Classification in R: Preparing Samples

We want to put the training samples created in the previous assignment in the form of a data frame – which most classifiers in R can handle easily (especially our RF and SVM algorithms).

Installing required packages, prerequisites, import

First of all, open RStudio. In order to use functionalities of the raster package, load it into your current session via `library()`.

```
#install.packages("raster")
library(raster)
```

Next: set your working directory, in which all your image and shapefile data is stored by giving a character (do not forget the quotation marks " ") variable to `setwd()`. Check your path with `getwd()` and the stored files in it via `dir()`:

```
setwd("~/YOUR_PATHFILE_HERE /Classification_R")
getwd()
"C:/YOUR_PATHFILE_HERE/Classification_R"
dir()
"LS8_Bonn_SC20200925100850.tif" "training_data.dbf"
"training_data.prj"             "training_data.shp"
"training_data.shx"
```

If you do not get your files listed, you have made a mistake in your work path – check again! Everything ready to go? Fine, then import your raster file as `img` and your shapefile as `shp` and have a look at them:

```
img <- brick("LS8_Bonn_SC20200925100850.tif")
img
class      : RasterBrick
dimensions : 681, 877, 597237, 7  (nrow, ncol, ncell, nlayers)
resolution : 30, 30  (x, y)
extent     : 357195, 383505, 5606985, 5627415  (xmin, xmax, ymin, ymax)
...

shp <- shapefile("training_data.shp")
shp
class       : SpatialPolygonsDataFrame
features    : 23
extent      : 363194.4, 375692.2, 5613568, 5622967  (xmin, xmax, ymin,
ymax)
crs         : +proj=utm +zone=32 +datum=WGS84 +units=m +no_defs variables
: 1
...
```

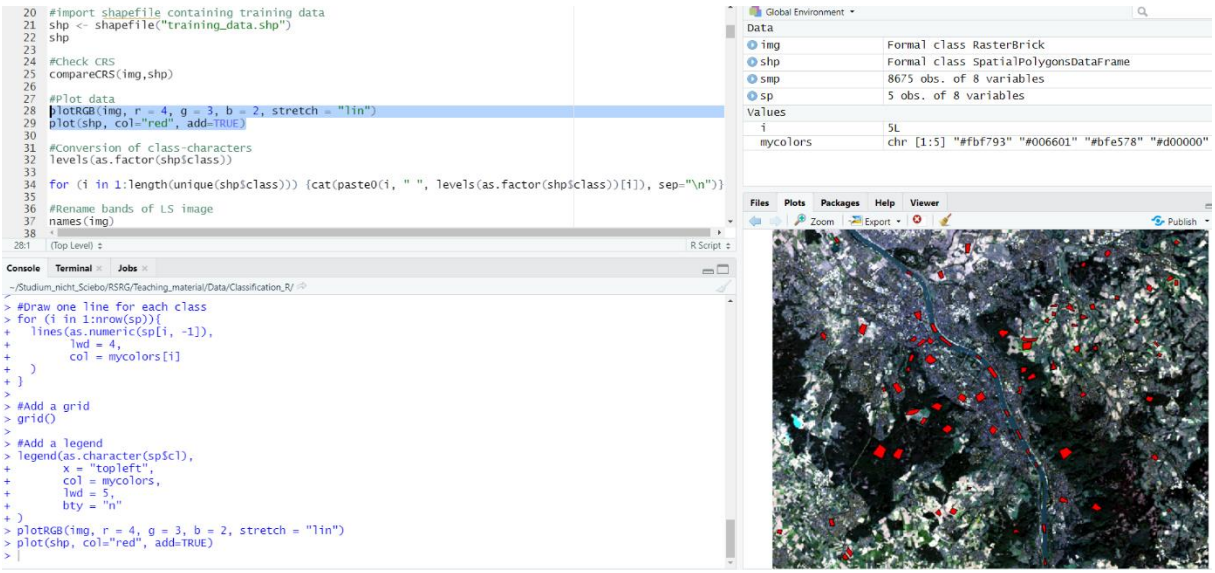
Both the `brick()` and `shapefile()` functions are provided by the *raster* package. As shown above, they create objects of the class *RasterBrick* and *SpatialPolygonsDataFrame* respectively. The L8 raster provides 7 bands, and our example shapefile 23 features, i.e., polygons. You can check whether the projections of the two datasets are identical or not by executing `compareCRS(shp, img)`. If this is not the case (output equals *FALSE*), the raster package will automatically re-project

your data on the fly later on. However, it is recommendable to adjust the projections manually in advance to prevent any future inaccuracies.

Plotting the data

Plot your data to make sure everything is imported properly. With the argument `add = TRUE` in line 2 several data layers can be displayed one above the other:

```
plotRGB(img, r = 4, g = 3, b = 2, stretch = "lin")
plot(shp, col="red", add=TRUE)
```



You should see something similar like above.

Prepare your classification

Your shapefile should provide an attribute called “class”, which includes your target classes as strings, e.g., “water” or “urban”. We will later turn this column into the factor data type because classifiers can only work with integer values instead of words like “water” or “urban”. When converting to factors, strings are sorted alphabetically and numbered consecutively. In order to be able to read the classification image at the end, you should make a note of your classification key:

```
levels(as.factor(shp$class))
"field"     "forest"    "grassland" "urban"     "water"

for (i in 1:length(unique(shp$class))) {cat(paste0(i, " ",
levels(as.factor(shp$class))[i]), sep="\n")}
1 field
2 forest
3 grassland
4 urban
5 water
```

The loop assigns digits from 1 to 5 to the characters of your class. The `levels()` function combines all occurrences in a factor-formatted vector. In the example shown above, value 1 in our classification image will correspond to the field class, value 2 to forest, value 3 to grassland, etc.

Let us take a look at the naming of the raster bands via the `names()` function. Those names can be quite bulky and cause problems in some illustrations when used as axis labels. You can easily rename it to something more concise by overriding the names with any string vector of the same length:

```
names(img)
"LS8_Bonn_SC20200925100850.1" "LS8_Bonn_SC20200925100850.2"
"LS8_Bonn_SC20200925100850.3" "LS8_Bonn_SC20200925100850.4"
"LS8_Bonn_SC20200925100850.5" "LS8_Bonn_SC20200925100850.6"
"LS8_Bonn_SC20200925100850.7"
names(img) <- c("b1", "b2", "b3", "b4", "b5", "b6", "b7")
names(img)
"b1" "b2" "b3" "b4" "b5" "b6" "b7"
```

Create dataframe

Our goal is to extract the raster values (*x*), i.e., all input feature values, and the class values (*y*) of every single pixel within our training polygons and put all together in a data frame. This data frame can then be read by our classifier. We extract the raster values using the command `extract()` from the raster package. The argument `df = TRUE` guarantees that the output is a data frame:

```
smp <- extract(img, shp, df = TRUE)
```

It may take some (long) time for this function to complete depending on the spatial resolution of your raster data and the spatial area covered by your polygons. If you have your data stored on an SSD, the process is completed much faster.

The data frame has as many rows as pixels could be extracted and as many columns as input features are given (in this example the spectral channels). If you like, check this out via the preview function (click on `"smp"` in your environment. In addition, `smp` also provides a column named “ID”, which holds the IDs of the former polygon for each pixel (each polygon is automatically assigned an ID). Furthermore, we also know which polygon, i.e., each ID, belongs to which class. Because of this, we can establish a relationship between the deposited ID of each pixel and the class using the `match()` function. We use this to add another column to our data query describing each class. Then we delete the ID column because we do not need it anymore:

```
smp$cl <- as.factor(shp$class[match(smp$ID, seq(nrow(shp)))])
smp <- smp[-1]
```

It is advisable to save the resulting object to the hard drive and load it from your working directory. On this way, the extract function does not have to be repeated again and again…

```
#Save dataframe to your wd
save(smp, file = "smp.rda")

#Load dataframe from your wd
load(file = "smp.rda")
```

Now, we print the summary of our new column via `summary(smp$cl)`. You can see the assignment of our former ID column as our desired classes from the training data:

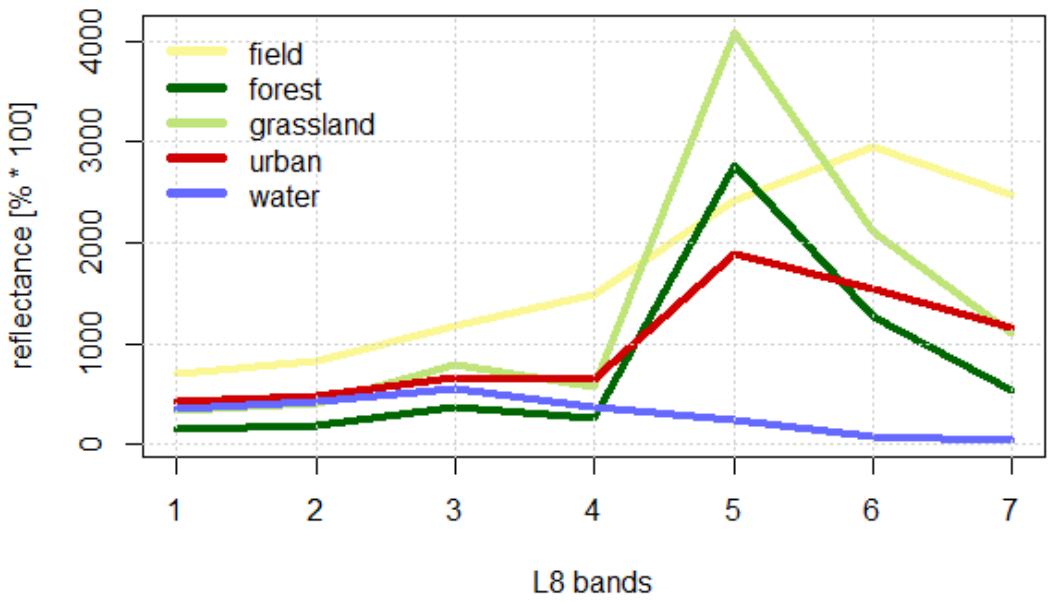
```
    field    forest grassland     urban     water

     1677      2483      1364      2139      1012
```

Plot spectral profile

If you only include spectral information in your classifier, as in our example, it is often helpful to plot the so-called spectral profiles, or z-profiles. Those represent the mean values of each class for the individual spectral bands. You can also represent other features, e.g., terrain height or precipitation, however, you must then pay attention to the value range in the presentation and possibly normalize the data at first. The magic here happens in the `aggregate()` command, which combines all the rows of the same class `. ~ cl` and calculates the arithmetic mean of those groups `FUN = mean`. This happens for all classes, in the `cl` column, where `NA` values are to be ignored via `na.rm = TRUE`. The rest of the functions in the following script are for visualization purposes only and include standard functions such as `plot()`, `lines()`, `grid()` and `legend()`. Use the help function for a detailed description of the arguments!

```
#Aggregate cl-column
sp <- aggregate( . ~ cl, data = smp, FUN = mean, na.rm = TRUE )

#Plot empty plot of a defined size
plot(0,
     ylim = c(min(sp[2:ncol(sp)]), max(sp[2:ncol(sp)])),
     xlim = c(1, ncol(smp)-1),
     type = 'n',
     xlab = "L8 bands",
     ylab = "reflectance [% * 100]"
)

#Define colors for class representation - one color per class necessary!
mycolors <- c("#fbf793", "#006601", "#bfe578", "#d00000", "#6569ff")

#Draw one line for each class
for (i in 1:nrow(sp)){
  lines(as.numeric(sp[i, -1]),
        lwd = 4,
        col = mycolors[i]
  )
}

#Add a grid
grid()

#Add a legend
legend(as.character(sp$cl),
       x = "topleft",
       col = mycolors,
       lwd = 5,
       bty = "n"
)
```

The given code should result in the following plot:



Note that the values represent only the arithmetic mean of the classes and do not allow any statement about the underlying distribution. However, such a z-profile plot helps to visually assess the separation of classes at the beginning.

Now you are ready to start training your classifier as described in the next assignment!

Questions / prove your knowledge:

- How can you plot several layers one above the other, e.g. in order to plot both your remote sensing image and the training data?
- Why is it essential to extract the raster values, all input feature values and the class values to a data frame?
- What does the spectral profile represent?