Data pre-processing in R, part 1/3

Reading and visualizing raster data

Multiband satellite images are raster data and thus three-dimensional arrays of numerical values. Each image band consists of rows and columns of numerical pixel values that are displayed as gray values. Pixels with identical X- and Y-coordinates in different bands account for the third dimension. We can thus easily treat image data as data frames or matrices in R. Similarly, most remote sensing operations are based on multivariate statistical techniques that you may already know from your introductory classes to multivariate statistics. Of course, the size of the data sets that often include several thousands or millions of pixels and the data formats may be challenging. Still, with the help of the R packages 'raster' and 'rgdal', an analysis of remote sensing data in R is quite comfortable.

Open a new R script file and load the packages 'raster' and 'rgdal' into your workspace. If the packages are not already available, they have to be installed before they can be loaded.

```
library (raster)
library (rgdal)
```

The functions in the raster package are relying on the rgdal package, which is the R implementation of the dgal library. This library is an open source library for handling geospatial data. It enables us to read file formats such as GEOTIFF, ENVI HDR, and others or to convert data from one projection to another. The raster package provides us with three different function to read raster data in different dgal-supported formats. These functions are raster(), stack(), and brick(). The raster() function reads single band raster data such as a black-and-white image or a digital elevation model. It can also be used to read a single band of a multi-band image or to create a new raster file. You will learn the different variation to use this functions in this class. Raster data sets with just a single band are called raster layer according to the R naming convention. The stack() function reads simultaneously all bands of a multi-band image. It can also be used to combine multiple raster layers to a multi-band image (similar to the 'merge' function in PCI geomatics). In both cases, the output of the stack() function is stored in the memory of your computer and reffered to as a raster stack. The brick() function is similarly able to read multi-band raster data. Differing from stack() it only establishes a link to the storage place of the image on your hard-drive and does not read it into the memory. This link is called a raster brick. On the one hand, a raster brick saves memory as the image is only read when needed. On the other hand, the computation time increases considerably. As a rule of thumb, stack() is faster than brick() as long as enough memory is available.

Table 1.1: Spectral bands of Landsat 5 TM and Landsat 8.

Spectral region	**Landsat 5 TM**	**Landsat 8 OLI & TIRS**
Coastal / aerosole		Band 1 (433 - 453 nm)
Blue	Band 1 (450 - 520 nm)	Band 2 (450 - 515 nm)
Green	Band 2 (520 - 600 nm)	Band 3 (525 - 600 nm)
Red	Band 3 (630 - 690 nm)	Band 4 (630 - 680 nm)
NIR	Band 4 (760 - 900 nm)	Band 5 (845 - 885 nm)
SWIR A	Band 5 (1550 - 1750 nm)	Band 6 (1560 - 1660 nm)
SWIR B	Band 7 (2080 - 2350 nm)	Band 7 (2100 - 2300 nm)
Cirrus		Band 9 (1360 - 1390 nm)
Thermal A	Band 6 (10400 - 12500 nm)	Band 10 (10300 - 11300 nm)
Thermal B		Band 11 (11500 - 12500 nm)
Panchromatic		Band 8 (500 - 680 nm)

Today, we start working with two Landsat images of the Trinity and Shasta lake area. Both images were taken in early September, the first one by Landsat 5 TM on September 9th, 2011, the second by Landsat 8 OLI & TIRS on September 1st, 2014. Both images were downloaded from the USGS Earth Explorer and have not been further processed. We want to use these images to answer the following questions: ***To what extent has the surface of the Shasta and Trinity lake reservoirs changed from 2011 to 2014? Which county (adminstrative unit) shows the largest change?***

Unzip the two *.zip files containing the two images and set your working directory to the folder containing the unzipped image data.

```
    setwd ("PATH TO YOUR FOLDER")
```

Each image band is available as single Geotiff-file. The file name contains the year and Julian day of the acquisition as well as the band number. Table 1.1 gives an overview on the spectral coverage of the bands. In addition, a meta file (_MTL.txt) is provided. This file contains important information that will be needed later in this class.

We use the raster() function to read all bands of the Landsat 5 TM image as individual objects in our workspace.

```
    blue2011 <- raster ("LT50450322011252PAC01_B1.TIF")
    green2011 <- raster ("LT50450322011252PAC01_B2.TIF")
    red2011 <- raster ("LT50450322011252PAC01_B3.TIF")
    nir2011 <- raster ("LT50450322011252PAC01_B4.TIF")
    swira2011 <- raster ("LT50450322011252PAC01_B5.TIF")
    swirb2011 <- raster ("LT50450322011252PAC01_B7.TIF")
    the2011 <- raster ("LT50450322011252PAC01_B6.TIF")
```

Then we combine all but the thermal band to a multi-band image using stack().

```
    ls52011 <- stack (blue2011, green2011, red2011, nir2011, swira2011, swirb2011)
```

The raster package offers different plot-functions to visualize the imagery. The plot() function plots a single band as black-and -white or pseudocolor image with a coordinate grid and legend; the plotRGB() function allows to combine three bands to a RGB image. After using this function it may be necessary to clear all graphic settings with dev.off(). PlotRGB has different options for contrast enhancements that can be chosen with the 'stretch' parameter. The most common enhancements are the 'lin' and 'hist' stretches.

```
    plot (nir2011) ## plots the NIR band with the default inverted terrain.colors()
                   ## gradient
    plot (nir2011, col=gray.colors (255, 0, 1, 1)) ## plots the NIR band as black-
                   ## and-white image.
    plotRGB(ls52011, 4, 3, 2, stretch="lin") ## plots a RGB composite of bands 4
                   ## (R), 3 (G), and 2 (B) with a linear contrast stretch
```

Q1.1: What happens if you pass the multi-band image stack to plot()?

Each spectral band is plotted as an individual image.

Q1.2: The plot() function accepts several other color gradients. Which are available?

rainbow, heat.colors, topo.colors, bpy.colors, terrain.colors. Note that - differing from gray.colors - only the number of color intervals needs to be specified (e.g., rainbow (100) for a rainbow gradient). Gray.colors() requires the number of intervals, the darkest and brightest hues - normally 0 and 1 - and a gamma value.

Q1.3: You can build your own custom color gradients with the function colorRampPalette(). Play around with this functions and try to create a color gradient from blue to yellow to red. How does it work?

```
    my.gradient <- colorRampPalette (c ("blue", "yellow", "red"))
    plot (nir2011, col=my.gradient (100))
```

Instead of reading only a single band at a time, we can read and stack multiple files simultaneously. For this purpose, we first generate a character vector with the respective file names and then use this vector and the stack function to create a raster stack. The function dir() lists the names of all files in your working directory. With the argument 'pattern' this list can be restricted to files that have a distinct character expression in their name. All Landsat 8 OLI bands have the expression "LC80450322014244LGN00" in their file name.

```
bands <- dir(pattern="LC80450322014244LGN00")
```

The file names are listed in alphabetical (10, 11, 1, 2, …) and not in numerical order (1, 2, …, 10, 11). Since we are only interested in bands 2 to 7 that correspond to the visible, NIR and SWIR bands of Landsat 5 TM (see table 1), we further restrict this list to its elements 4 to 9.

```
bands <- bands[4:9]
```

This vector bands can now be used to read and stack the seven bands in a single line of code.

```
ls82014 <- stack (bands)
plotRGB (ls82014, 4, 3, 2, stretch="lin")
```

We will later need the two thermal bands and thus read them as raster layers into our workspace.

```
the1.2014 <- raster ("LC80450322014244LGN00_B10.TIF")
the2.2014 <- raster ("LC80450322014244LGN00_B11.TIF")
```

Getting information on raster data sets

Often, we need to know more about the raster data sets we are working with. Frequently needed information includes the spatial resolution of the pixels, the number of bands of a raster stack, the number of rows, columns, and pixels, the projection, as well as the extent of the image. The raster package provides several function that allow to extract these informations from the data set.

```
res (ls82014) ## determines the spatial resolution of a pixel in X- and Y-
              ## direction
```

Q1.4: Does this resolution match the expected resolution of Landsat data?

Each pixel has a resolution of 30 m x 30 m and thus matches the expected resolution for these Landsat bands.

```
projection (ls82014) ## determines the projection of the data set in the gdal
nomenclature
```

Q1.5: Which projection is assigned to the data?

'+proj=utm +zone=10 +datum=WGS84 +units=m +no_defs +ellps=WGS84 +towgs84=0,0,0', translating to UTM Zone 10N, WGS 84.

```
dim (ls82014) ## determines the dimensionality (rows of pixels, columns of
              ## pixels, bands)
nlayers (ls82014) ## determines the number of bands of a raster stack
extent (ls82014) ## determines the extent of the bounding box of a raster data
              ## set and returns the range of X- and Y-coordinates
ncell(ls82014) ## determines the number of pixels (rows x columns) of a raster
              ## data set
```

Clipping raster data

Both Landsat images cover a larger extent than the Trinity and Shasta lake area. We thus clip the data sets to save computation time. Clipping raster data in R is often much easier and more comfortable than in any GIS software. The new extent can be displayed in a plotted image using the abline() function.

```
    plot (nir2011)
    abline (v=510000) ## draws a vertical line at the new xmin
    abline (v=580000) ## draws a vertical line at the new xmax
    abline (h=4490000) ## draws a horizontal line at the new ymin
    abline (h=4543000) ## draws a horizontal line at the new ymax
```

We use the function crop() to clip the raster data set to a new extent. This extent has to be defined by using the function extent() in a different way, assigning four numeric values as the new xmin, xmax, ymin, and ymax coodinates. Both raster stacks and the three thermal bands are cropped to the same extent.

```
    ls82014.subs <- crop (ls82014, extent (510000, 580000, 4490000, 4543000))
    ls52011.subs <- crop (ls52011, extent (510000, 580000, 4490000, 4543000))
    ls5the.subs <- crop (the2011, extent (510000, 580000, 4490000, 4543000))
    ls8the1.subs <- crop (the1.2014, extent (510000, 580000, 4490000, 4543000))
    ls8the2.subs <- crop (the2.2014, extent (510000, 580000, 4490000, 4543000))

    plotRGB (ls82014.subs, 3, 2, 1)
    plotRGB (ls52011.subs, 4, 3, 2)
```

Q1.6: What is the dimensionality of the clipped data?

1767 rows x 2333 columns x 7 bands

Q1.7: You have learned from using the projection() function that the images come in the UTM Zone 10N, WGS84 projection. The coordinates are thus in meters. How large is the area on the ground covered by the images in km²?

```
1767*2333*30*30/1000000
```

= 3710.17 km²

Eliminate bad pixels

Raw image data often contain some bad pixels, i.e., pixels with numeric values that are outside the expected data range. These pixels may cause interferences in the processing and analysis of the images and are thus eliminated. In our case, all pixels with negative numeric values are bad pixels. We eliminate these pixels by reclassifying all negative values to NA, i.e., missing values. A reclassification of pixel values can be done with the reclassify() function. This function requires a raster object and a three-column matrix as input. Each row of this matrix thus contains three entries that define a range of values (from-to, the first two entries) that are reclassified as a new value (the third entry). All pixel values not covered by the data ranges in the first two columns remain unchanged. Here, we want to set all values < 0 (but not values=0) to NA. The image consists of integer numbers, thus all values from -1 to – Infinity (-inf) need to be reclassified.

```
    ls82014.subs <- reclassify (ls82014.subs, rcl=matrix (c (-Inf, -1, NA), 1, 3))
    ls52011.subs <- reclassify (ls52011.subs, rcl=matrix (c (-Inf, -1, NA), 1, 3))
```

Q1.8: Read the documentation of reclassify(). How are the optional arguments 'include.lowest' and 'right' working?

'include.lowest' indicates whether the 'from' value in the from-to-table should be reclassified, too. 'right' indicates whether the 'to' value is included in the reclassification.

Save your workspace and script. We will continue working with it in the next session.

```
To save the work space : save.image(*.RData")
To load the work space : load("*.RData")
```