CIBIO-InBIO Advanced Course - Models in Invasion Ecology 2019 | Practical session on Remote Sensing - part II | Running an object-based image analysis for mapping areas invaded by Acacia dealbata

Introduction

In this exercise we are going to map areas invaded by *Acacia dealbata* in **Vilar da Veiga**, a mountainous area located in Peneda-Gerês National Park, NW Portugal. Total precipitation in the are amounts to 1300 mm/year and mean annual temperature to 13 C, with large amplitude in mean annual values between cold and warm seasons (-10 to 35 C). Elevation ranges from 145 to 1253m with steep slopes.

Methods

To map the actual distribution of *A. dealbata*, we will use one WorldView-2 (WV2) image (with eight spectral bands) and training data (i.e., hand-digitized patches delineated using the WV2 image and after field visits to the site). These data contains both invaded and non-invaded patches.

The figure below presents a RGB composite of the WV2 image with training areas overlapped.

Input image data

The Worldview-2 (WV2) image tat we are going to use was collected during summer (23 June 2013). WV2 platform records data in eight spectral bands with a ground resolution of 2 m:

- [B1] Coastal blue (400 - 450nm),
- [B2] Blue (450 - 510nm),
- [B3] Green (510 - 580nm),
- [B4] Yellow (590 - 630nm),
- [B5] Red (630 - 690nm),
- [B6] Red edge (710 - 750nm),
- [B7] Near-Infrared 1 (770 - 900nm), and
- [B8] Near Infrared 2 (860 - 1040nm).

Imagery pre-processing consisted in the orthorectification of the scene using a Digital Elevation Model (DEM; 20 m spatial resolution) and a bilinear convolution algorithm based on six ground control points, yielding average Root Mean Square Error values (RMSE) of 1.45 m. Surface reflectance was obtained by further processing the data with atmospheric correction.

Object-based Image Analysis (OBIA)

An Object-based Image Analysis (OBIA) approach will be employed in this exercise. The workflow is presented in the figure below.

Figure 1: Acacia dealbata picture

To map the target species we will use *SegOptim* - a R package developed in ECOCHANGE team used to perform object-based analyses of EO data. *SegOptim* allows to interface multiple GIS/RS analysis software, such as Orfeo Toolbox (OTB), GRASS, SAGA, etc. and use them to perform image segmentation.

Using the segmented image, we can then run supervised classification, which allows combining field or digitized training data with spectral image features to obtain maps with the actual distribution of target species.

SegOptim uses three main input layers described in the figure below.

A bried description of *SegOptim* package inputs:

- **Training data**: typically a single-layer raster dataset containing samples for training a classifier.
- **Segmentation features**: typically a multi-layer raster dataset with features used only for the segmentation stage (e.g., spectral bands, spectral indices, texture). This data will be used by the segmentation algorithm to group similar neighboring pixels into objects (or 'super-pixels').
- **Classification features**: also a multi-layer raster dataset with features used for classification (e.g., spectral bands and indices, texture, elevation, slope). This will allow the classification algorithm to extract/learn the spectral profile and get the classification rules.

Use R help system to get more info on *SegOptim* functions.

Now, let's code!

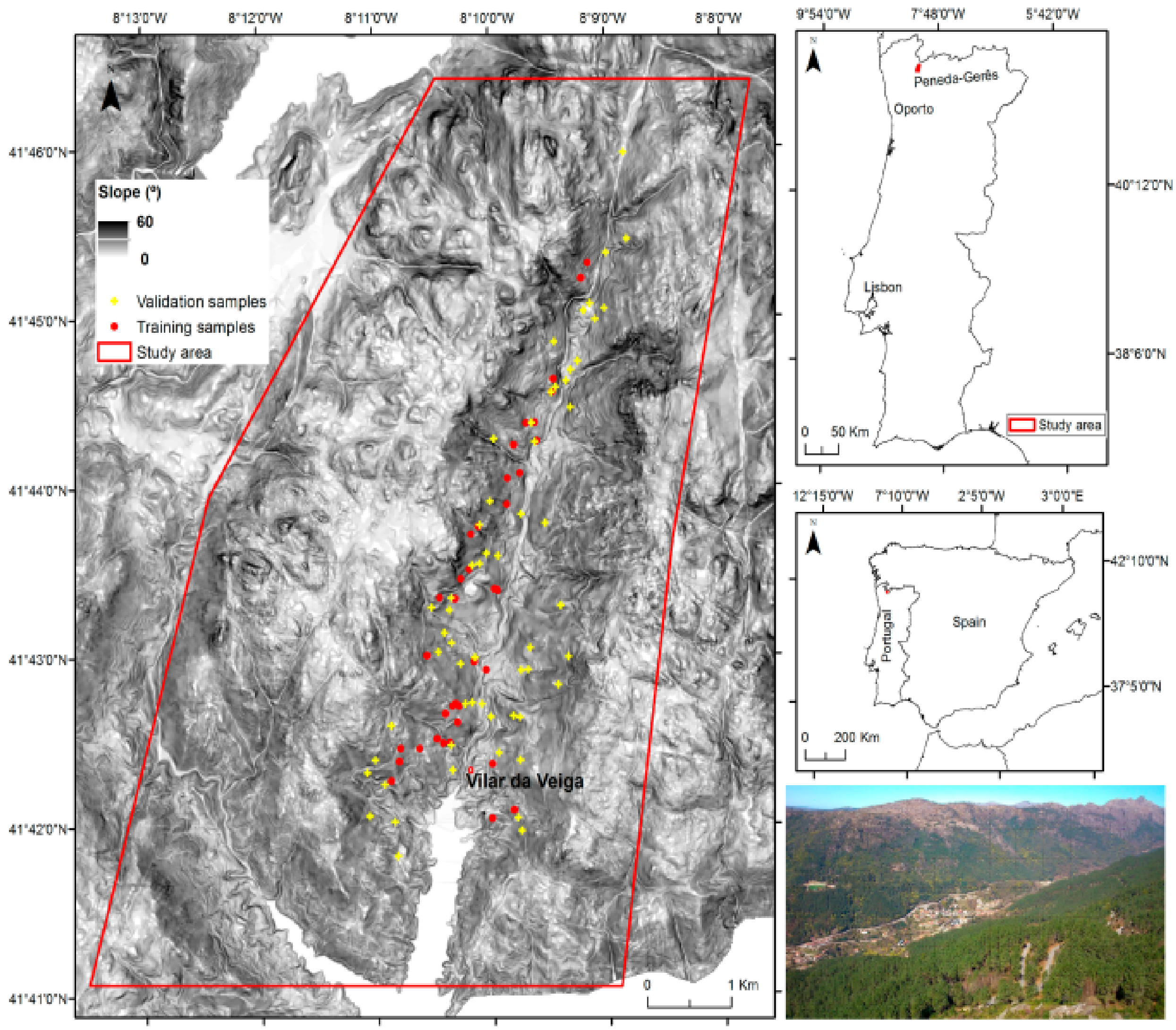


Figure 2: Study-area location in NW Portugal

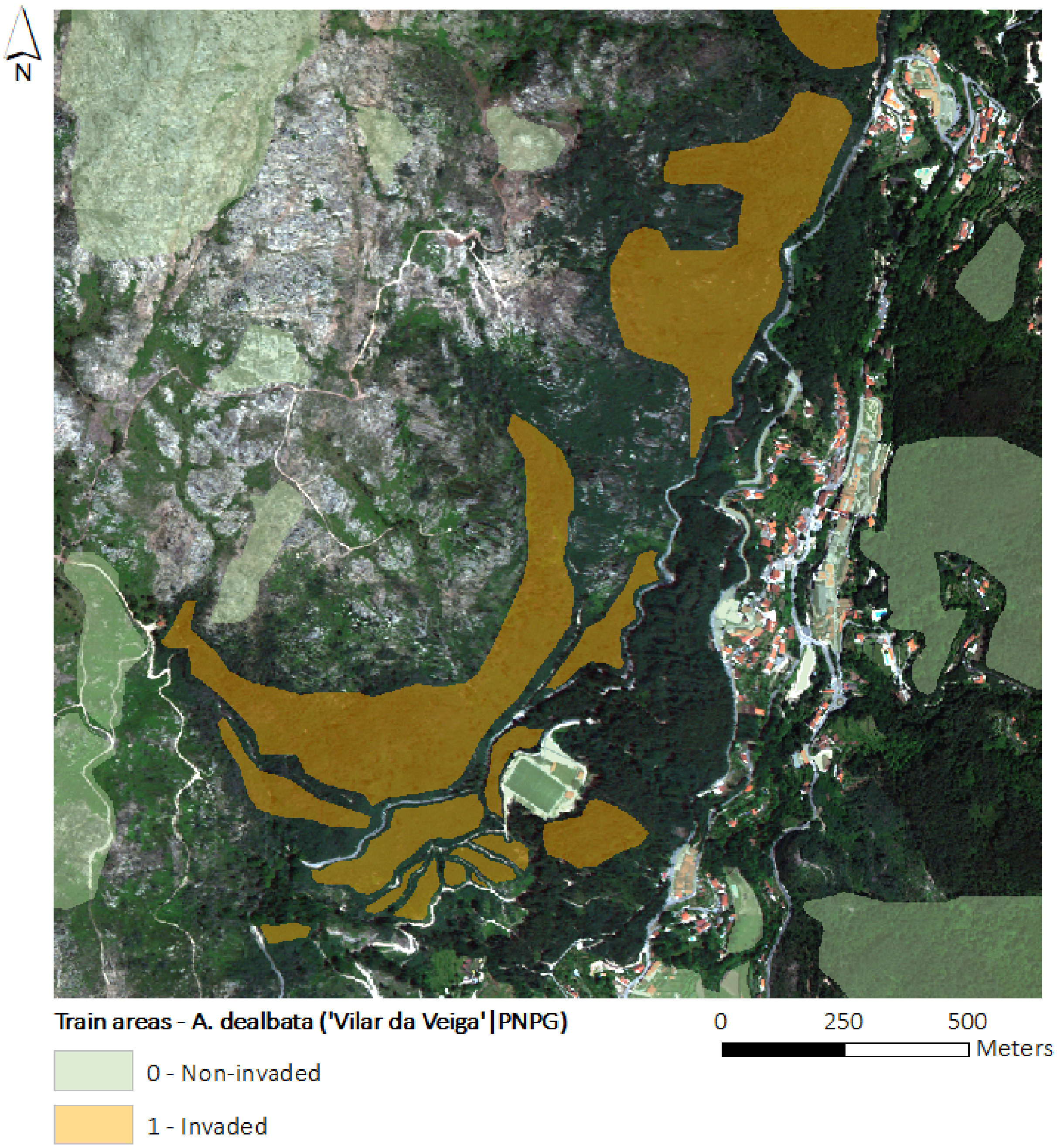


Figure 3: RGB composite showing training areas in the landscape

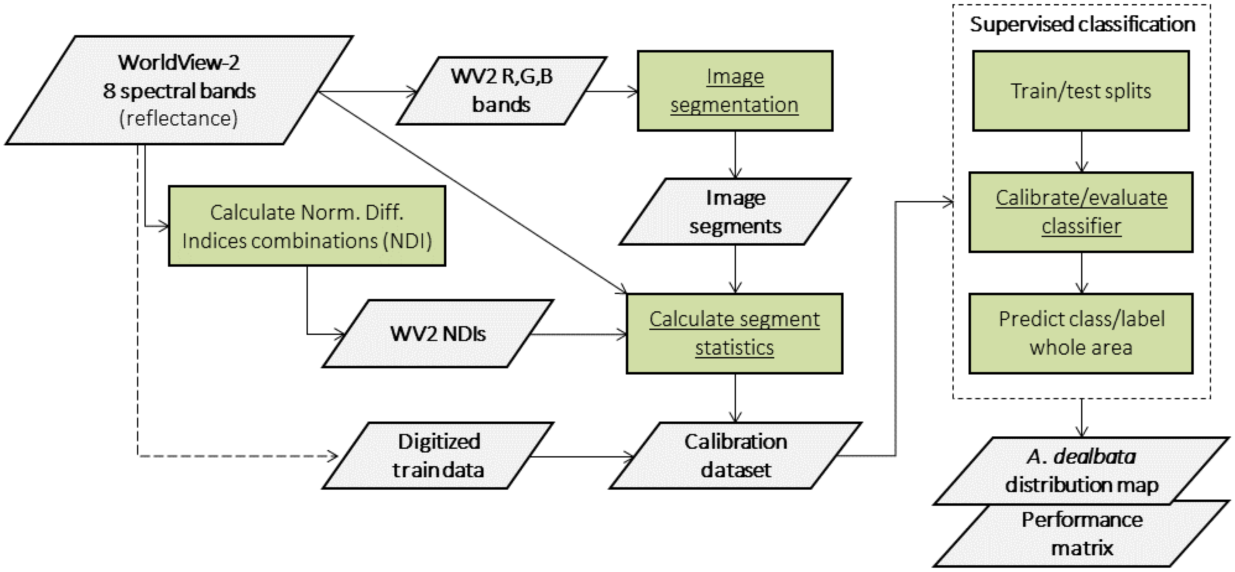


Figure 4: Exercise workflow

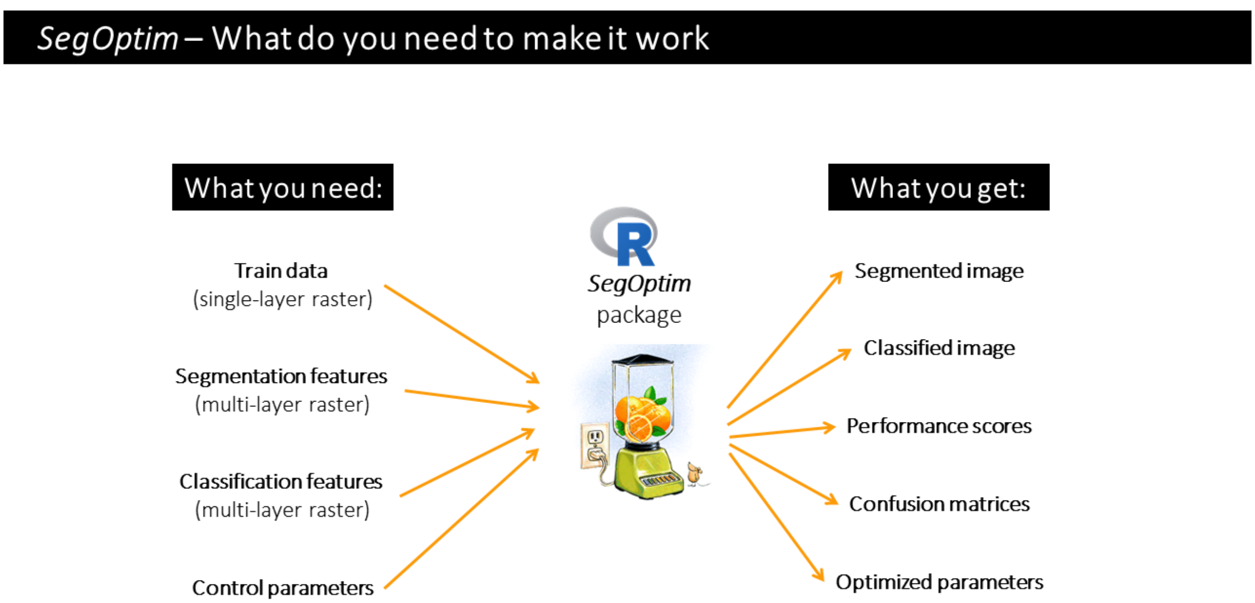


Figure 5: SegOptim package inputs and outputs

Inputs

First, define the correct working directory according to your own settings (change what is inside the “”):

```r
setwd("D:/MyDocs/R-dev/ECOCHANGE/AdvancedCourse_ModsInvasionEco2019/Day1_RemoteSensing_UAVs")
```

Now load the libraries and define the data inputs:

```r
library(raster)
library(randomForest)
library(SegOptim)

# Path to raster data used for image segmentation
# In SEGM_FEAT directory
inputSegFeat.path <- "./SEGM_FEAT/WV2_b532_VilarVeiga_smallTestSite.tif"

# Path to training raster data
# [0] Non-invaded areas [1] Acacia dealbata invaded areas
# In TRAIN_AREAS directory
trainData.path <- "./TRAIN_AREAS/trainAreas_Adealbata_VVeiga_WV2_v1.tif"

# Path to raster data used as classification features
# In CLASSIF_FEAT directory
classificationFeatures.path <-
  c("./CLASSIF_FEAT/WV2_NDIcombns_VilarVeiga_smallTestSite.tif",
    "./CLASSIF_FEAT/WV2_VilarVeiga_smallTestSite.tif")

# Path to Orfeo Toolbox binaries
otbPath <- "C:/OTB/bin"
```

Outputs

Now, let´s define the ouput files generated by the analyses:

```r
## Output file from OTB image segmentation
outSegmRst.path <- "segmRaster.tif"

# The final output file containing the distribution of the target species
outClassRst.path <- "WV2_VilarVeiga_AcaciaSpp.tif"
```

Run OTB image segmentation

This section will use *SegOptim* interface functions to run OTB’s Large Scale Mean Shift (LSMS) image segmentation algorithm. Segmentation involves the partitioning of an image into a set of jointly exhaustive and mutually disjoint regions (a.k.a. segments, composed by multiple image pixels), that are internally more homogeneous and similar, compared to adjacent ones. Image segments are then related to geographic objects of interest (e.g., forests, agricultural or urban areas) through some form of object-based classification (the following step in this exercise).

```r
## To know more about the algorithm and its parameters use the help
?segmentation_OTB_LSMS

## Run the segmentation
outSegmRst <- segmentation_OTB_LSMS(
                      # Input raster with features/bands to segment
                      inputRstPath = inputSegFeat.path,
                      # Algorithm params
                      SpectralRange = 3.1,
                      SpatialRange = 4.5,
                      MinSize = 21,
                      # Output
                      outputSegmRst = outSegmRst.path,
                      verbose = TRUE,
                      otbBinPath = otbPath,
                      lsms_maxiter = 50)
```

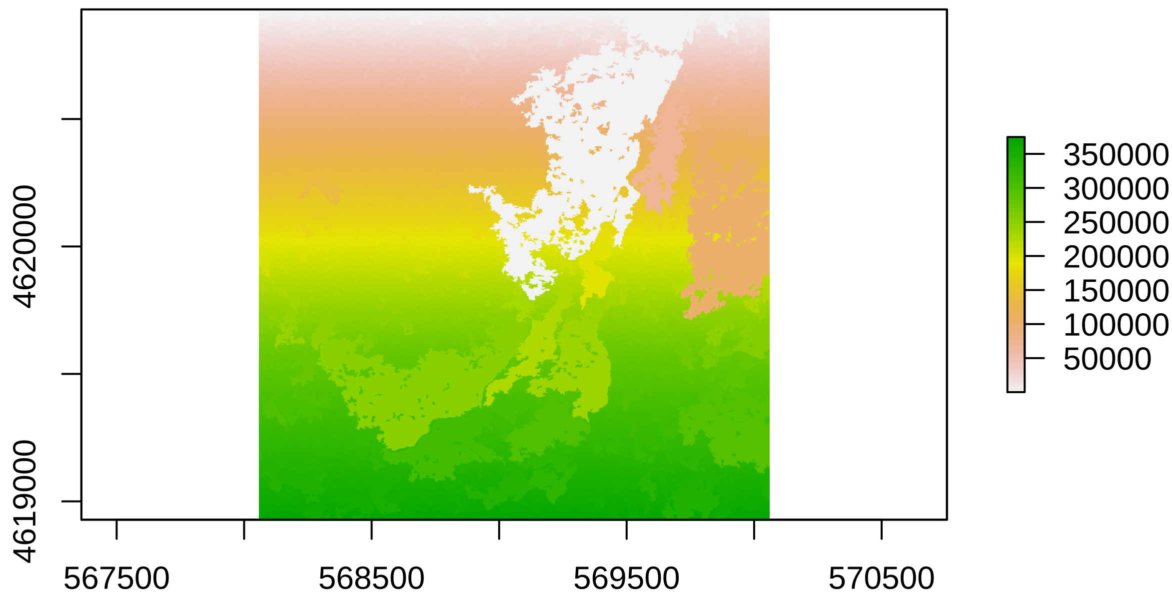
```
## SpectralRange = 3.1
## SpatialRange = 4.5
## MinSize = 21
```

```r
# Check the file paths with outputs
print(outSegmRst)
```

```
## $FilteredRange
## [1] "./otb_filt_range_uir92l.tif"
##
## $FilteredSpatial
## [1] "./otb_filt_spatial_4vct7z.tif"
##
## $Segmentation
## [1] "./otb_segm_init_7gk02c.tif"
##
## $segm
## [1] "segmRaster.tif"
##
## attr(,"class")
## [1] "SOptim.SegmentationResult"
```

```r
# Load the segmented raster and plot it
segmRst <- raster(outSegmRst$segm)

plot(segmRst)
```



You can use a GIS software to better inspect and visualize the output segmented raster. Notice that each segment is indexed by an integer number.

Load train data and classification features

This code chunk is used to load train data and classification features. Train data for *SegOptim* is simply a raster file with digitized training cases. These are later imported into image segments as we will see. In our case, the file contains areas invaded by *A. dealbata* (1's) and non-invaded (0's).

As for classification features, the files contain:

- Surface reflectance for the eight WV2 bands;
- All combinations (without order) of **Normalized Difference Indices** (NDI; 28 in total), calculated as:

$$NDI = \frac{(b_i - b_j)}{(b_i + b_j)}$$

Where b_i and b_j are any two different spectral bands. In total we will use 36 features (or variables) for supervised classification. Run the following code chunk to load the data using *raster* package functionalities.

```
# Train data
trainDataRst <- raster(trainData.path)

# Classification features
```

```r
classificationFeatures <- stack(classificationFeatures.path)
# Change the names for each layer
names(classificationFeatures) <- c(paste("NDI_",1:28,sep=""),paste("SpecBand_",1:8,sep=""))
```

Prepare the calibration dataset

In this step we will assemble the training data required to the supervised classification. This will import training data into each image segment (via a threshold rule). In this case, segments that are covered in 50% or more by the training pixels will be considered as valid cases either for 0's (non-invaded) or 1's (invaded).

We will also use this functions to calculate some statistics from the classification features raster. This will aggregate pixel values in each segment through the average function (other aggregation measures can be used; e.g. std.-dev., median, quantiles, skewness, kurtosis).

Using this function will produce an object of class *SOptim.CalData* containing two elements:

1. **calData** - A data frame object containing calibration data for training and evaluating a classifier algorithm. The first column (named "SID") contains the ID of each segment, and the second column (named "train") holds the segment class (or label). The following n columns hold the classification features for training;
2. **classifFeatData** - A data frame containing all segments and features from inputs. The first column (named "SID") holds the unique identifier for each image segment. The following n columns are used as classification features. Typically this dataset is used for predicting the target class after calibrating a certain classifier algorithm.

Run the following code chunk to generate the calibration dataset.

```r
# Check help for details
?prepareCalData


calData <- prepareCalData(rstSegm = segmRst,
                          trainData = trainDataRst,
                          rstFeatures = classificationFeatures,
                          thresh = 0.5,
                          funs = "mean",
                          minImgSegm = 30,
                          verbose = TRUE)
```

```
## -> [1/3] Loading train data into image segments...
## done.
##
## -> [2/3] Calculating feature statistics for all image segments...
## done.
##
## -> [3/3] Merging train and feature data...
## done.
```

Calibrate/evaluate the supervised classifier

We will use the training data in combination with satellite data (NDIs and original spectral bands) to map the distribution of *A. dealbata*.

The Random Forest (RF) algorithm will be used for this purpose. RF are an ensemble learning method for classification, regression and other tasks, that operate by constructing multiple decision trees during the training stage and outputting the class that is the mode of the classes (classification) or the average prediction (regression) of the individual trees. This way, RF correct for decision trees' habit of over-fitting to their training set.

For evaluation purposes, we will employ 10 fold cross-validation (i.e., ten splits with 90% train / 10% test).

Cohen's *Kappa* was selected as the evaluation metric (although others can be calculated later).

A minimum of 30 total train cases were set, and in each train/test split we set 10 for training and 5 for testing as the minimum. Notice that the *runFullCalibration* option must always be set to *TRUE* (check help for more details).

```
# Check the function help
?calibrateClassifier

# Calibrate/evaluate the RF classifier
classifObj <- calibrateClassifier( calData                   = calData,
                                   classificationMethod      = "RF",
                                   balanceTrainData          = FALSE,
                                   balanceMethod             = "ubOver",
                                   evalMethod                = "10FCV",
                                   evalMetric                = "Kappa",
                                   minTrainCases             = 30,
                                   minCasesByClassTrain      = 10,
                                   minCasesByClassTest       = 5,
                                   runFullCalibration        = TRUE)
```

```
## ## --- TRAINING ROUND 1 --- ##
##
## .. Frequency table by class for train data:
##
##    0    1
## 1945  157
##
## .. Frequency table by class for test data:
##
##   0   1
## 220  14
##
## .. Evaluation round #1 | N(train) = 2102 | N(test) = 234 | Kappa = 0.929
##
## ## --- TRAINING ROUND 2 --- ##
##
## .. Frequency table by class for train data:
##
##    0    1
## 1949  153
##
## .. Frequency table by class for test data:
##
##   0   1
## 216  18
```

```
##
## .. Evaluation round #2 | N(train) = 2102 | N(test) = 234 | Kappa = 0.828
##
## ## --- TRAINING ROUND 3 --- ##
##
## .. Frequency table by class for train data:
##
##    0    1
## 1943  159
##
## .. Frequency table by class for test data:
##
##   0   1
## 222  12
##
## .. Evaluation round #3 | N(train) = 2102 | N(test) = 234 | Kappa = 0.772
##
## ## --- TRAINING ROUND 4 --- ##
##
## .. Frequency table by class for train data:
##
##    0    1
## 1957  145
##
## .. Frequency table by class for test data:
##
##   0   1
## 208  26
##
## .. Evaluation round #4 | N(train) = 2102 | N(test) = 234 | Kappa = 0.978
##
## ## --- TRAINING ROUND 5 --- ##
##
## .. Frequency table by class for train data:
##
##    0    1
## 1946  156
##
## .. Frequency table by class for test data:
##
##   0   1
## 219  15
##
## .. Evaluation round #5 | N(train) = 2102 | N(test) = 234 | Kappa = 0.837
##
## ## --- TRAINING ROUND 6 --- ##
##
## .. Frequency table by class for train data:
##
##    0    1
## 1954  149
##
## .. Frequency table by class for test data:
##
```

```
##   0   1
## 211  22
##
## .. Evaluation round #6 | N(train) = 2103 | N(test) = 233 | Kappa = 0.923
##
## ## --- TRAINING ROUND 7 --- ##
##
## .. Frequency table by class for train data:
##
##    0    1
## 1949  154
##
## .. Frequency table by class for test data:
##
##   0   1
## 216  17
##
## .. Evaluation round #7 | N(train) = 2103 | N(test) = 233 | Kappa = 0.937
##
## ## --- TRAINING ROUND 8 --- ##
##
## .. Frequency table by class for train data:
##
##    0    1
## 1946  157
##
## .. Frequency table by class for test data:
##
##   0   1
## 219  14
##
## .. Evaluation round #8 | N(train) = 2103 | N(test) = 233 | Kappa = 0.882
##
## ## --- TRAINING ROUND 9 --- ##
##
## .. Frequency table by class for train data:
##
##    0    1
## 1947  156
##
## .. Frequency table by class for test data:
##
##   0   1
## 218  15
##
## .. Evaluation round #9 | N(train) = 2103 | N(test) = 233 | Kappa = 0.857
##
## ## --- TRAINING ROUND 10 --- ##
##
## .. Frequency table by class for train data:
##
##    0    1
## 1949  153
##
```

```
## .. Frequency table by class for test data:
##
##   0   1
## 216  18
##
## .. Evaluation round #10 | N(train) = 2102 | N(test) = 234 | Kappa = 0.873
##
## ## --- FULL DATASET TRAINING ROUND --- ##
##
## .. Frequency table by class for train data:
##
##    0    1
## 2165  171
##
## .. Frequency table by class for test data:
##
##    0    1
## 2165  171
##
## .. Evaluation round #11 | N(train) = 2336 | N(test) = 2336 | Kappa = 0.867
##
## .. Average Kappa = 0.882
```

```r
# Get more evaluation measures
evalMatrix <- evalPerformanceClassifier(classifObj)

print(round(evalMatrix,2))
```

```
##             Kappa Kappa_Thresh  PSS PSS_Thresh  GSS GSS_Thresh  AUC
## TestSet_1    0.93         0.41 0.99       0.41 0.99       0.41 1.00
## TestSet_2    0.83         0.36 0.92       0.22 0.92       0.22 0.99
## TestSet_3    0.77         0.26 0.96       0.10 0.96       0.10 0.99
## TestSet_4    0.98         0.26 0.96       0.10 0.96       0.26 1.00
## TestSet_5    0.84         0.65 0.97       0.28 0.97       0.28 1.00
## TestSet_6    0.92         0.50 0.96       0.22 0.96       0.22 1.00
## TestSet_7    0.94         0.30 0.97       0.08 0.97       0.08 1.00
## TestSet_8    0.88         0.45 0.98       0.15 0.98       0.15 1.00
## TestSet_9    0.86         0.44 0.98       0.38 0.98       0.38 0.99
## TestSet_10   0.87         0.47 0.92       0.18 0.92       0.18 0.99
## FULL         0.87         0.49 0.95       0.18 0.95       0.18 0.99
##            AUC_Thresh
## TestSet_1        0.43
## TestSet_2        0.30
## TestSet_3        0.13
## TestSet_4        0.40
## TestSet_5        0.29
## TestSet_6        0.28
## TestSet_7        0.10
## TestSet_8        0.16
## TestSet_9        0.38
## TestSet_10       0.18
## FULL             0.19
```

Finally, predict the class label for the entire image (i.e., outside the training set) and also save the classified image:

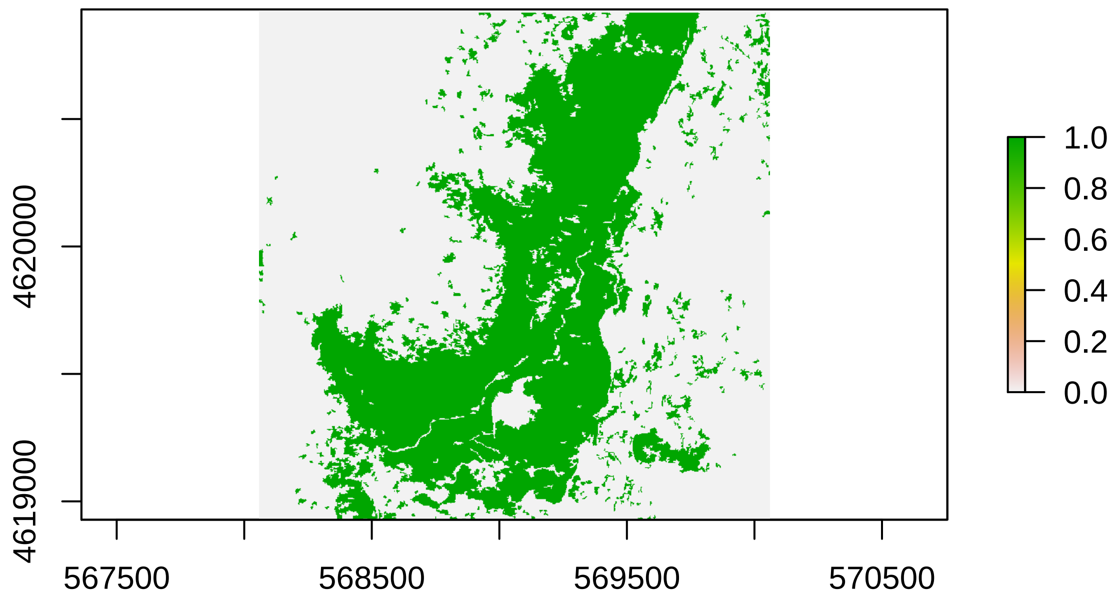
```r
rstPredSegmRF <- predictSegments(classifierObj = classifObj,
                                calData = calData,
                                rstSegm = segmRst,
                                predictFor = "all",
                                filename = outClassRst.path)
```

```
## -> Pre-processing data...
## done.
##
## -> Classifying raster segments...
## done.
##
## -> Merging data and creating new raster dataset with classifier predictions...
## done.
##
## -> Writing data...
## done.
```

```r
print(rstPredSegmRF)
```

```
## class       : RasterLayer
## dimensions  : 1001, 1001, 1002001  (nrow, ncol, ncell)
## resolution  : 2, 2  (x, y)
## extent      : 568058, 570060, 4618928, 4620930  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=utm +zone=29 +datum=WGS84 +units=m +no_defs +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 0, 1  (min, max)
```

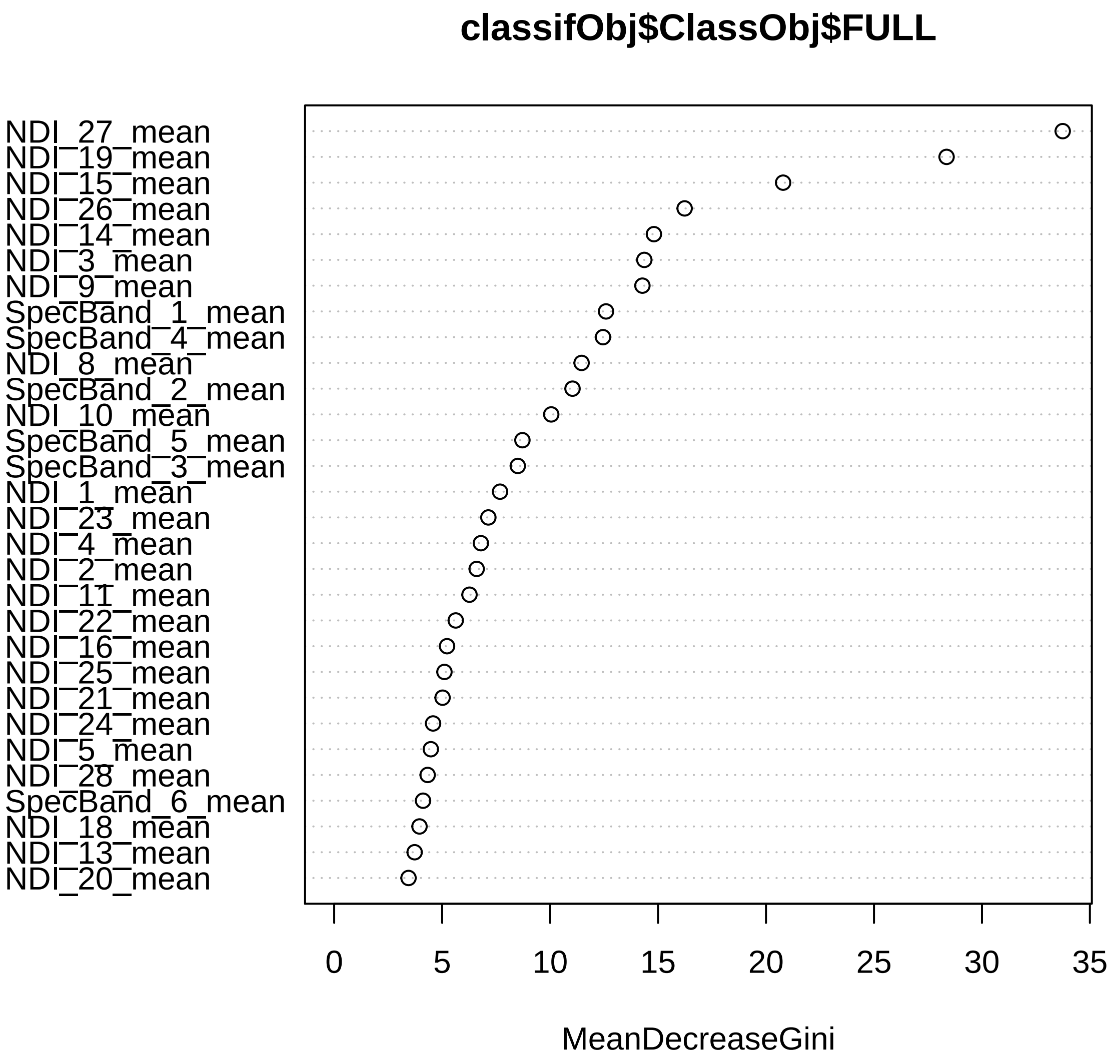
```r
plot(rstPredSegmRF)
```



As before, you can use a GIS software to better inspect and visualize the output classified raster.

You can further inspect other elements in the *classifObj* object, such as the most important features for the RF classifier. Play the code below.

```
# Variable importance
varImpPlot(classifObj$ClassObj$FULL)
```


classifObj$ClassObj$FULL
NDI_27_mean
NDI_19_mean
NDI_15_mean
NDI_26_mean
NDI_14_mean
NDI_3_mean
NDI_9_mean
SpecBand_1_mean
SpecBand_4_mean
NDI_8_mean
SpecBand_2_mean
NDI_10_mean
SpecBand_5_mean
SpecBand_3_mean
NDI_1_mean
NDI_23_mean
NDI_4_mean
NDI_2_mean
NDI_11_mean
NDI_22_mean
NDI_16_mean
NDI_25_mean
NDI_21_mean
NDI_24_mean
NDI_5_mean
NDI_28_mean
SpecBand_6_mean
NDI_18_mean
NDI_13_mean
NDI_20_mean
0
5
10
15
20
25
30
35
MeanDecreaseGini