

10.2.4 Argparse

Argparse (<https://docs.python.org/3.6/library/argparse.html>) is module to parse options and arguments from the command-line interface. It defines what are the mandatory argument, generates help and usages messages and errors at runtime.

For instance, suppose we have a function that needs two parameters: the name of a multispectral file and the size of the template filter. Argparse handles everything:

```
import argparse  
  
# Initialization of the filter  
parser = argparse.ArgumentParser()  
  
# Add arguments  
parser.add_argument("-in", dest="image", help="Image to be processed", type=str)  
parser.add_argument("-p", help="Size of the template", type=int, default=1)  
  
args = parser.parse_args()  
  
print args.image  
print args.p
```

None

1