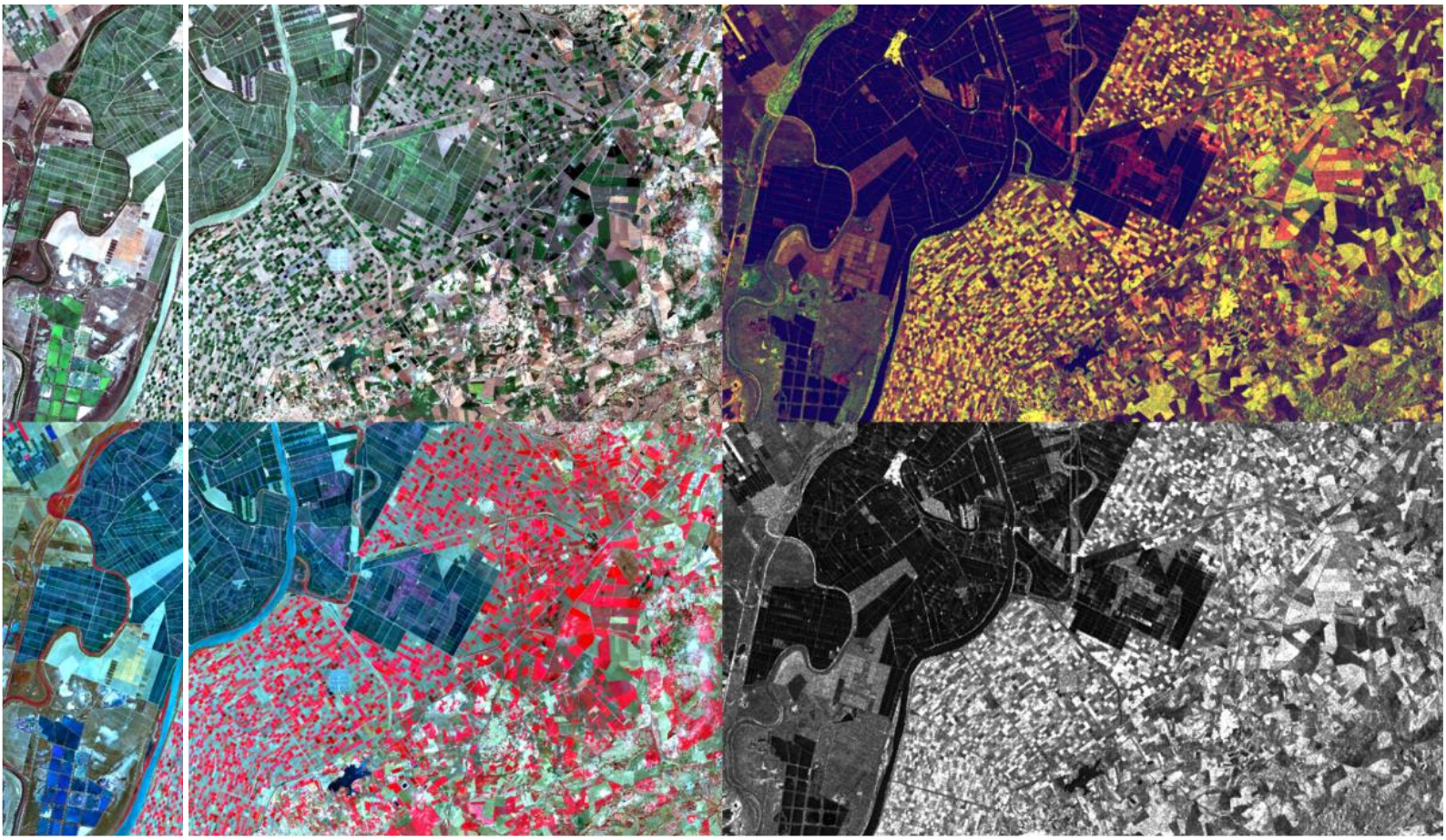
European Commission

esa

RUS

Copernicus



TRAINING KIT – LIS0218

CROP MAPPING WITH SENTINEL-1 AND SENTINEL-2

Case Study: Seville 2017, Spain

serco NOVELTIS ALONG-TRACK ROMANIA

Did you find this material useful?

Authors would be glad to receive your feedback or suggestions and to know how this material was used. Please, contact us on training@rus-coperenicus.eu

Enjoy RUS!

Table of Contents

1 Introduction to RUS

The Research and User Support for Sentinel core products (RUS) service provides a free and open scalable platform in a powerful computing environment, hosting a suite of open source toolboxes pre-installed on virtual machines, to handle and process data derived from the Copernicus Sentinel satellites constellation.

In this tutorial, we will employ RUS to run a supervised classification using the Random Forest algorithm and Sentinel-1 / Sentinel-2 as input data over an agricultural area in Seville, Spain.

2 Crop mapping – background

Agricultural area near Seville (Spain) seen by Sentinel-2. Source: RUS Copernicus

Reliable information on agriculture and crops is required to assist and help in the decision-making process of different applications. Different methods can be used to gather this information but satellite earth observation offers a suitable approach based on the coverage and type of data that are provided.

A few years ago, the European Union (EU) started an ambitious program, Copernicus, which includes the launch of a new family of earth observation satellites known as the Sentinels. Amongst other applications, this new generation of remote sensing satellites will improve the observation, identification, mapping, assessment, and monitoring of crop dynamics at a range of spatial and temporal resolutions.

3 Training

Approximate duration of this training session is two hours.

3.1 Data used

- 7 Sentinel-2A images acquired from June 1st until July 31st 2017 [downloadable at https://scihub.copernicus.eu/ using the .meta4 file provided in the AuxData folder of this exercise]
- 7 Sentinel-1A/1B images acquired from June 1st until July 31st 2017 [downloadable at https://scihub.copernicus.eu/ using the .meta4 file provided in the AuxData folder of this exercise]
- Pre-processed data stored locally *@/shared/Training/TAT0618_CropMapping_Croatia/AuxData*

3.2 Software in RUS environment

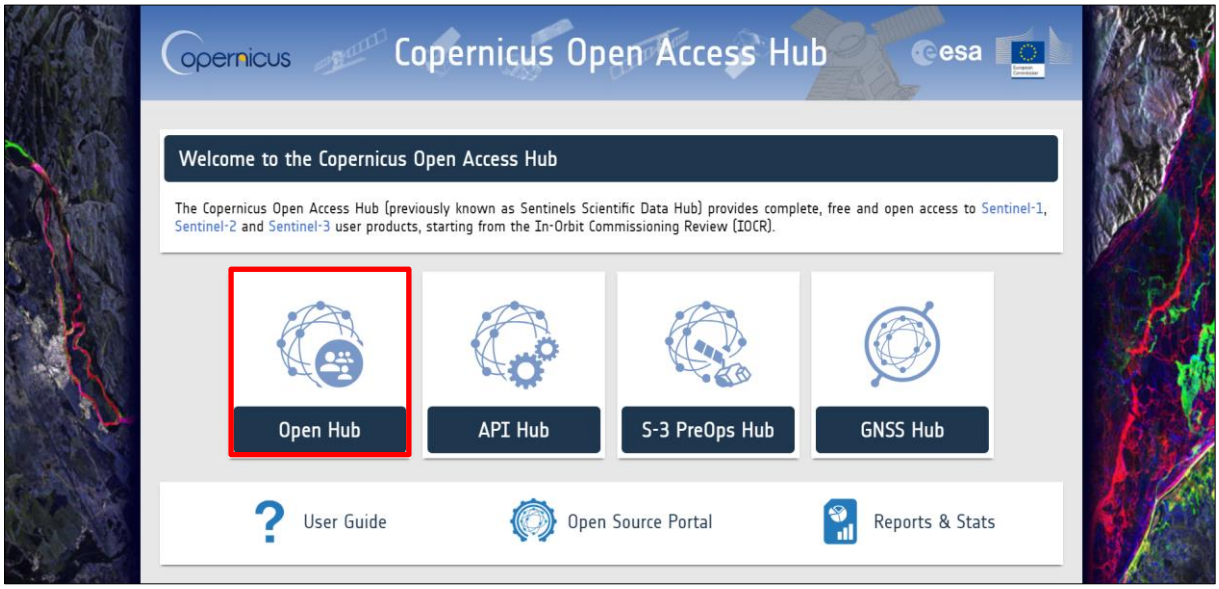
Internet browser, SNAP + S1 Toolbox, R + RStudio

4 Step by step

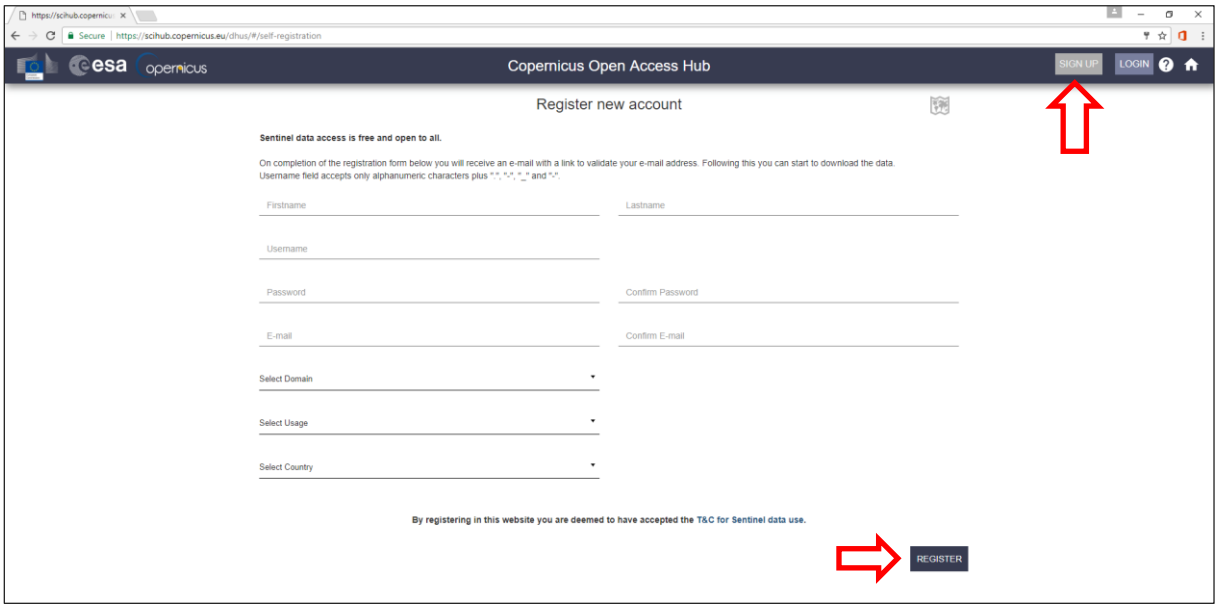
4.1 Data download – ESA SciHUB

Before starting the exercise, we need to make sure that we are registered in the Copernicus Open Access Hub so that we can access the free data provided by the Sentinel satellites.

Go to https://scihub.copernicus.eu/



Go to Open HUB. If you do not have an account please sign up in the upper right corner, fill in the details and click register.



You will receive a confirmation email on the e-mail address you have specified: open the email and click on the link to finalize the registration.

Once your account is activated – or if you already have an account – log in.

4.2 Download data

In this exercise, we will use 7 Sentinel-1 and 7 Sentinel-2 images during the year 2017. The following table shows the date and reference of the images that will be used:

SATELLITE	DATE	IMAGE ID
Sentinel-1	2017-06-01	S1A_IW_GRDH_1SDV_20170601T182618_20170601T182643_016844_01C013_F35E
	2017-06-13	S1A_IW_GRDH_1SDV_20170613T182618_20170613T182643_017019_01C583_4C25
	2017-06-19	S1B_IW_GRDH_1SDV_20170619T182531_20170619T182556_006123_00AC18_303B
	2017-07-02	S1A_IW_GRDH_1SDV_20170702T181819_20170702T181844_017296_01CDF5_4DD6
	2017-07-08	S1A_IW_GRDH_1SDV_20170708T062702_20170708T062727_017376_01D051_2318
	2017-07-20	S1A_IW_GRDH_1SDV_20170720T062702_20170720T062727_017551_01D5A7_A95F
	2017-07-31	S1A_IW_GRDH_1SDV_20170731T182621_20170731T182646_017719_01DAD0_8200
Sentinel-2	2017-06-01	S2A_MSIL2A_20170601T110651_N0205_R137_T30STG_20170601T111225
	2017-06-11	S2A_MSIL2A_20170611T110621_N0205_R137_T30STG_20170611T111012
	2017-06-21	S2A_MSIL2A_20170621T110651_N0205_R137_T30STG_20170621T111222
	2017-07-01	S2A_MSIL2A_20170701T111051_N0205_R137_T30STG_20170701T111746
	2017-07-11	S2A_MSIL2A_20170711T110651_N0205_R137_T30STG_20170711T111223
	2017-07-21	S2A_MSIL2A_20170721T110621_N0205_R137_T30STG_20170721T112025
	2017-07-31	S2A_MSIL2A_20170731T110651_N0205_R137_T30STG_20170731T111220

To improve the data acquisition process, we will use a download manager (See NOTE 1) that will take care of downloading all products that will be used in this exercise. The metadata of the Sentinel products are contained in a *products.meta4* file created using the 'Cart' option of the Copernicus Open Access Hub

> NOTE 1: A download manager is a computer program dedicated to the task of downloading possibly unrelated stand-alone files from (and sometimes to) the Internet for storage. For this exercise, we will use aria2. Aria2 is a lightweight multi-protocol & multi-source command-line download utility. More info at: https://aria2.github.io/

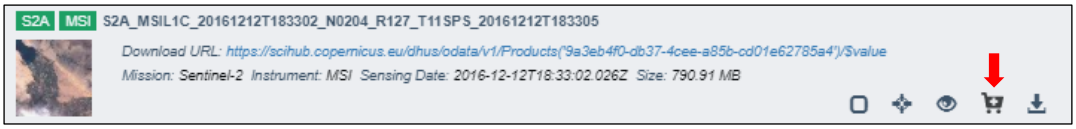
The *products.meta4* file containing the links to the Sentinel-1 and Sentinel-2 products to be downloaded have been already created following the methodology explained (See NOTE 2). You can find the *products.meta4* files saved in the following path:

Path: */shared/Training/TAT0618_CropMapping_Croatia/Original/*

Before using the downloading manager and the .meta4 file, let us test if *aria2* is properly installed in the Virtual Machine. To do this, open the Command Line (in the bottom of your desktop window) and type:

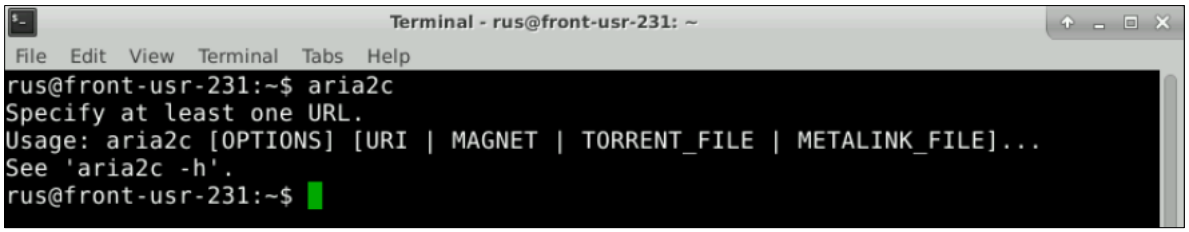
```
aria2c
```

NOTE 2: The Copernicus Open Access Hub allows you to add products to a 'Cart'. For that, perform a query; select the desired products from the result list and click on the 'Add Product to Cart' icon -



To view the products present in the cart just click anytime on the User Profile icon on top right corner of the screen and then on "Cart". To download the products contained in the cart just click on "Download

If *aria2* is properly installed, the response should be as follows. If the response is '*-bash aria2c: command not found*' it means *aria2* is not installed (See NOTE 3).



NOTE 3: If **(and only if)** the response is '*-bash aria2c: command not found',* you need to install aria2. In the command line, type: *sudo apt-get install aria2*
When requested, type: Y
Once finished, test the installation as explanied before.

Once *aria2* is ready to use, we can start the download process. For that, we need to navigate to the folder where the *products.meta4* is stored. Type the following command in the terminal and run it.

```
cd /shared/Training/TAT0618_CropMapping_Croatia/Original/
```

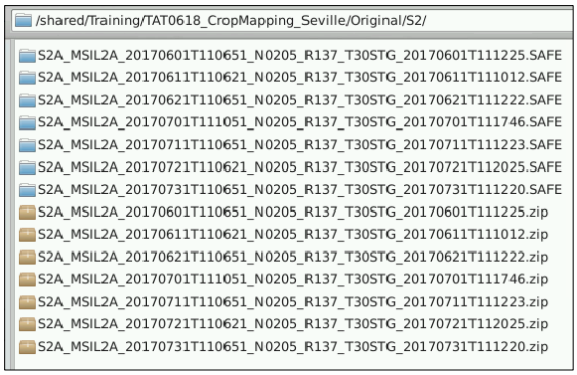
Next, type the following command (in a single line) to run the download tool. Replace *username* and *password* (leave the quotation marks) with your login credentials for Copernicus Open Access Hub (COAH). Do not clear your cart in the COAH until the download process is finished.

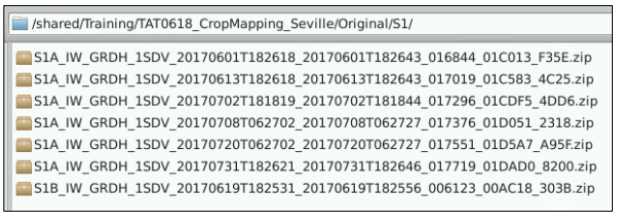
```
aria2c --http-user='username' --http-passwd='password' --check-certificate=
false --max-concurrent-downloads=2 -M products.meta4
```

The Sentinel products will be saved in the same path where the *products.meta4* is stored. Move the Sentinel-1 and Sentinel-2 images to their corresponding folder and do not forget to unzip the Sentinel-2 products after that. Your folders should have the same structure as shown below.

Sentinel-1 folder → */shared/Training/TAT0618_CropMapping_Croatia/Original/S1/*

Sentinel-2 folder → */shared/Training/TAT0618_CropMapping_Croatia/Original/S2/*



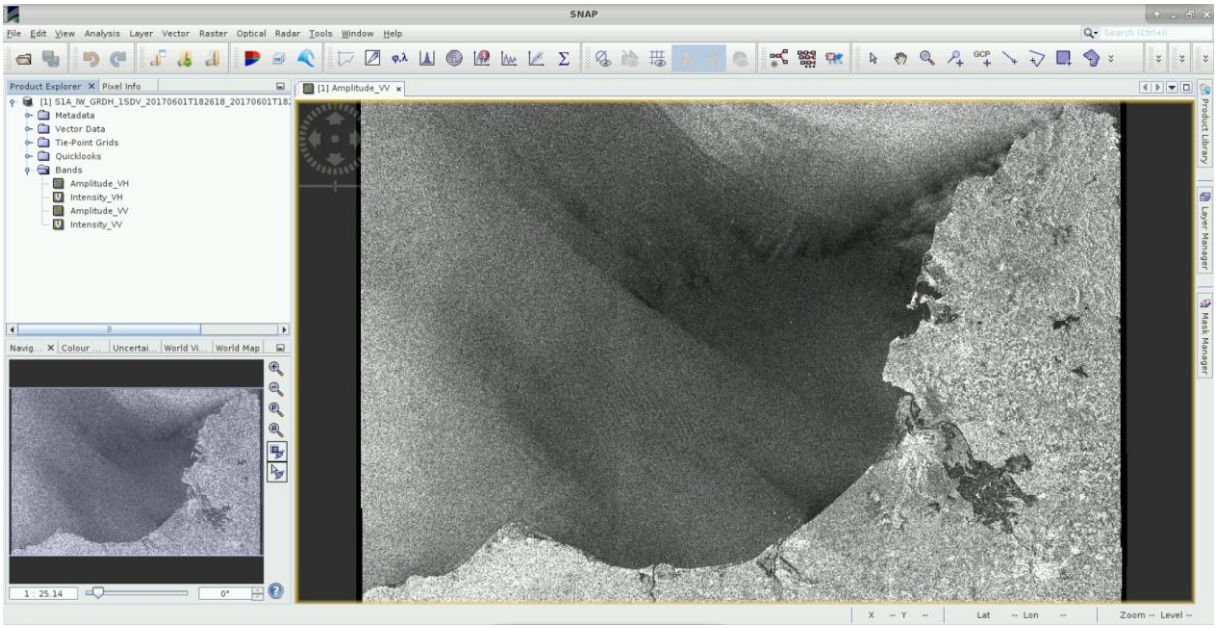


4.3 Sentinel-1 SNAP Preprocessing

Once the Sentinel-1 images are downloaded, we need to run some pre-processing steps before they can be used for the classification. For this purpose, we will use the SNAP software. In *Applications -> Other* open **SNAP Desktop**; click **Open product**  , navigate to the following path and open the first S1 image (2017-06-01)

Path: */shared/Training/TAT0618_CropMapping_Seville/Original/S1*

The opened product will appear in Product Explorer. Click + to expand the contents of the first image, then expand the *Bands* folder and double click on the *Amplitude_VV* band to visualize it.



In order to process this and the other Sentinel-1 images, we will take advantage of the batch processing option available in SNAP. In this way, we can define a specific processing chain and apply it to several images in an automatic way. This allows reducing processing time and storage requirement since no intermediate steps are created. Only the final product is physically saved.

Before running batch processing, it is necessary to create a graph containing all the processing steps. Go to *Tools -> Graph Builder.* So far, the graph only has two operators: Read (to read the input) and Write (to write the output). With right-click on the top panel, you can add an operator while a corresponding tab is created and added on the bottom panel.

4.3.1 Apply orbit file

The first step of our Sentinel-1 preprocessing chain will update the orbit metadata (See NOTE 4) of the product to provide accurate satellite position and velocity information. To add the operator to our graph, right click and navigate to *Add -> RADAR -> Apply-Orbit-File*. Connect the new *Apply-Orbit-File* operator with the *Read* operator by clicking to the right side of the *Read* operator and dragging the red arrow towards the *Apply-Orbit-File* operator. In the corresponding tab, leave all the parameters for this operator as default.

> NOTE 4: The orbit state vectors provided in the metadata of a SAR product are generally not accurate and can be refined with the precise orbit files, which are available days-to-weeks after the generation of the product. The orbit file provides accurate satellite position and velocity information. Based on this information, the orbit state vectors in the abstract metadata of the product are updated. (*SNAP Help*)

4.3.2 Thermal Noise Removal

Next, we will remove the thermal noise (See NOTE 5). To add the operator to our graph, right click and navigate to *Add -> RADAR -> Radiometric -> ThermalNoiseRemoval*. In the corresponding tab, leave all the parameters for this operator as default.

> NOTE 5: Thermal noise in SAR imagery is the background energy that is generated by the receiver itself. (*SNAP Help*) It skews the radar reflectivity to towards higher values and hampers the precision of radar reflectivity estimates. Level-1 products provide a noise LUT for each measurement dataset, provided in linear power, which can be used to remove the noise from the product.

4.3.3 Calibration

Now, we can perform the Radiometric calibration. The objective of SAR calibration is to provide imagery in which the pixel values can be directly related to the radar backscatter. Though uncalibrated SAR imagery is sufficient for qualitative use, calibrated SAR images are essential to quantitative use of SAR data (See NOTE 6). To add the operator to our graph, right click and navigate to *Add -> RADAR -> Radiometric -> Calibration*. In the corresponding tab, leave all the parameters for this operator as default.

NOTE 6: Typical SAR data processing, which produces level-1 images, does not include radiometric corrections and significant radiometric bias remains. The radiometric correction is necessary for the pixel values to truly represent the radar backscatter of the reflecting surface and therefore for comparison of SAR images acquired with different sensors, or acquired from the same sensor but at different times, in different modes, or processed by different processors. (*SNAP Help*)

4.3.4 Speckle filter

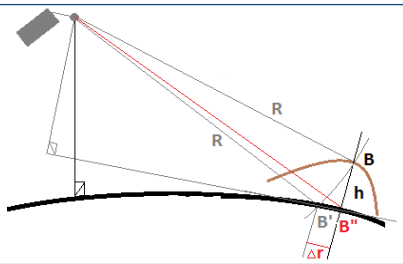
SAR images have inherent salt and pepper like texturing called speckles that degrade the quality of the image and make interpretation of features more difficult (See NOTE 7). To reduce the speckle effect and smooth the image we apply a speckle filter. For this exercise, we will use the default speckle filter used in SNAP (Lee Sigma). To add the operator to our graph, right click and navigate to *Add -> RADAR -> Speckle Filtering -> Speckle-Filter*. In the corresponding tab, leave all the parameters for this operator as default.

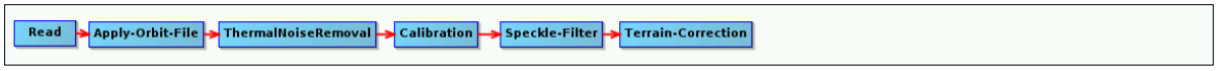
NOTE 7: Speckle is caused by random constructive and destructive interference of the de-phased but coherent return waves scattered by the elementary scatters within each resolution cell. Speckle noise reduction can be applied by either spatial filtering or multilook processing. (*SNAP Help*)

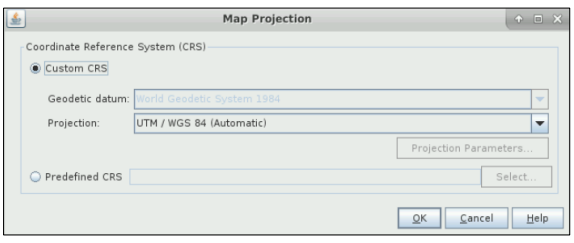
4.3.5 Terrain correction

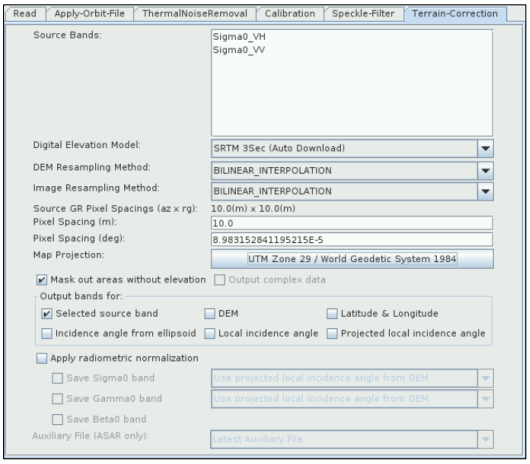
Our data are still in radar geometry, moreover due to topographical variations of a scene and the tilt of the satellite sensor, the distances can be distorted in the SAR images. Therefore, we will apply terrain correction to compensate for the distortions and reproject the scene to geographic projection (See NOTE 5). To add the operator to our graph, right click and navigate to *Add -> RADAR -> Speckle Filtering -> Speckle-Filter*. In the corresponding tab, make sure you select *UTM / WGS 84 (Automatic)* as Map Projection.

NOTE 5: The geometry of topographical distortions in SAR imagery is shown on the right. Here we can see that point **B** with elevation **h** above the ellipsoid is imaged at position **B'** in SAR image, though its real position is **B"**. The offset **Δr** between **B'** and **B"** exhibits the effect of topographic distortions. *(SNAP Help)*





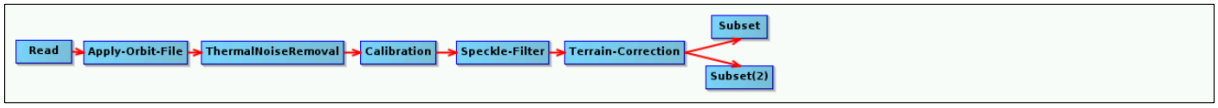


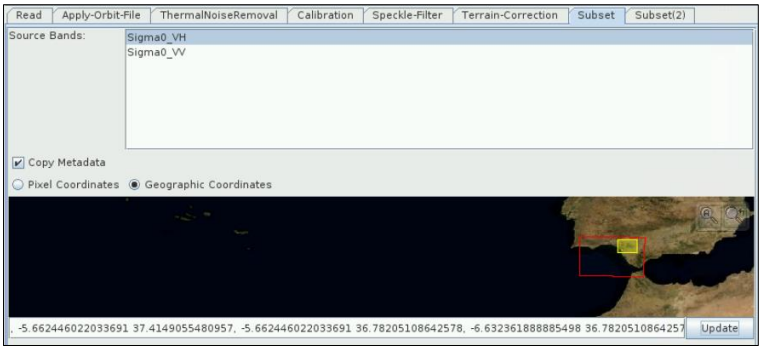


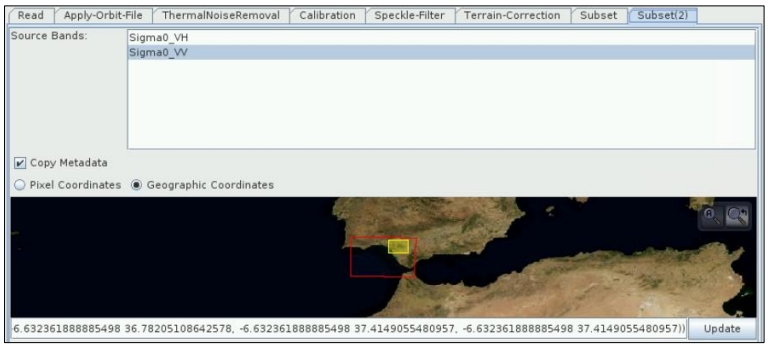
4.3.6 Subset

In the last step of the Sentinel-1 pre-processing chain, we will subset the original extent of the image. This will reduce the size of the product and processing time. In addition, we will use the subset operator to save the two output bands (Sigma0_VV and Sigma0_VH) as separate products. This time, we will add two subset operators. For that, right click and navigate to *Add -> Raster -> Geometric -> Subset.* Repeat the process one more time to add the second operator. In the first subset tab, make sure **you select only the Sigma0_VH tab.** Then, select the option *Geographic Coordinates* and paste the following Well-Known Text to define to subset area. Then, click *Update* and visualize the area by clicking on the zoom icon.  Repeat the same process for the second subset tab - *Subset(2)* – **but this time select only Sigma0_VV** as source band.

```
POLYGON ((-6.632361888885498 37.4149055480957, -5.662446022033691 37.4149055480957,
-5.662446022033691 36.78205108642578, -6.632361888885498 36.78205108642578, -6.6323
61888885498 37.4149055480957, -6.632361888885498 37.4149055480957))
```

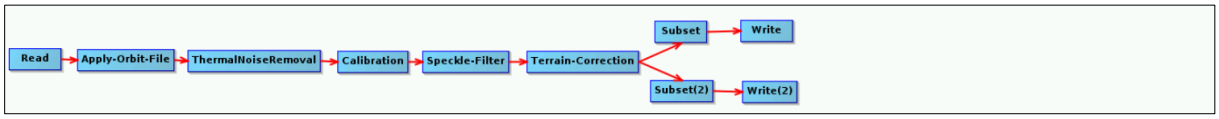


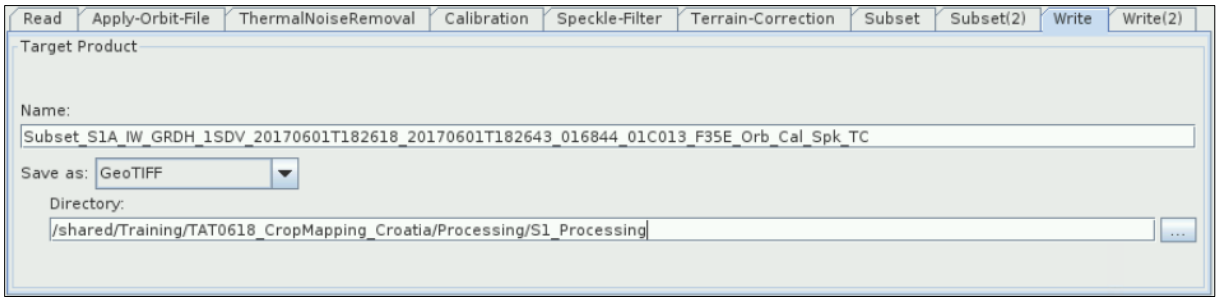




4.3.7 Write

Finally, we just need to properly save the output. For that, we first need to add two W*rite* operators to our graph. Right click and navigate to *Add -> Input-Output -> Write*. Repeat the process one more time to add the second *Write* tool. **In the two *Write* tabs**, make sure you set the **output format as GeoTIFF.**





Once finished, click on the *Save* icon located on the lower part of the graph builder. Navigate to the following path and save the graph as *S1_Processing_Original.xml*.

Path: */shared/Training/TAT0618_CropMapping_Croatia/AuxData*

4.3.8 Bulk processing preparation

To use the batch processing option for all the Sentinel-1 images using GPT, we first need to change the input and output variables defined on the graph we have just created. Navigate to the following path, right click on the graph file (called *S1_Processing_Original.xml*) and select *Open With -> Open with Mousepad*

Path: */shared/Training/TAT0618_CropMapping_Croatia/AuxData*

Once the .xml file is open, click on *View -> Line Numbers*. In **line 7**, delete the path (do not remove *<file>* and *</file>)* to the input image and write *$input1*. Line 7 should look like this:

```
<file>$input1</file>
```

```
1 <graph id="Graph">
2   <version>1.0</version>
3   <node id="Read">
4     <operator>Read</operator>
5     <sources/>
6     <parameters class="com.bc.ceres.binding.dom.XppDomElement">
7       <file>$input1</file>
8     </parameters>
9   </node>
```

In **line 163**, delete the path (do not remove <file> and </file> to the input image and write *$target1*. Line 163 should look like this:

```
<file>$target1</file>
```

```
157   <node id="Write(2)">
158     <operator>Write</operator>
159     <sources>
160       <sourceProduct refid="Subset(2)"/>
161     </sources>
162     <parameters class="com.bc.ceres.binding.dom.XppDomElement">
163       <file>$target1</file>
164       <formatName>GeoTIFF</formatName>
165     </parameters>
166   </node>
```

In **line 173**, delete the path (do not remove <file> and </file> to the input image and write *$target2*. Line 173 should look like this:

```
<file>$target2</file>
```

```
167   <node id="Write">
168     <operator>Write</operator>
169     <sources>
170       <sourceProduct refid="Subset"/>
171     </sources>
172     <parameters class="com.bc.ceres.binding.dom.XppDomElement">
173       <file>$target2</file>
174       <formatName>GeoTIFF</formatName>
175     </parameters>
176   </node>
```

Once the input and output variables are changed, save the graph as a new xml file. Go to *File->Save As.* Navigate to the following path and save it as *S1_Processing_GPT.xml.* We will call this graph using R later on this exercise to batch process all the Sentinel-1 images.

Path: */shared/Training/TAT0618_CropMapping_Croatia/AuxData*

4.4 R – Processing

At this point, we are ready to start the next part of our analysis. This exercise will be done using RStudio, an integrated development environment (IDE) for R (See NOTE 6). It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management. The choice of this software (e.g. instead of SNAP) is based on the necessity of applying a replicable processing chain to a large dataset in a specific period.

> NOTE 6: R is a language and environment for statistical computing and graphics. It provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering…) and graphical techniques, and is highly extensible. R can be extended (easily) via *packages*. There are about eight packages supplied with the R distribution and many more are available through the CRAN family of Internet sites covering a very wide range of modern statistics. It is available as Free Software under the terms of the Free Software Foundation's GNU General Public License in source code form.

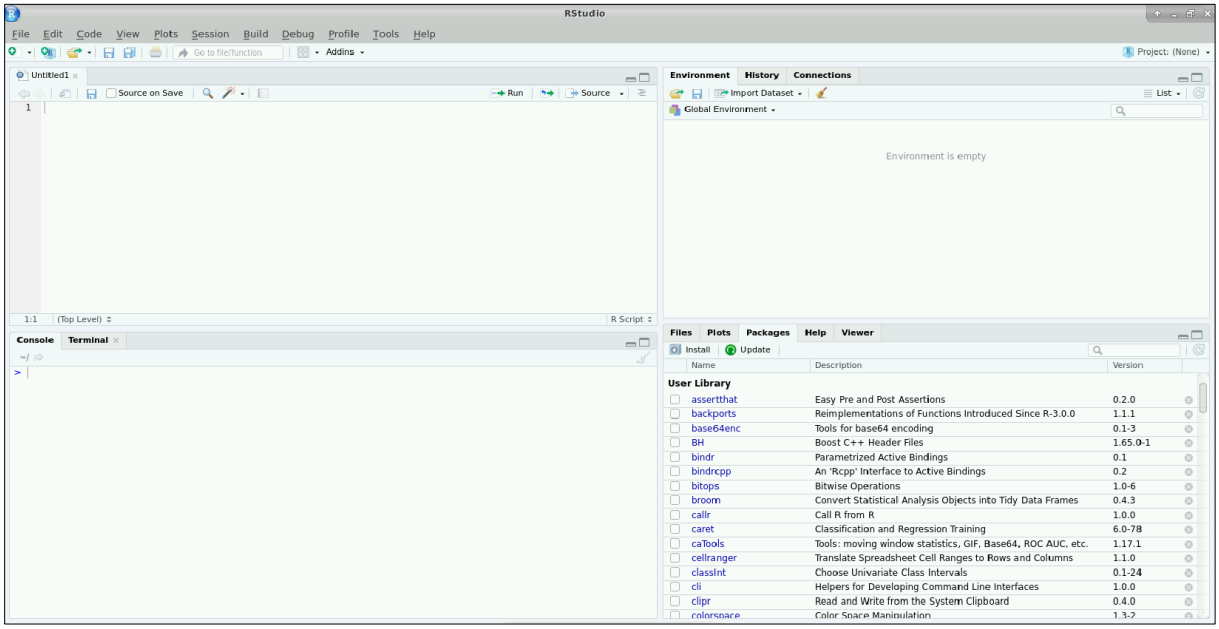
Go to *Applications/Development/R/RStudio* and open RStudio. Now, click on *File -> New File -> R script*. Before starting, save the file in the following path and name it '*R_Code_TAT0618'.* From time to time, remember to save the file by clicking on *File -> Save.*

Path: */shared/Training/TAT0618_CropMapping_Croatia/AuxData/*

The R code that will be used to perform the analysis is provided to you in this step-by-step guide. Copy-Paste each section in your R Script and run it. The code is divided in different blocks, each one containing detailed explanations.

4.4.1 Setting R

Before starting to create our code, it is important to introduce the default layout of RStudio.



The Graphical User Interface (GUI) has four main parts. The top left quadrant is the editor, where you write R code you want to keep for later – functions, classes, packages, etc. The lower left quadrant is the console. It is a REPL (Read-Eval-Print-Loop) for R in which the code written in the editor can be tested. The top right quadrant has two tabs: environment and history. The Environment will list all the variables, data, functions, etc. used in your code. The bottom right panel is the misc panel, and contains five separate tabs, where the most important are plots (will contain the graphs you generated with R), packages (tells which packages are available to use and let you install additional ones) and help (allows you to search for extra information about functions and packages).

Before running the code, it is important to highlight that R works by calling packages, which contain the functions that are used in the analysis. They can be installed directly from the console and later loaded to the current R session.

For this exercise, we will need the following packages.

```
# Install and load packages

pck<-c("tidyr","rgdal","ggplot2","randomForest","RColorBrewer", "caret",
"reshape2", "raster", "e1071", "rasterVis")

new_pck<-pck[!pck %in% installed.packages()[,"Package"]]

if(length(new_pck)){install.packages(new_pck)}


lapply(pck, require, character.only=TRUE)


# Specify working directory

setwd("/shared/Training/TAT0618_CropMapping_Seville/AuxData/")
```

4.4.2 Load AuxData

Load the auxiliary data that will be used in our analysis. Make sure they are located in the AuxData folder of this training kit. (*@/shared/Training/TAT0618_CropMapping_Seville/AuxData/)*

```
Study_area<-readOGR("Study_Area.shp")
training<-readOGR("Training.shp")
validation<-readOGR("Validation.shp")
```

4.4.3 User input

Select which Sentinel mission you want to use for the classification. Depending on your choice, different parts of the code will be run or not (look for the *if* statements).

```
# Do you want to use Sentinel-1, Sentinel-2 or both for the classification?
# TRUE/FALSE
S1_input<-FALSE
S2_input<-FALSE
S1_S2_input<-TRUE
```

Set the path to the directory where you Sentinel-1 and Sentinel-2 images are stored.

```
# Where are your images stored? Set the path.
S1_path<-"/shared/Training/TAT0618_CropMapping_Seville/Original/S1"
S2_path<-"/shared/Training/TAT0618_CropMapping_Seville/Original/S2/"
```

Specify the number of images that have been downloaded.

```
# How many images do you have?
t_images<-7
```

Set the number of images you want to use for the classification.

```
# How many images do you want to use for the classification?
n_images<-5
```

In case you are using Sentinel-2, choose which bands you want to use as input for the classification. Copy this code if you want to use all the Sentinel-2 bands except band 1 and 10.

```
# For S2, bands to be used for classification
bands<-c("B((0[2348]_10m)|(((0[567])|(1[12])|(8A))_20m)).jp2$")
```

If you prefer to try with bands 2,3,4,8,11,12 use the following code instead

```
# Bands 2,3,4,8,12
bands<-c("B((0[2348]_10m)|((1[12])_20m)).jp2$")
```

In case you have already pre-processed the Sentinel-1 images in SNAP, set the variable as TRUE. If not, write FALSE.

```
# Are your Sentinel-1 images pre-processed already in SNAP?
S1_pp<-TRUE
```

Define the path where the GPT version of SNAP is stored in your Virtual Machine.

```
# Path to GPT?
GPT_path<-"/usr/local/snap6/bin/gpt"
```

Set the path where the graph that will be used to pre-processed the Sentinel-1 images is stored.

```
# Path to SNAP graph?
GPT_graph<-
"/shared/Training/TAT0618_CropMapping_Seville/AuxData/S1_Processing_GPT.xml"
```

Select the folder where you want to store the Sentinel-1 pre-processed data.

```
# Output directory for S1 preprocessed data?
S1_Out_Path<-
"/shared/Training/TAT0618_CropMapping_Seville/Processing/S1_Processing/"
```

Define the number of trees you want to use in the Random Forest Classification.

```
# How many trees do you want to use in the RF classification?
n_trees<-500
```

Last, we will create several internal cross-references that will be used later on the code.

```
# Internal cross-references
nrow<-ifelse(S1_input, 1, ifelse(S2_input & nchar(bands)>40, 2, ifelse
(S1_S2_input & nchar(bands)>40, 3, ifelse (S2_input & nchar(bands)<40, 4, 5
))))
```

```
condition_1<-parse(text=ifelse(S1_S2_input==TRUE, print("S1_S2_images"),
ifelse(S2_input==TRUE, print("S2_images"), print("S1_images"))))
```

```
condition_2<-parse(text=paste0(condition_1, "[[1]]"))
```

```
extent_tmpl<-extent(219800,230540,4097480,4104080)
```

4.4.4 S1 | SNAP Pre-processing

This section will only be run if you have chosen S1 as input for the classification (S1 or S1 + S2) and the images are not pre-processed already. The code uses the xml graph *(S1_Processing_GPT.xml)* created before (*See 4.3.8 Bulk processing preparation*). It will modify the variables *$input1, $target1* and *$target2* with the appropriate reference for each S1 image and save each output *(Sigma0_VV and Sigma0_VH)* with a unique name (e.g. 20170601_VV) in a specific folder.

First, we create a list with the names of the Sentinel-1 original products. The sensing date contained in the name is extracted.

```
# Create list of S1 and extract date
if ((S1_input | S1_S2_input) & !S1_pp){
  S<-list.files(S1_path, recursive = TRUE, full.names = TRUE, pattern="S1")
  dates<-substr(S, 75,82)
```

```
> S
[1] "/shared/Training/TAT0618_CropMapping_Seville/Original/S1/S1A_IW_GRDH_1SDV_20170601T182618_20170601T182643_016844_01C013_F35E.zip"
[2] "/shared/Training/TAT0618_CropMapping_Seville/Original/S1/S1A_IW_GRDH_1SDV_20170613T182618_20170613T182643_017019_01C583_4C25.zip"
[3] "/shared/Training/TAT0618_CropMapping_Seville/Original/S1/S1A_IW_GRDH_1SDV_20170702T181819_20170702T181844_017296_01CDF5_4DD6.zip"
[4] "/shared/Training/TAT0618_CropMapping_Seville/Original/S1/S1A_IW_GRDH_1SDV_20170708T062702_20170708T062727_017376_01D051_2318.zip"
[5] "/shared/Training/TAT0618_CropMapping_Seville/Original/S1/S1A_IW_GRDH_1SDV_20170720T062702_20170720T062727_017551_01D5A7_A95F.zip"
[6] "/shared/Training/TAT0618_CropMapping_Seville/Original/S1/S1A_IW_GRDH_1SDV_20170731T182621_20170731T182646_017719_01DAD0_8200.zip"
[7] "/shared/Training/TAT0618_CropMapping_Seville/Original/S1/S1B_IW_GRDH_1SDV_20170619T182531_20170619T182556_006123_00AC18_303B.zip"
```

```
> dates
[1] "20170601" "20170613" "20170702" "20170708" "20170720" "20170731" "20170619"
```

Next, we create a list for the *$input1* variable by pasting together '*–Pinput1=*' and the name of each Sentinel-1 product.

```
# Prepare script for GPT
input<-list()
  for (i in S){
    input[[i]]<-paste("-Pinput1=", i, sep="")}
```

The same procedure is used to define the *$target1* variable. In this case, the list contains the output path, date and VV polarization for each output.

```
output1<-list()
for (i in 1:length(S)){
    output1[[i]]<-paste("-Ptarget1=", S1_Out_Path, dates[[i]], "_VV",
".tif", sep="")}
```

The same procedure is used again to define the *$target2* variable. In this case, the list contains the output path, date and VH polarization for each output.

```
output2<-list()
for (i in 1:length(S)){
    output2[[i]]<-paste("-Ptarget2=", S1_Out_Path, dates[[i]], "_VH",
".tif", sep="")}
```

Next, we combine all the parts we have created.

```
# Create final script for GPT

script<-paste(GPT_path,GPT_graph, input, output1, output2)
```

```
> script[[1]]
[1] "/usr/local/snap6/bin/gpt /shared/Training/TAT0618_CropMapping_Seville/AuxData/S1_Processing_GPT.
xml -Pinput1=/shared/Training/TAT0618_CropMapping_Seville/Original/S1/S1A_IW_GRDH_1SDV_20170601T18261
8_20170601T182643_016844_01C013_F35E.zip -Ptarget1=/shared/Training/TAT0618_CropMapping_Seville/Proce
ssing/S1_Processing/20170601_VV.tif -Ptarget2=/shared/Training/TAT0618_CropMapping_Seville/Processing
/S1_Processing/20170601_VH.tif"
```

Finally, GPT is called for each element of the list.

```
# Bulk processing

for (i in 1:length(script)){

  system(script[[i]])}}
```

4.4.5 S1 | R Processing

This section will only be run if you have chosen Sentinel-1 as input for the classification (S1 or S1 + S2). First, we load as raster files the Sentinel-1 images we have selected for the classification.

```
# Load S1 images

if (S1_input | S1_S2_input){

  S1<-list.files(S1_Out_Path, recursive = TRUE, full.names = TRUE, pattern=
"VV|VH")

  S1<-S1[1:(n_images*2)]

  S1<-lapply(1:length(S1), function (x) {raster(S1[x])})
```

```
> S1
[[1]]
class       : RasterLayer
dimensions  : 7311, 8856, 64746216  (nrow, ncol, ncell)
resolution  : 10, 10  (x, y)
extent      : 175824.6, 264384.6, 4073992, 4147102  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=utm +zone=30 +datum=WGS84 +units=m +no_defs +ellps=WGS84 +towgs84=0,0,0
data source : /shared/Training/TAT0618_CropMapping_Seville/Processing/S1_Processing/20170601_VH.tif
names       : X20170601_VH
```

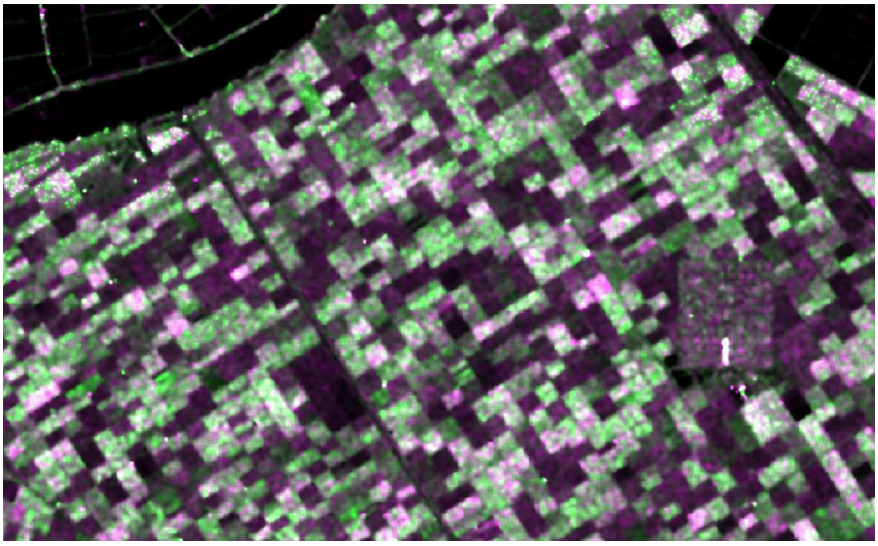
Next, the products are crop using the study area shapefile as reference and stacked together.

```
  # Crop and Stack

  S1<-lapply(S1, FUN=function (S1) {crop(S1, Study_area, snap="out")})

  for (i in 1:(n_images*2)) {S1[[i]]<-setExtent(S1[[i]], extent_tmpl)}

  S1_images<-stack(S1)

  S1_images
```

```
> S1_images
class       : RasterStack
dimensions  : 660, 1074, 708840, 10  (nrow, ncol, ncell, nlayers)
resolution  : 10, 10  (x, y)
extent      : 219800, 230540, 4097480, 4104080  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=utm +zone=30 +datum=WGS84 +units=m +no_defs +ellps=WGS84 +towgs84=0,0,0
names       : X20170601_VH, X20170601_VV, X20170613_VH, X20170613_VV, X20170619_VH, X20170619_VV, X20170702_VH, X20170702_VV, X20170708_VH, X20170708_VV
min values  : 0.000000e+00, 2.667823e-03, 1.437411e-05, 3.335070e-03, 2.234252e-04, 4.926375e-03, 0.000000e+00, 4.900703e-04, 3.651371e-05, 5.078124e-03
max values  :    0.9121780,   22.3900623,    0.8948286,   23.6558838,    0.8642644,   25.8782654,    1.0299687,   20.5808182,    0.9786080,   34.9954948
```

The result is displayed as an RGB false colour composition. The following band combination is used: R=VV G=VH B=VV/VH

```
  # Plot RGB (VV-VH-VV/VH)

  plotRGB(S1_images, r=2, g=1, b=2/1, scale=maxValue(S1_images[[1]]),
stretch="lin")}
```



4.4.6 S2 | R Processing

This section will only be run if you have chosen Sentinel-2 as input for the classification (S2 or S1 + S2). First, we load as raster files the Sentinel-2 images we have selected for the classification.

```
  # Load S2 images
if (S2_input | S1_S2_input){
  S2<-list.files(S2_path, full.names = TRUE, pattern = ".SAFE")
  S2<-S2[1:n_images]
  S2<-list.files(S2, recursive = TRUE, full.names = TRUE, pattern=bands)
  S2<-lapply(1:length(S2), function (x) {raster(S2[x])})
  head(S2)
```

```
> head(S2)
[[1]]
class       : RasterLayer
dimensions  : 10980, 10980, 120560400  (nrow, ncol, ncell)
resolution  : 10, 10  (x, y)
extent      : 199980, 309780, 4090200, 4200000  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=utm +zone=30 +datum=WGS84 +units=m +no_defs +ellps=WGS84 +towgs84=0,0,0
data source : /shared/Training/TAT0618_CropMapping_Seville/Original/S2/S2A_MSIL2A_20170601T110651_N0205_R137_T30STG_20170601
T111225.SAFE/GRANULE/L2A_T30STG_A010144_20170601T111225/IMG_DATA/R10m/L2A_T30STG_20170601T110651_B02_10m.jp2
names       : L2A_T30STG_20170601T110651_B02_10m
values      : 0, 4273  (min, max)
```

Next, the products are crop using the study area shapefile as reference, resample to a common spatial resolution (10 meters pixel size) and stacked together.

```
 # Crop - Resample - Stack
```

```
  S2_crop<-lapply(S2, function(S2) if(xres(S2)==10) {crop(S2, Study_area,
snap="out")} else {crop(S2, Study_area, snap="near")})

  S2_rsp<-lapply(S2_crop, FUN = function(S2_crop) {if (xres(S2_crop)==20)
{disaggregate(S2_crop, fact=2, method="")} else {S2_crop}})

  S2_images<-stack(S2_rsp)

  S2_images
```

```
> S2_images
class       : RasterStack
dimensions  : 660, 1074, 708840, 30  (nrow, ncol, ncell, nlayers)
resolution  : 10, 10  (x, y)
extent      : 219800, 230540, 4097480, 4104080  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=utm +zone=30 +datum=WGS84 +units=m +no_defs +ellps=WGS84 +towgs84=0,0,0
names       : L2A_T30ST//51_B02_10m, L2A_T30ST//51_B03_10m, L2A_T30ST//51_B04_10m, L2A_T30ST//51_B08_10m, L2A_T30ST//51_B11_20m, L2A_T30ST//51_B12_20m, L2A_T30ST//21_B02_10m, L2A_T30
ST//21_B03_10m, L2A_T30ST//21_B04_10m, L2A_T30ST//21_B08_10m, L2A_T30ST//21_B11_20m, L2A_T30ST//21_B12_20m, L2A_T30ST//51_B02_10m, L2A_T30ST//51_B03_10m, L2A_T30ST//51_B04_10m, ...
min values  :                    1,                    1,                  127,                    6,                    1,                   88,                   86,                    1,
                 1,                    1,                    1,                    1,                    1,                   50,                  198,                    7, ...
max values  :                 8702,                 9090,                 7589,                 7322,                 5379,                 5474,                 8743,
              9298,                 7585,                 8530,                 5712,                 5905,                 8048,                 7692,                 7187, ...
```

The result is displayed as an RGB true/false colour composition.

```
  # Plot True/False color

  plotRGB(S2_images, r=3, g=2, b=1, scale=maxValue(S2_images[[1]]),
stretch="lin")

  plotRGB(S2_images, r=4, g=3, b=2, scale=maxValue(S2_images[[1]]),
stretch="lin")}
```

4.4.7 Stack S1 and S2

This section will only be run if you have chosen Sentinel-1 and Sentinel-2 as input for the classification. The extent of S1 and S2 images is matched to avoid inconsistencies in the datasets. If the Sentinel-1 images are terrain-corrected into the same Coordinate Reference System (CSR) and pixel size as Sentinel-2, the accuracy in pixel alignment is at the sub-pixel level.

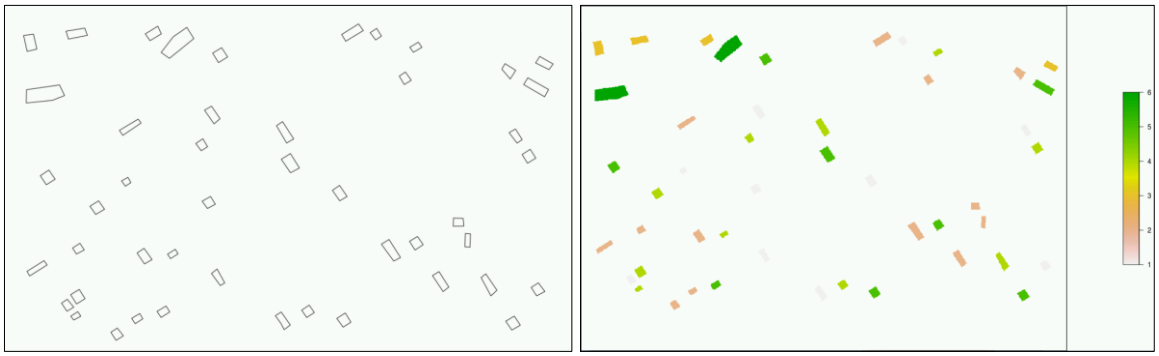
```
# Stack Sentinel-1 and Sentinel-2 processed products

if (S1_S2_input){

  S1_images<-setExtent(S1_images, S2_images)

  S1_S2_images<-stack(S1_images,S2_images)}

  S1_S2_images
```

```
> S1_S2_images
class       : RasterStack
dimensions  : 660, 1074, 708840, 40  (nrow, ncol, ncell, nlayers)
resolution  : 10, 10  (x, y)
extent      : 219800, 230540, 4097480, 4104080  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=utm +zone=30 +datum=WGS84 +units=m +no_defs +ellps=WGS84 +towgs84=0,0,0
names       : X20170601_VH, X20170601_VV, X20170613_VH, X20170613_VV, X20170619_VH, X20170619_VV, X20170702_VH, X20170702_VV, X20170708_VH, X20170708_
VV, L2A_T30ST//51_B02_10m, L2A_T30ST//51_B03_10m, L2A_T30ST//51_B04_10m, L2A_T30ST//51_B08_10m, L2A_T30ST//51_B11_20m, ...
min values  : 0.000000e+00, 2.667823e-03, 1.437411e-05, 3.335070e-03, 2.234252e-04, 4.926375e-03, 0.000000e+00, 4.900703e-04, 3.651371e-05, 5.078124e-
03,          1.000000e+00,          1.270000e+02,          6.000000e+00,          1.000000e+00,          8.800000e+01, ...
max values  :    0.9121780,    22.3900623,     0.8948286,    23.6558838,     0.8642644,    25.8782654,     1.0299687,    20.5808182,     0.9786080,    34.99549
48,          8702.0000000,          9090.0000000,          7589.0000000,          7322.0000000,          5379.0000000, ...
```

4.4.8 Training data

Once the Sentinel-1 products are preprocessed, we need to add the training data to our dataset. For that, we convert to raster the training polygons.

```
# Rasterize training data

plot(training)

training_r<-rasterize(training,eval(condition_2), field=training$Crop_ID)

names(training_r)<-"Class"

plot(training_r)
```



Next, we add the rasterized training data to the stack of Sentinel products (depending on the input sensor you have selected – S1, S2, S1+S2 – you will have a different raster stack).

```
# Add training raster to stack

S_images_t<-addLayer(eval(condition_1), training_r)

S_images_t
```

```
> S_images_t
class       : RasterStack
dimensions  : 660, 1074, 708840, 41  (nrow, ncol, ncell, nlayers)
resolution  : 10, 10  (x, y)
extent      : 219800, 230540, 4097480, 4104080  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=utm +zone=30 +datum=WGS84 +units=m +no_defs +ellps=WGS84 +towgs84=0,0,0
names       : X20170601_VH, X20170601_VV, X20170613_VH, X20170613_VV, X20170619_VH, X20170619_VV, X20170702_VH, X20170702_VV, X20170708_VH, X20170708_
VV, L2A_T30ST//51_B02_10m, L2A_T30ST//51_B03_10m, L2A_T30ST//51_B04_10m, L2A_T30ST//51_B08_10m, L2A_T30ST//51_B11_20m, ...
min values  : 0.000000e+00, 2.667823e-03, 1.437411e-05, 3.335070e-03, 2.234252e-04, 4.926375e-03, 0.000000e+00, 4.900703e-04, 3.651371e-05, 5.078124e-
03,          1.000000e+00,          1.270000e+02,          6.000000e+00,          1.000000e+00,          8.800000e+01, ...
max values  :    0.9121780,    22.3900623,     0.8948286,    23.6558838,     0.8642644,    25.8782654,     1.0299687,    20.5808182,     0.9786080,    34.99549
48,          8702.0000000,          9090.0000000,          7589.0000000,          7322.0000000,          5379.0000000, ...
```

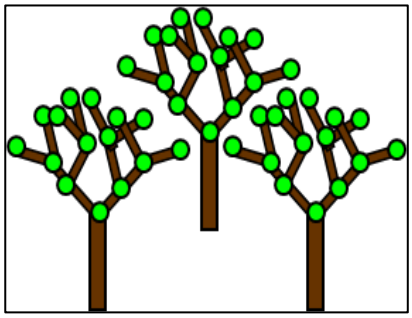
For visualization, the following code displays an RGB composition together with the training data.

```
# Plot training data

if (S2_input|S1_S2_input){

  plotRGB(S2_images, r=3, g=2, b=1, scale=maxValue(S2_images[[1]]),
stretch="lin")
```

```
  plot(training, add=TRUE, col="orange")
} else {
  plotRGB(S1_images, r=2, g=1, b=2/1, scale=maxValue(S1_images[[1]]),
stretch="lin")
  plot(training, add=TRUE, col="orange")}
```

4.4.9 Random Forest preparation

Before running the Random Forest classification (See NOTE 5), we need to train the model. The first step will be to extract the pixels identified as training pixels and save them in a dataframe.



NOTE 5: The Random Forest (Breiman, 2001) algorithm is a machine learning technique that can be used for classification or regression. In opposition to parametric classifiers (e.g. Maximum Likelihood), a machine learning approach does not start with a data model but instead learns the relationship between the training and the response dataset. The Random Forest classifier is an aggregated model, which means it uses the output from different models (trees) to calculate the response variable.

Decision trees are predictive models that recursively split a dataset into regions by using a set of binary rules to calculate a target value for classification or regression purposes. Given a training set with *n* number of samples and *m* number of variables, a random subset of samples *n* is selected with replacement and used to construct a tree. At each node of the tree, a random selection of variables *m* is used and, out of these variables, only the one providing the best split is used to create two sub-nodes.

By combining trees, the forest is created. Each pixel of a satellite image is classified by all the trees of the forest, producing as many classifications as number of trees. Each tree votes for a class membership and then, the class with the maximum number of votes is selected as the final class.

```
# Extract values for training pixels
training_S<-raster::extract(S_images_t, training, df=TRUE)
```

Next, we convert the class column to a factor (this steps allows R to read the data properly).

```
training_S$Class<-factor(training_S$Class)
```

Now, we are ready to train our Random Forest.

```
# Create and train the forest

RF<-randomForest(x=training_S[ ,c(2:(length(training_S)-1))], y=training_S$
Class, importance = TRUE, ntree=n_trees)
```

You can access the details of the Ranfom Forest model with the following code

```
# Check the Random Forest model

RF
```

```
> RF

Call:
 randomForest(x = training_S[, c(2:(length(training_S) - 1))],
   y = training_S$Class, ntree = n_trees, importance = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 6

        OOB estimate of  error rate: 0.02%
Confusion matrix:
     1    2    3    4    5    6  class.error
1 3377    0    0    0    0    0 0.0000000000
2    0 5117    0    0    0    0 0.0000000000
3    0    0 2018    0    0    0 0.0000000000
4    0    0    0 3472    2    0 0.0005757052
5    1    0    0    1 3706    0 0.0005393743
6    0    0    0    0    0 3479 0.0000000000
```

4.4.10 Random Forest Classification

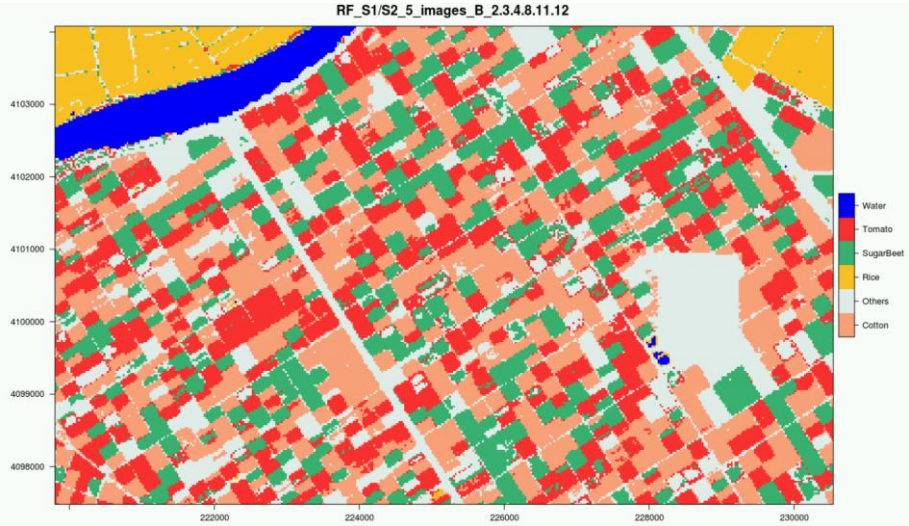
Once our model is trained, we can run the Random Forest Classification on the remaining pixels.

```
# Classification

LC<-predict(eval(condition_1), model=RF, na.rm=TRUE)
```

Use the following code to plot the output. First, we arrange the classified raster

```
# Prepare output for plot

LC<-as.factor(LC)

LC_l<-levels(LC)[[1]]

LC_l[["Crop"]]<-c("Cotton","Others","Rice","SugarBeet","Tomato", "Water")

levels(LC)<-LC_l
```

```
# Plot classification

rasterVis::levelplot(LC, col.regions=c("lightsalmon1","azure2","goldenrod1"
,"mediumseagreen","firebrick1","blue"),main=paste("RF_", if (S1_input) {"S1
"}, if (S2_input) {"S2"}, if (S1_S2_input) {"S1/S2"}, "_", n_images, "_imag
es", if (!S1_input && nchar(bands)>40) {"_B_2:9.11.12"}, if (!S1_input &&
nchar(bands)<40) {"_B_2.3.4.8.11.12"}, sep=""))
```



Now, we will save the result as GeoTIFF using an appropriate name based on the input data, number of images and bands used. The product will be saved in the following path:

Path: */shared/Training/TAT0618_CropMapping_Seville/Processing/*

```
# Save classification as GeoTIFF

writeRaster(LC, filename = paste(
"/shared/Training/TAT0618_CropMapping_Seville/Processing/", paste("RF_", if
(S1_input) {"S1"}, if (S2_input) {"S2"}, if (S1_S2_input) {"S1_S2"},"_",
n_images, "_images", if (!S1_input && nchar(bands)>40) {"_B_291112"}, if
(!S1_input && nchar(bands)<40) {"_B_23481112"}, sep = "")),format="GTiff",
overwrite=TRUE)
```

4.4.11 Accuracy assessment

The last step of this exercise will be to assess the accuracy of the classification. For that, we will use an independent validation dataset that was imported in the *4.4.2 Load AuxData* section. First, we need to rasterize the validation data.

```
# Rasterize validation data

validation_r<-rasterize(validation,eval(condition_2), field=
validation$Crop_ID)
```

Next, we extract the pixel values in the rasterized validation dataset and in the classification output.

```
# Extract values at validation pixels

test<-raster::extract(validation_r, validation, df=TRUE)

prediction<-raster::extract(LC, validation, df=TRUE)
```

Create a confusion matrix to compare reference and classification values and assess the accuracy.

```
# Create confusion matrix

CM<-caret::confusionMatrix(data=as.factor(prediction$layer), reference=
as.factor(test$layer))

CM
```

```
> CM
Confusion Matrix and Statistics

          Reference
Prediction    1    2    3    4    5    6
         1 4607   19    0  478  126    0
         2 1193 2805    1  196    4    0
         3    0    0 4243    0    0    0
         4    0 2564    1 3924 1084    0
         5  101 1386    0    1 6863    0
         6    0    0    0    0    0 2819

Overall Statistics

               Accuracy : 0.7793
                 95% CI : (0.7747, 0.7838)
    No Information Rate : 0.2492
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7313
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
Sensitivity            0.7807  0.41408   0.9995   0.8532   0.8497  1.00000
Specificity            0.9765  0.94563   1.0000   0.8688   0.9389  1.00000
Pos Pred Value         0.8809  0.66802   1.0000   0.5182   0.8218  1.00000
Neg Pred Value         0.9524  0.85934   0.9999   0.9728   0.9496  1.00000
Prevalence             0.1820  0.20898   0.1310   0.1419   0.2492  0.08697
Detection Rate         0.1421  0.08653   0.1309   0.1211   0.2117  0.08697
Detection Prevalence   0.1613  0.12954   0.1309   0.2336   0.2576  0.08697
Balanced Accuracy      0.8786  0.67986   0.9998   0.8610   0.8943  1.00000
```

The final step will store the accuracy value in a dataframe called *accatable*. Since the methodology can be run with different parameters (number of images, sensor, bands used), it is important to keep track of the result in each configuration to be able to compare the output with different settings.

```
# Save result in dataframe
if (!exists("accatable")) {
  accatable<-as.data.frame(matrix(nrow=5, ncol=t_images, dimnames = list(c("S1", "S2", "S1_S2", "S2*", "S1_S2*"),as.character(seq(1,t_images)))))
  accatable[nrow, n_images]<-CM$overall["Accuracy"]} else {
    accatable[nrow, n_images]<-CM$overall["Accuracy"]}
accatable
```

```
> accatable
          1  2  3  4         5  6  7
S1       NA NA NA NA        NA NA NA
S2       NA NA NA NA        NA NA NA
S1_S2    NA NA NA NA        NA NA NA
S2*      NA NA NA NA        NA NA NA
S1_S2* NA NA NA NA 0.7792997 NA NA
```

You can now repeat the classification with different settings (number of images, bands, sensor) and compare the performance. Go back to section *4.4.2 User input*, change the parameters you want and re-run the code. The consecutive results will be saved in *accatable*.

THANK YOU FOR FOLLOWING THE EXERCISE!

5 Further reading and resources

An Introduction to R

R Cheat Sheets

Breiman, L. (2001). Random Forests. *Machine Learning, 45, 5–32, 45*(1), 5–32.

Mentch, L., & Hooker, G. (2016). Quantifying Uncertainty in Random Forests via Confidence Intervals and Hypothesis Tests. Journal of Machine Learning Research, 17(1), 1–41. http://doi.org/10.1080/10618600.2016.1256817

FOLLOW US!!!

@RUS-Copernicus

RUS-Copernicus

RUS-Copernicus

RUS Copernicus Training

RUS-Copernicus website

RUS-Copernicus Training website