Remote Sensing Image Analysis with R

Aniruddha Ghosh and Robert J. Hijmans

Nov 30, 2023

CONTENTS

Aniruddha Ghosh and Robert J. Hijmans

CHAPTER

ONE

INTRODUCTION

This section provides a short introduction to satellite data analysis with R. Before reading this you should first learn the basics of the terra package.

Getting satellite images for a specific project remains a challenging task. You have to find data that is suitable for your objectives, and that you can get access to. Important properties to consider while searching the remotely sensed (satellite) data include:

1. Spatial resolution, that is the size of the grid cells
2. Temporal resolution, that is the return time or frequency that data is collected; as well as the availability of historical images, and for a particular moment in time
3. Spectral resolution, that is, the parts of the electromagnetic spectrum (wavelengths) for which measurements are made
4. Radiometric resolution (sensor sensitivity; ability to measure small differences)
5. Quality issues, such as the presence of cloud-cover or of artifacts in the data (read about problems in Landsat ETM+

There are numerous sources of remotely sensed data from satellites. Generally, the very high spatial resolution data is available as (costly) commercial products. Lower spatial resolution data is freely available from NASA, ESA, and other organizations. In this tutorial we'll use freely available Landsat 8, Landsat 7, Landsat 5, Sentinel and MODIS data. The Landsat program started in 1972 and is is the longest running Earth-observation satellite program.

You can access public satellite data from several sources, including:

i. http://earthexplorer.usgs.gov/
ii. https://lpdaacsvc.cr.usgs.gov/appeears/
iii. https://search.earthdata.nasa.gov/search
iv. https://lpdaac.usgs.gov/data_access/data_pool
v. https://scihub.copernicus.eu/
vi. https://aws.amazon.com/public-data-sets/landsat/

See this web site for more sources of freely available satellite remote sensing data.

It is possible to download some satellite data using R-packages. For example, you can use the luna, MODIS or MODIS-Tools package to search, download and pre-process different MODIS products.

1.1 Terminology

Most remote sensing products consist of observations of reflectance data. That is, they are measures of the intensity of the sun's radiation that is reflected by the earth. Reflectance is normally measured for different wavelengths of the electromagnetic spectrum. For example, it can be measured in the near-infrared, red, green, and blue wavelengths. If that is the case, satellite data can be referred to as "multi-spectral" (or hyper-spectral if there are many separate wavelengths)(reading)[https://gisgeography.com/multispectral-vs-hyperspectral-imagery-explained/.

The data are normally stored as raster data, and are generally referred to as "images". Each separate image (for a place and time) is referred to as as "scene" or "tile". As there are measurements in multiple wavelengths, a single "satellite image" has multiple observations for each pixel, that are stored in separate raster layers. In remote sensing jargon, these layers (variables) are referred to as "bands" as they typically represent reflectance values for a particular spectral bandwith, and grid cells are referred to as "pixels".

1.2 Data

You can download all the data required for the examples used in this book using the R code below.

```
dir.create("data", showWarnings = FALSE)
if (!file.exists("data/rs/samples.rds")) {
    download.file("https://biogeo.ucdavis.edu/data/rspatial/rs.zip", dest = "data/rs.zip
↪")
    unzip("data/rs.zip", exdir="data")
}
```

1.3 Resources

Here is a short list of some resources to learn more about remote sensing image analysis

- Remote Sensing Digital Image Analysis
- Introductory Digital Image Processing: A Remote Sensing Perspective
- A survey of image classification methods and techniques for improving classification performance
- A Review of Modern Approaches to Classification of Remote Sensing Data
- Online remote sensing course

CHAPTER

TWO

EXPLORATION

In this chapter we describe how to explore satellite remote sensing data with *R*. We also show how to use them to make maps.

We will primarily use a spatial subset of a Landsat 8 scene collected on June 14, 2017. The subset covers the area between Concord and Stockton, in California, USA.

All Landsat scenes have a unique product ID and metadata. You can find the information on Landsat sensor, satellite, location on Earth (WRS path, WRS row) and acquisition date from the product ID. For example, the product identifier of the data we will use is 'LC08_044034_20170614'. Based on this guide, you can see that the Sensor-Satellite is OLI/TIRS combined Landsat 8, WRS Path 44, WRS Row 34 and collected on June 14, 2017. Landsat scenes are most commonly delivered as separate files for each band, combined into a single zip file.

We will start by exploring and visualizing the data (See the instructions in Chapter 1 for data downloading instructions if you have not already done so).

2.1 Image properties

Create SpatRaster objects for single Landsat layers (bands)

```
library(terra)
## terra 1.7.62

# Blue
b2 <- rast('data/rs/LC08_044034_20170614_B2.tif')

# Green
b3 <- rast('data/rs/LC08_044034_20170614_B3.tif')

# Red
b4 <- rast('data/rs/LC08_044034_20170614_B4.tif')

# Near Infrared (NIR)
b5 <- rast('data/rs/LC08_044034_20170614_B5.tif')
```

Print the variables to check. E.g.

```
b2
## class       : SpatRaster
## dimensions  : 1245, 1497, 1  (nrow, ncol, nlyr)
## resolution  : 30, 30  (x, y)
```

(continues on next page)

(continued from previous page)

```
## extent      : 594090, 639000, 4190190, 4227540  (xmin, xmax, ymin, ymax)
## coord. ref. : WGS 84 / UTM zone 10N (EPSG:32610)
## source      : LC08_044034_20170614_B2.tif
## name        : LC08_044034_20170614_B2
## min value   :               0.0748399
## max value   :               0.7177562
```

You can see the spatial resolution, extent, number of layers, coordinate reference system and more.

2.2 Image information and statistics

The below shows how you can access various properties from a SpatRaster object.

```
# coordinate reference system (CRS)
crs(b2)
## [1] "PROJCRS[\"WGS 84 / UTM zone 10N\",\n    BASEGEOGCRS[\"WGS 84\",\n        DATUM[\
↪"World Geodetic System 1984\",\n            ELLIPSOID[\"WGS 84\",6378137,298.257223563,
↪\n                LENGTHUNIT[\"metre\",1]]],\n        PRIMEM[\"Greenwich\",0,\n␣
↪    ANGLEUNIT[\"degree\",0.0174532925199433]],\n        ID[\"EPSG\",4326]],\n␣
↪CONVERSION[\"UTM zone 10N\",\n        METHOD[\"Transverse Mercator\",\n            ID[\
↪"EPSG\",9807]],\n        PARAMETER[\"Latitude of natural origin\",0,\n␣
↪ANGLEUNIT[\"degree\",0.0174532925199433],\n            ID[\"EPSG\",8801]],\n␣
↪PARAMETER[\"Longitude of natural origin\",-123,\n            ANGLEUNIT[\"degree\",0.
↪0174532925199433],\n            ID[\"EPSG\",8802]],\n        PARAMETER[\"Scale factor␣
↪at natural origin\",0.9996,\n            SCALEUNIT[\"unity\",1],\n            ID[\
↪"EPSG\",8805]],\n        PARAMETER[\"False easting\",500000,\n            LENGTHUNIT[\
↪"metre\",1],\n            ID[\"EPSG\",8806]],\n        PARAMETER[\"False northing\",0,\
↪n            LENGTHUNIT[\"metre\",1],\n            ID[\"EPSG\",8807]]],\n␣
↪CS[Cartesian,2],\n        AXIS[\"(E)\",east,\n            ORDER[1],\n␣
↪LENGTHUNIT[\"metre\",1]],\n        AXIS[\"(N)\",north,\n            ORDER[2],\n␣
↪    LENGTHUNIT[\"metre\",1]],\n    USAGE[\n        SCOPE[\"Navigation and medium␣
↪accuracy spatial referencing.\"],\n        AREA[\"Between 126°W and 120°W, northern␣
↪hemisphere between equator and 84°N, onshore and offshore. Canada - British Columbia␣
↪(BC); Northwest Territories (NWT); Nunavut; Yukon. United States (USA) - Alaska (AK).\
↪"],\n        BBOX[0,-126,84,-120]],\n    ID[\"EPSG\",32610]]"

# Number of cells, rows, columns
ncell(b2)
## [1] 1863765
dim(b2)
## [1] 1245 1497    1

# spatial resolution
res(b2)
## [1] 30 30

# Number of layers (bands in remote sensing jargon)
nlyr(b2)
## [1] 1
```

(continues on next page)

(continued from previous page)

```
# Do the bands have the same extent, number of rows and columns, projection, resolution,␣
↪and origin
compareGeom(b2,b3)
## [1] TRUE
```

You can create a SpatRaster with multiple layers from the existing SpatRaster (single layer) objects.

```
s <- c(b5, b4, b3)
# Check the properties of the multi-band image
s
## class       : SpatRaster
## dimensions  : 1245, 1497, 3  (nrow, ncol, nlyr)
## resolution  : 30, 30  (x, y)
## extent      : 594090, 639000, 4190190, 4227540  (xmin, xmax, ymin, ymax)
## coord. ref. : WGS 84 / UTM zone 10N (EPSG:32610)
## sources     : LC08_044034_20170614_B5.tif
##               LC08_044034_20170614_B4.tif
##               LC08_044034_20170614_B3.tif
## names       : LC08_04403~0170614_B5, LC08_04403~0170614_B4, LC08_04403~0170614_B3
## min values  :          0.0008457669,            0.02084067,            0.04259216
## max values  :          1.0124315023,            0.78617686,            0.69246972
```

You can also create the multi-layer SpatRaster using the filenames.

```
# first create a list of raster layers to use
filenames <- paste0('data/rs/LC08_044034_20170614_B', 1:11, ".tif")
filenames
##  [1] "data/rs/LC08_044034_20170614_B1.tif"
##  [2] "data/rs/LC08_044034_20170614_B2.tif"
##  [3] "data/rs/LC08_044034_20170614_B3.tif"
##  [4] "data/rs/LC08_044034_20170614_B4.tif"
##  [5] "data/rs/LC08_044034_20170614_B5.tif"
##  [6] "data/rs/LC08_044034_20170614_B6.tif"
##  [7] "data/rs/LC08_044034_20170614_B7.tif"
##  [8] "data/rs/LC08_044034_20170614_B8.tif"
##  [9] "data/rs/LC08_044034_20170614_B9.tif"
## [10] "data/rs/LC08_044034_20170614_B10.tif"
## [11] "data/rs/LC08_044034_20170614_B11.tif"

landsat <- rast(filenames)
landsat
## class       : SpatRaster
## dimensions  : 1245, 1497, 11  (nrow, ncol, nlyr)
## resolution  : 30, 30  (x, y)
## extent      : 594090, 639000, 4190190, 4227540  (xmin, xmax, ymin, ymax)
## coord. ref. : WGS 84 / UTM zone 10N (EPSG:32610)
## sources     : LC08_044034_20170614_B1.tif
##               LC08_044034_20170614_B2.tif
##               LC08_044034_20170614_B3.tif
##               ... and 8 more source(s)
## names       : LC08_~14_B1, LC08_~14_B2, LC08_~14_B3, LC08_~14_B4,  LC08_~14_B5,  LC08_
↪~14_B6, ...
```

(continues on next page)

(continued from previous page)

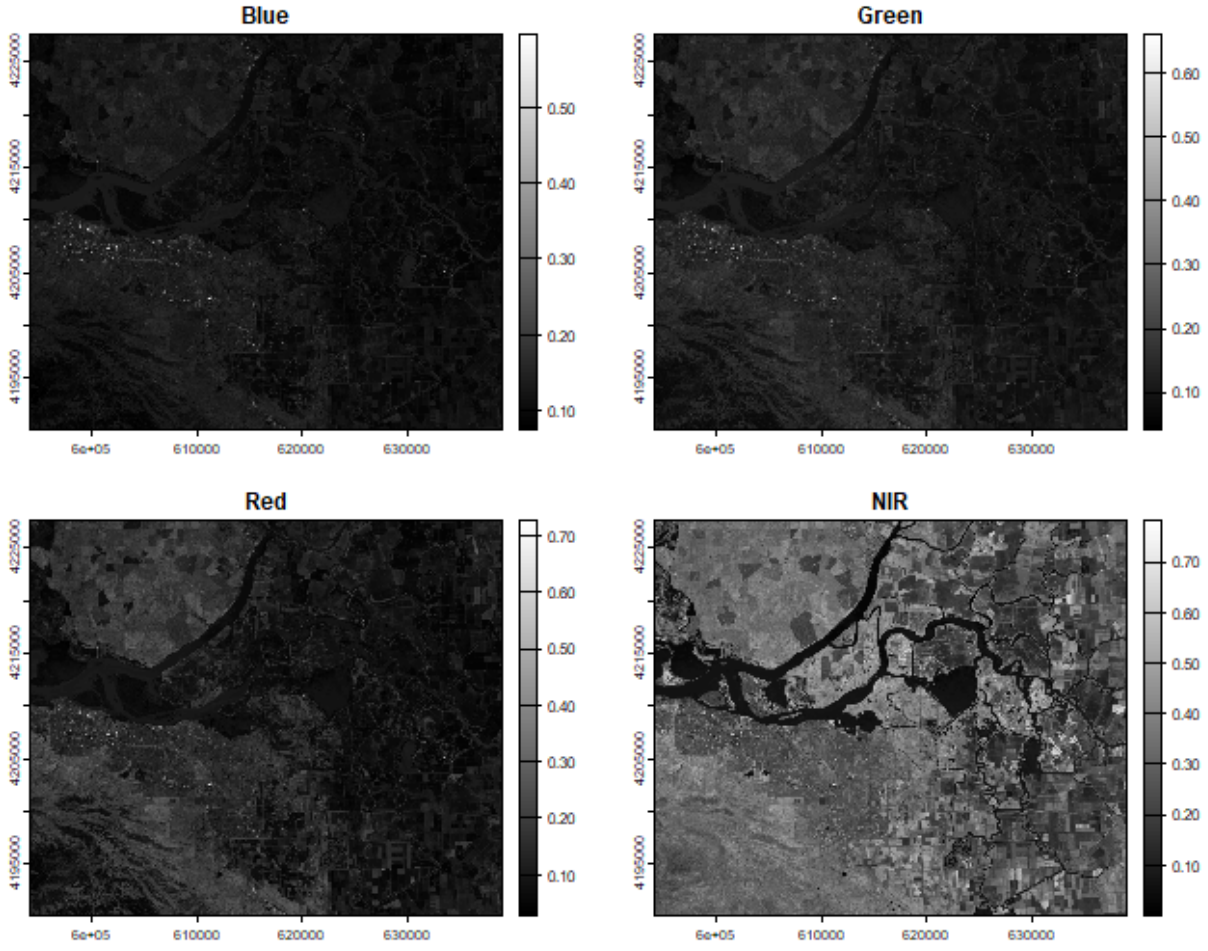
```
## min values  :  0.09641791,   0.0748399,  0.04259216,  0.02084067, 0.0008457669, -0.
↪007872183, ...
## max values  :  0.73462820,   0.7177562,  0.69246972,  0.78617686, 1.0124315023,  1.
↪043204546, ...
```

Above we created a SpatRaster with 11 layers. The layers represent reflection intensity in the following wavelengths: Ultra Blue, Blue, Green, Red, Near Infrared (NIR), Shortwave Infrared (SWIR) 1, Shortwave Infrared (SWIR) 2, Panchromatic, Cirrus, Thermal Infrared (TIRS) 1, Thermal Infrared (TIRS) 2.

2.3 Single band and composite maps

You can plot individual layers of a multi-spectral image.

```
par(mfrow = c(2,2))
plot(b2, main = "Blue", col = gray(0:100 / 100))
plot(b3, main = "Green", col = gray(0:100 / 100))
plot(b4, main = "Red", col = gray(0:100 / 100))
plot(b5, main = "NIR", col = gray(0:100 / 100))
```



The legends of the maps created above can range between 0 and 1. Notice the difference in shading and range of legends

between the different bands. This is because different surface features reflect the incident solar radiation differently. Each layer represent how much incident solar radiation is reflected for a particular wavelength range. For example, vegetation reflects more energy in NIR than other wavelengths and thus appears brighter. In contrast, water absorbs most of the energy in the NIR wavelength and it appears dark.

We do not gain that much information from these grey-scale plots; they are often combined into a "composite" to create more interesting plots. You can learn more about color composites in remote sensing here and also in the section below.

To make a "true (or natural) color" image, that is, something that looks like a normal photograph (vegetation in green, water blue etc), we need bands in the red, green and blue regions. For this Landsat image, band 4 (red), 3 (green), and 2 (blue) can be used. With `plotRGB` we can combine them into a single composite image. Note that use of `strecth = "lin"` (otherwise the image will be pitch-dark).

```
landsatRGB <- c(b4, b3, b2)
plotRGB(landsatRGB, stretch = "lin")
```

The true-color composite reveals much more about the landscape than the earlier gray images. Another popular image visualization method in remote sensing is known "false color" image in which NIR, red, and green bands are combined.

This representation is popular as it makes it easy to see the vegetation (in red).

```
landsatFCC <- c(b5, b4, b3)
plotRGB(landsatFCC, stretch="lin")
```

Question 1: *Now use the plotRGB function with the multi-band (11 layers) ``landsat`` SpatRaster to create a true and false color composite (hint remember the position of the bands).*

2.4 Subset and rename bands

You can select specific layers (bands) using `subset` function, or via indexing.

```
# select first 3 bands only
landsatsub1 <- subset(landsat, 1:3)
# same
landsatsub2 <- landsat[[1:3]]

# Number of bands in the original and new data
nlyr(landsat)
## [1] 11
nlyr(landsatsub1)
## [1] 3
nlyr(landsatsub2)
## [1] 3
```

We won't use the last four bands in `landsat`. You can remove those by selecting the ones we want.

```
landsat <- subset(landsat, 1:7)
```

For clarity, it is useful to set the names of the bands. (source)[https://www.usgs.gov/faqs/what-are-band-designations-landsat-satellites?qt-news_science_products=0#qt-news_science_products]

```
names(landsat)
## [1] "LC08_044034_20170614_B1" "LC08_044034_20170614_B2"
## [3] "LC08_044034_20170614_B3" "LC08_044034_20170614_B4"
## [5] "LC08_044034_20170614_B5" "LC08_044034_20170614_B6"
## [7] "LC08_044034_20170614_B7"
names(landsat) <- c('ultra-blue', 'blue', 'green', 'red', 'NIR', 'SWIR1', 'SWIR2')
names(landsat)
## [1] "ultra-blue" "blue"       "green"      "red"        "NIR"
## [6] "SWIR1"      "SWIR2"
```

2.5 Spatial subset or crop

Spatial subsetting can be used to limit analysis to a geographic subset of the image. Spatial subsets can be created with the `crop` function, using a `SpatExtent` object, or another spatial object from which an Extent can be extracted.

```
ext(landsat)
## SpatExtent : 594090, 639000, 4190190, 4227540 (xmin, xmax, ymin, ymax)
e <- ext(624387, 635752, 4200047, 4210939)

# crop landsat by the extent
landsatcrop <- crop(landsat, e)
```

Question 2: *Use the ``landsatcrop`` image to plot a true and false color composite*

2.6 Saving results to disk

At this stage we may want to save the raster to disk with `writeRaster`. Multiple file types are supported. We will use the commonly used GeoTiff format.

```
writeRaster(landsatcrop, filename="cropped-landsat.tif", overwrite=TRUE)
```

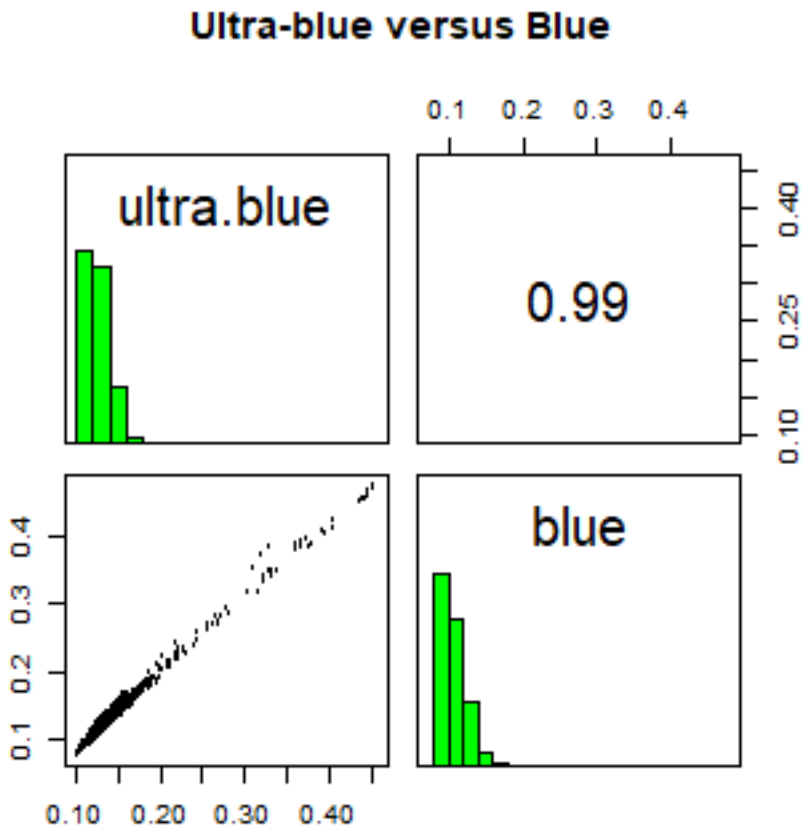
Note: Check for package documentation (`help(writeRaster)`) for additional helpful arguments that can be added.

2.7 Relation between bands

A scatterplot matrix can be helpful in exploring relationships between raster layers. This can be done with the `pairs` function.

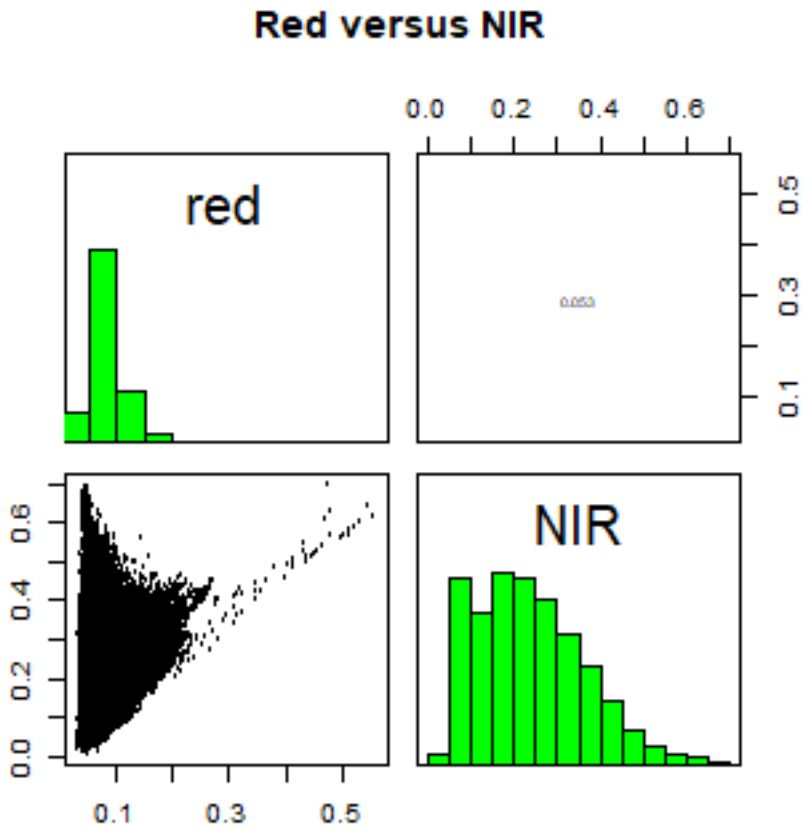
A plot of reflection in the ultra-blue wavelength against reflection in the blue wavelength.

```
pairs(landsatcrop[[1:2]], main = "Ultra-blue versus Blue")
```



A plot of reflection in the red wavelength against reflection in the NIR wavelength.

```
pairs(landsatcrop[[4:5]], main = "Red versus NIR")
```



The first plot reveals high correlations between the blue wavelength regions. Because of the high correlation, we can just use one of the blue bands without losing much information.

This distribution of points in second plot (between NIR and red) is unique due to its triangular shape. Vegetation reflects very highly in the NIR range than red and creates the upper corner close to NIR (y) axis. Water absorbs energy from all the bands and occupies the location close to origin. The furthest corner is created due to highly reflecting surface features like bright soil or concrete (see Baret et al)[http://www.ipgp.fr/~jacquemoud/publications/baret1993a.pdf].

2.8 Extract cell values

Often we want to get the values of raster cells (pixels in remote sensing jargon) for specific geographic locations or area. The `extract` function is used to get raster values at the locations of other spatial data. You can use points, lines, polygons or an Extent (rectangle) object. You can also use cell numbers to extract values. When using points, `extract` returns the values of a `SpatRaster` object for the cells in which a set of points fall.

```
# load the polygons with land use land cover information
samp <- readRDS('data/rs/lcsamples.rds')

# generate 50 point samples from the polygons

set.seed(555)
```

(continues on next page)

(continued from previous page)

```
ptsamp <- spatSample(samp, 50, 'regular')

# We use the x-y coordinates to extract the spectral values for the locations
df <- extract(landsat, ptsamp)

# To see some of the reflectance values
head(df)
##   ID ultra-blue       blue      green        red       NIR     SWIR1      SWIR2
## 1  1  0.1152851 0.09450950 0.09054089 0.06332441 0.4896148 0.1536918 0.07551219
## 2  2  0.1158706 0.09381554 0.08429519 0.05584258 0.4386517 0.1711277 0.08882765
## 3  3  0.1151116 0.09260110 0.09442276 0.05775099 0.5586210 0.1705638 0.07067610
## 4  4  0.1100153 0.08713611 0.08986861 0.05057278 0.4605116 0.1765059 0.07436280
## 5  5  0.1109695 0.08767828 0.08394821 0.04983544 0.6520245 0.1484437 0.04799209
## 6  6  0.1070009 0.08368797 0.06970022 0.04556321 0.3409542 0.1594387 0.07312667
```

2.9 Spectral profiles

A plot of the spectrum (all bands) for pixels representing a certain earth surface features (e.g. water) is known as a spectral profile. Such profiles demonstrate the differences in spectral properties of various earth surface features and constitute the basis for image analysis. Spectral values can be extracted from any multispectral data set using `extract` function. In the above example, we extracted values of Landsat data for the samples. These samples include: cropland, water, fallow, built and open. First we compute the mean reflectance values for each class and each band.

```
ms <- aggregate(df[,-1], list(ptsamp$class), mean)

# instead of the first column, we use row names
rownames(ms) <- ms[,1]
ms <- ms[,-1]
ms
##          ultra-blue       blue      green        red        NIR      SWIR1
## built     0.1715777 0.16020858 0.15984533 0.17228251 0.22624379 0.21830656
## cropland  0.1123755 0.08990475 0.08546264 0.05381490 0.49006296 0.16329528
## fallow    0.1348246 0.11801761 0.10233108 0.10520815 0.15550623 0.23542800
## open      0.1396925 0.13829443 0.15342000 0.20654889 0.34112771 0.35808941
## water     0.1342292 0.11744489 0.10053341 0.07981103 0.04949535 0.03381014
##               SWIR2
## built    0.18381969
## cropland 0.07174958
## fallow   0.21624273
## open     0.21559214
## water    0.02739490
```

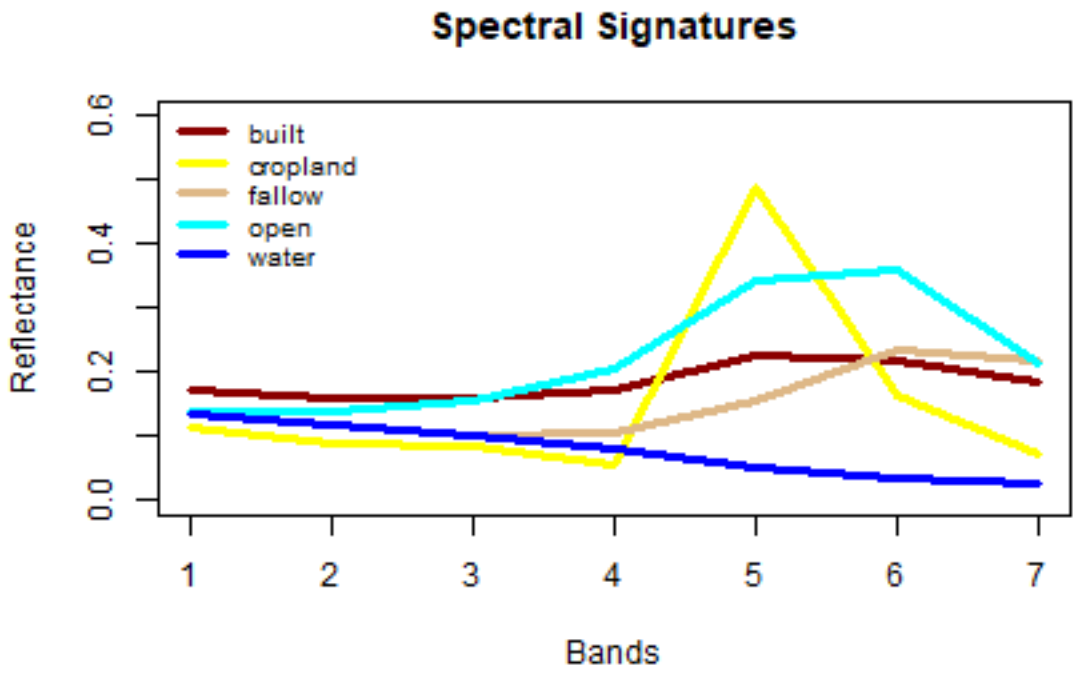
Now we plot the mean spectra of these features.

```
# Create a vector of color for the land cover classes for use in plotting
mycolor <- c('darkred', 'yellow', 'burlywood', 'cyan', 'blue')

#transform ms from a data.frame to a matrix
ms <- as.matrix(ms)
```

(continues on next page)

(continued from previous page)

```
# First create an empty plot
plot(0, ylim=c(0,0.6), xlim = c(1,7), type='n', xlab="Bands", ylab = "Reflectance")

# add the different classes
for (i in 1:nrow(ms)){
  lines(ms[i,], type = "l", lwd = 3, lty = 1, col = mycolor[i])
}

# Title
title(main="Spectral Signatures", font.main = 2)

# Legend
legend("topleft", rownames(ms),
       cex=0.8, col=mycolor, lty = 1, lwd =3, bty = "n")
```



The spectral signatures (profile) shows (dis)similarity in the reflectance of different features on the earth's surface (or above it). 'Water' shows relatively low reflection in all wavelengths, and 'built', 'fallow' and 'open' have relatively high reflectance in the longer wavelengts.

CHAPTER

THREE

BASIC MATHEMATICAL OPERATIONS

The `terra` package supports many mathematical operations. Math operations are generally performed per pixel (grid cell). First we will do some basic arithmetic operations to combine bands. In the first example we write a custom math function to calculate the Normalized Difference Vegetation Index (NDVI). Learn more about vegetation indices here and NDVI.

We use the same Landsat data as in Chapter 2.

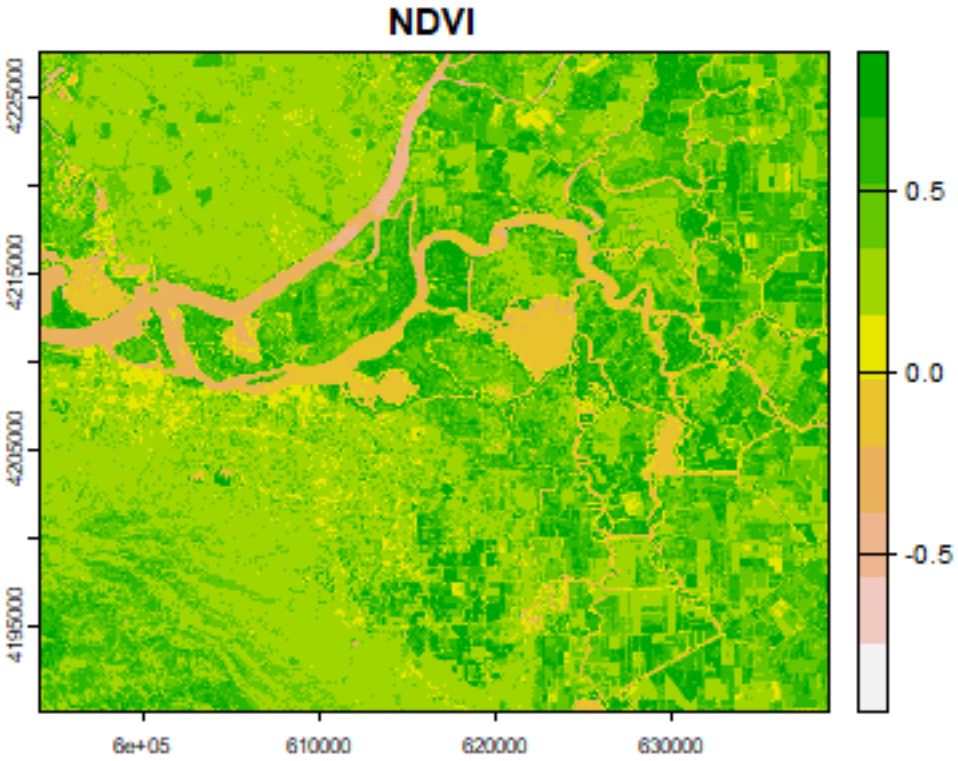
```
library(terra)
## terra 1.7.62
rfiles <- paste0('data/rs/LC08_044034_20170614_B', 1:11, ".tif")
landsat <- rast(rfiles)
landsatRGB <- landsat[[c(4,3,2)]]
landsatFCC <- landsat[[c(5,4,3)]]
```

3.1 Vegetation indices

Let's define a general function for a ratio based (vegetation) index. In the function below, `img` is a muti-layer SpatRaster object and `i` and `k` are the indices of the layers (layer numbers) used to compute the vegetation index.

```
vi <- function(img, k, i) {
  bk <- img[[k]]
  bi <- img[[i]]
  vi <- (bk - bi) / (bk + bi)
  return(vi)
}
```

```
# For Landsat NIR = 5, red = 4.
ndvi <- vi(landsat, 5, 4)
plot(ndvi, col=rev(terrain.colors(10)), main = "NDVI")
```



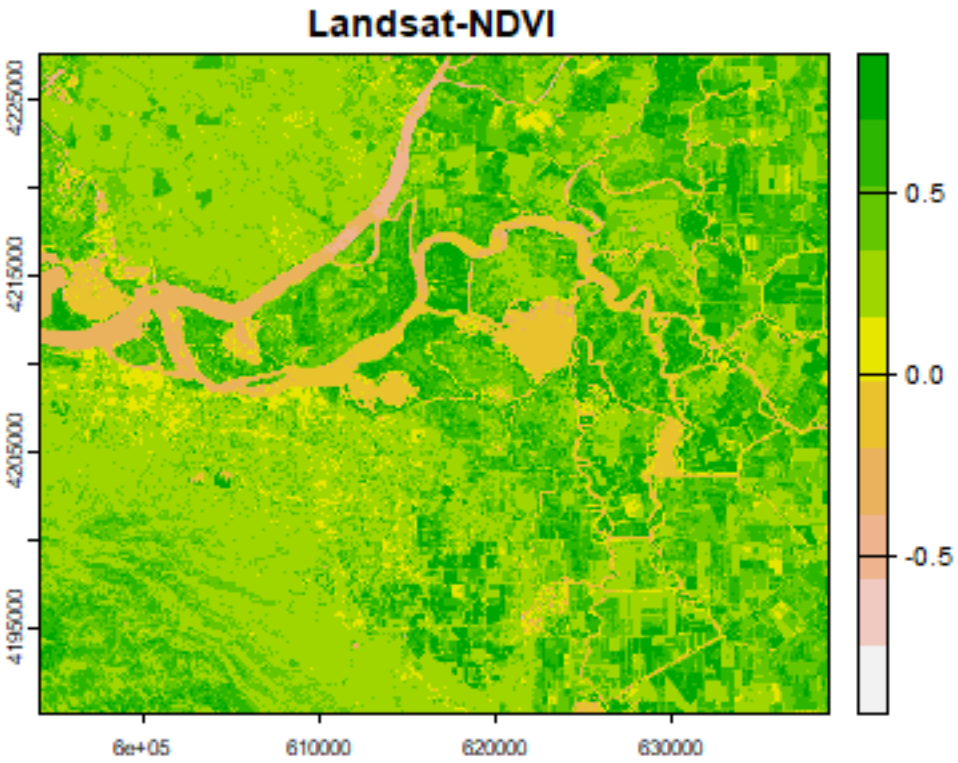
You can see the variation in greenness from the plot.

Below is an alternative way to accomplish this. First write a general function that can compute 2-layer NDVI type indices.

```
vi2 <- function(x, y) {
    (x - y) / (x + y)
}
```

And use that function as an argument in `lapp`

```
nir <- landsat[[5]]
red <- landsat[[4]]
ndvi2 <- lapp(c(nir, red), fun = vi2)

# or in one line
#ndvi2 <- lapp(landsat[[5:4]], fun=vi2)

plot(ndvi2, col=rev(terrain.colors(10)), main="Landsat-NDVI")
```

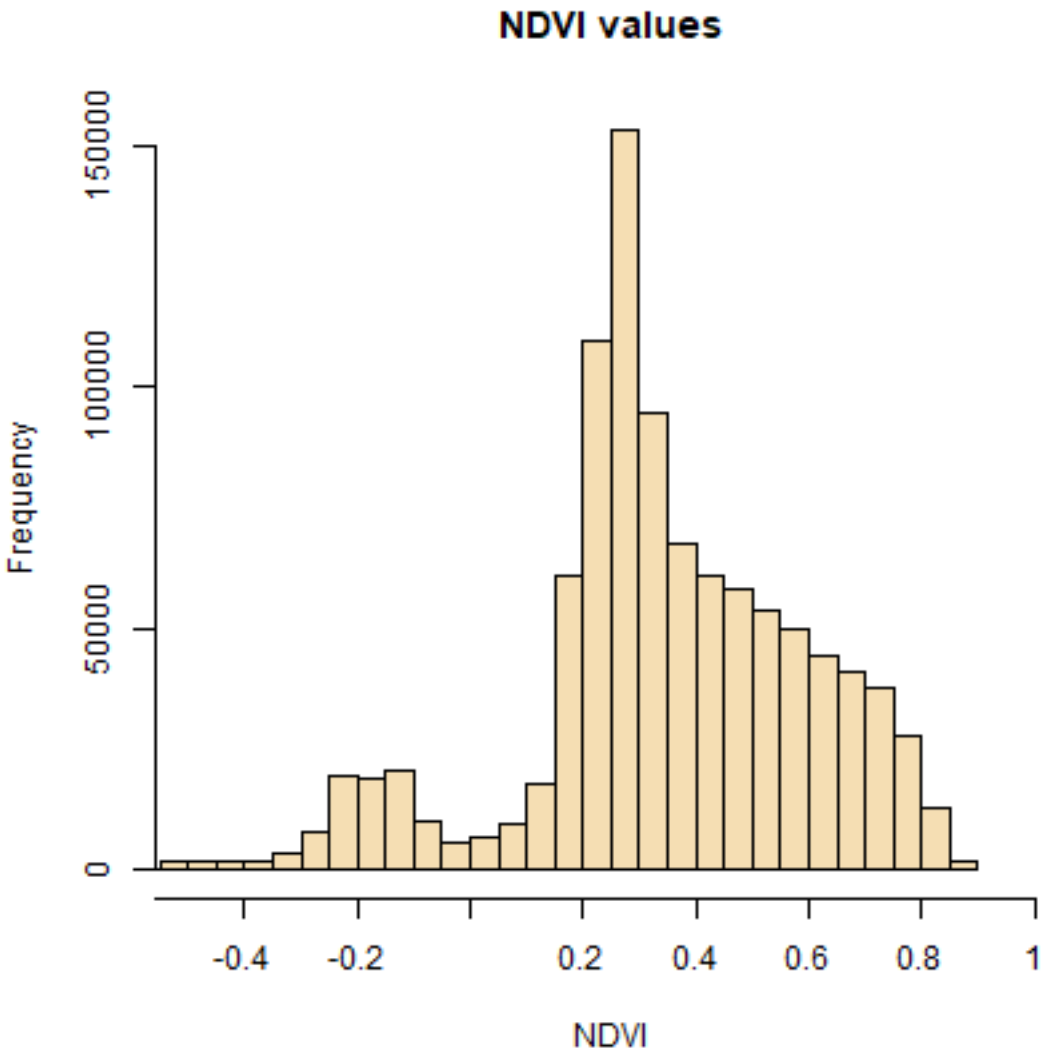


Question 1: *Adapt the code shown above to compute indices to identify i) water and ii) built-up. Hint: Use the spectral profile plot to find the bands having maximum and minimum reflectance for these two classes. Or read about it*here..

3.2 Histogram

We can explore the distribution of values contained within our raster using `hist` to produces a histogram. Histograms are often useful in identifying outliers and bad data values in our raster data.

```
hist(ndvi, main = "NDVI values", xlab = "NDVI", ylab= "Frequency",
    col = "wheat", xlim = c(-0.5, 1),  breaks = 30, xaxt = "n")
## Warning: [hist] a sample of54% of the cells was used

axis(side=1, at = seq(-0.6, 1, 0.2), labels = seq(-0.6, 1, 0.2))
```



We will refer to this histogram for the following sub-section on thresholding.

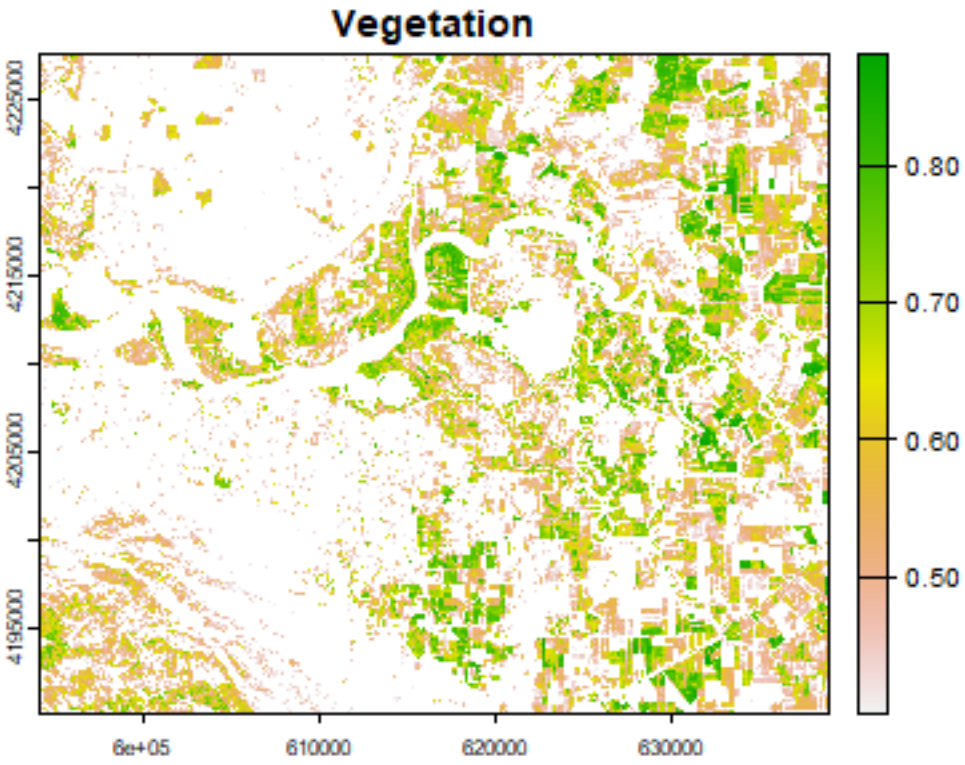
Question 2: *Make histograms of the values the vegetation indices developed in question 1.*

3.3 Thresholding

We can apply basic rules to get an estimate of spatial extent of different Earth surface features. Note that NDVI values are standardized and ranges between -1 to +1. Higher values indicate more green cover.

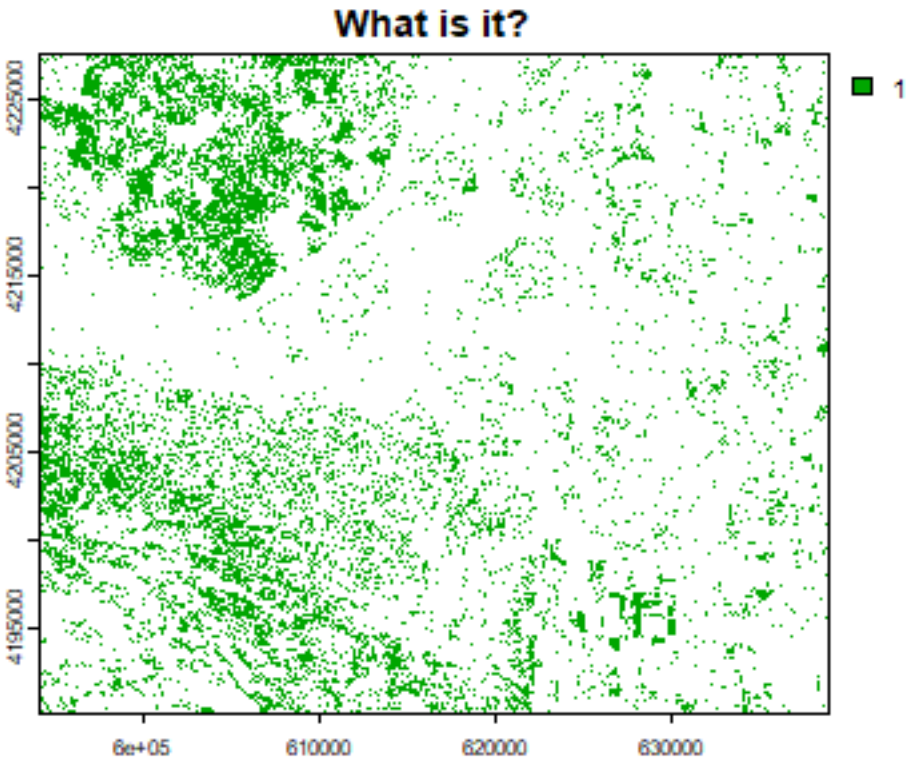
Cells with NDVI values greater than 0.4 are definitely vegetation. The following operation masks all cells that are perhaps not vegetation (NDVI < 0.4).

```
veg <- clamp(ndvi, 0.4, values=FALSE)
plot(veg, main='Vegetation')
```



Let's map the area that corresponds to the peak between 0.25 and 0.3 in the NDVI histogram.

```
m <- c(-Inf, 0.25, NA,  0.25, 0.3, 1,  0.3, Inf, NA)
rcl <- matrix(m, ncol = 3, byrow = TRUE)
land <- classify(ndvi, rcl)
plot(land, main = 'What is it?')
```

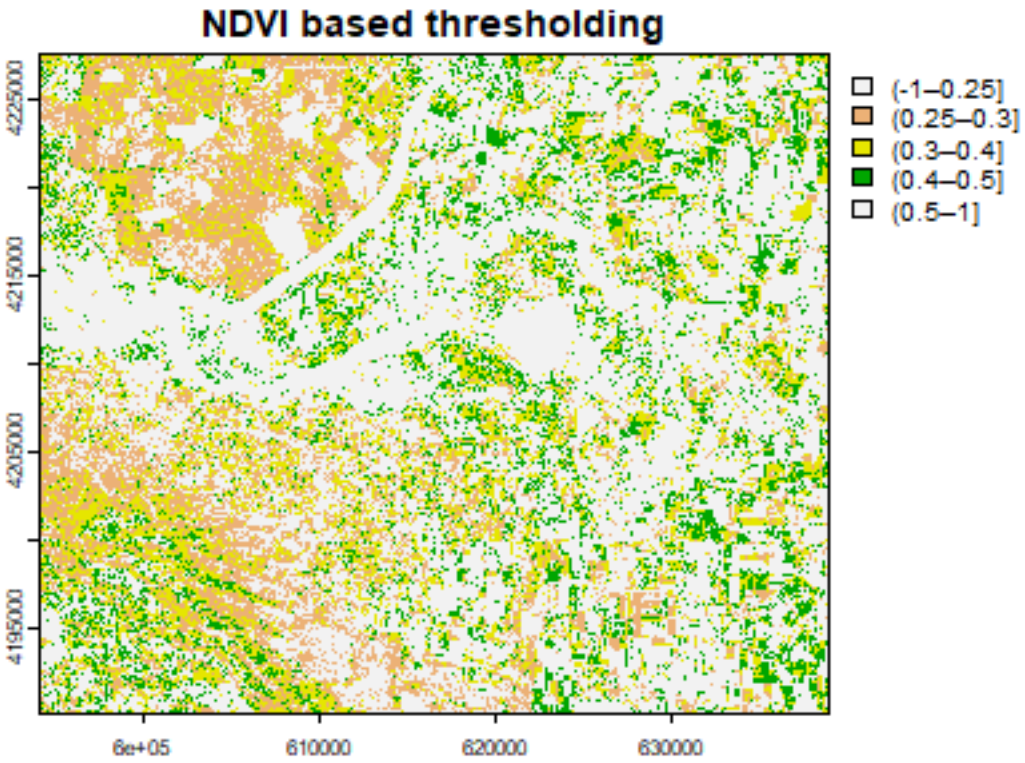


You can plot `land` on top of the `landsatRGB` raster to find out.

```
plotRGB(landsatRGB, r=1, g=2, b=3, axes=TRUE, stretch="lin")
plot(land, add=TRUE, legend=FALSE)
```

You can also create classes for different intensity of vegetation.

```
m <- c(-1,0.25, 0.3, 0.4, 0.5, 1)
vegc <- classify(ndvi, m)
plot(vegc, col = rev(terrain.colors(4)), main = 'NDVI based thresholding')
```



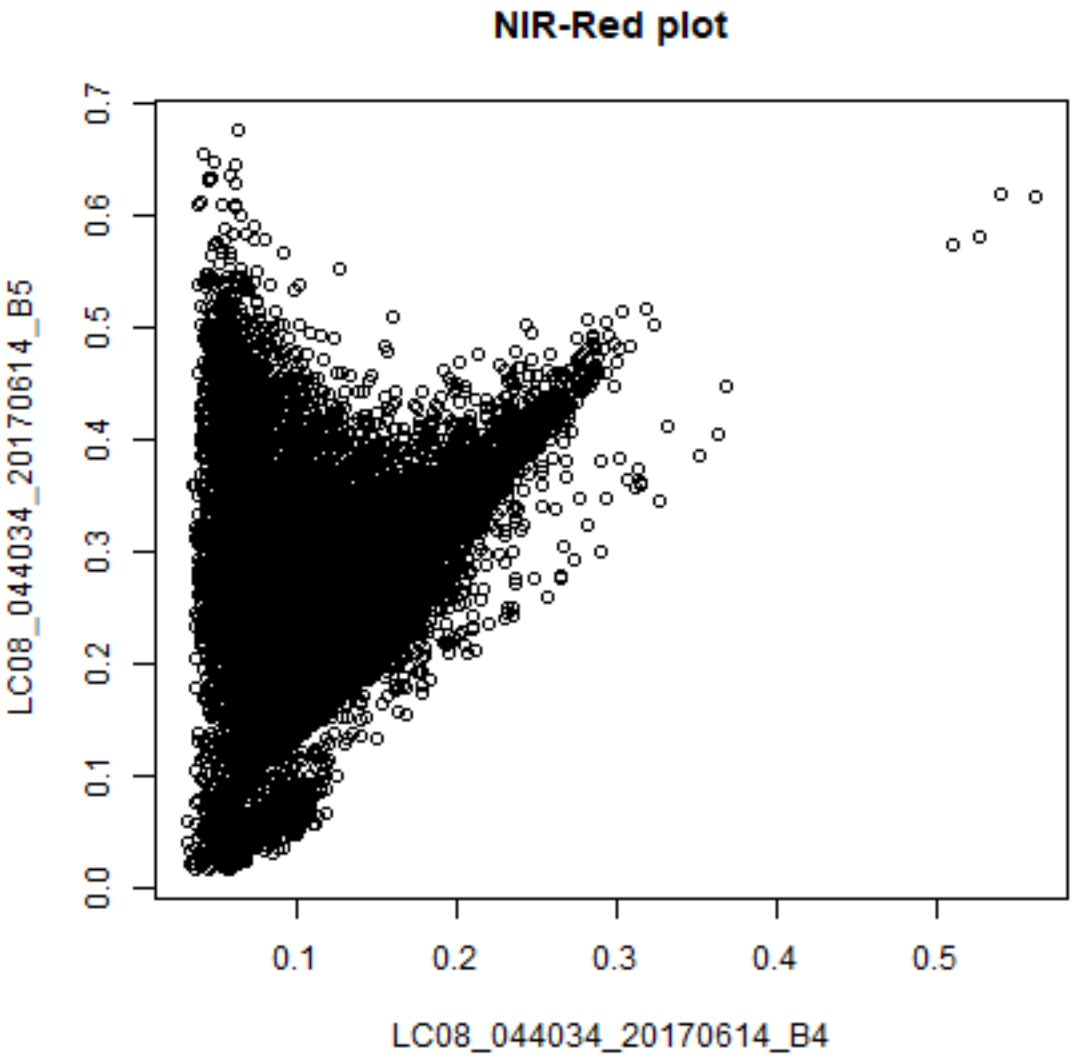
Question 3: *Is it possible to find water using thresholding of NDVI or any other indices?*

3.4 Principal component analysis

Multi-spectral data are sometimes transformed to helps to reduce the dimensionality and noise in the data. The principal components transform is a generic data reduction method that can be used to create a few uncorrelated bands from a larger set of correlated bands.

You can calculate the same number of principal components as the number of input bands. The first principal component (PC) explains the largest percentage of variance and other PCs explain additional the variance in decreasing order.

```
set.seed(1)
sr <- spatSample(landsat, 10000)
plot(sr[,c(4,5)], main = "NIR-Red plot")
```



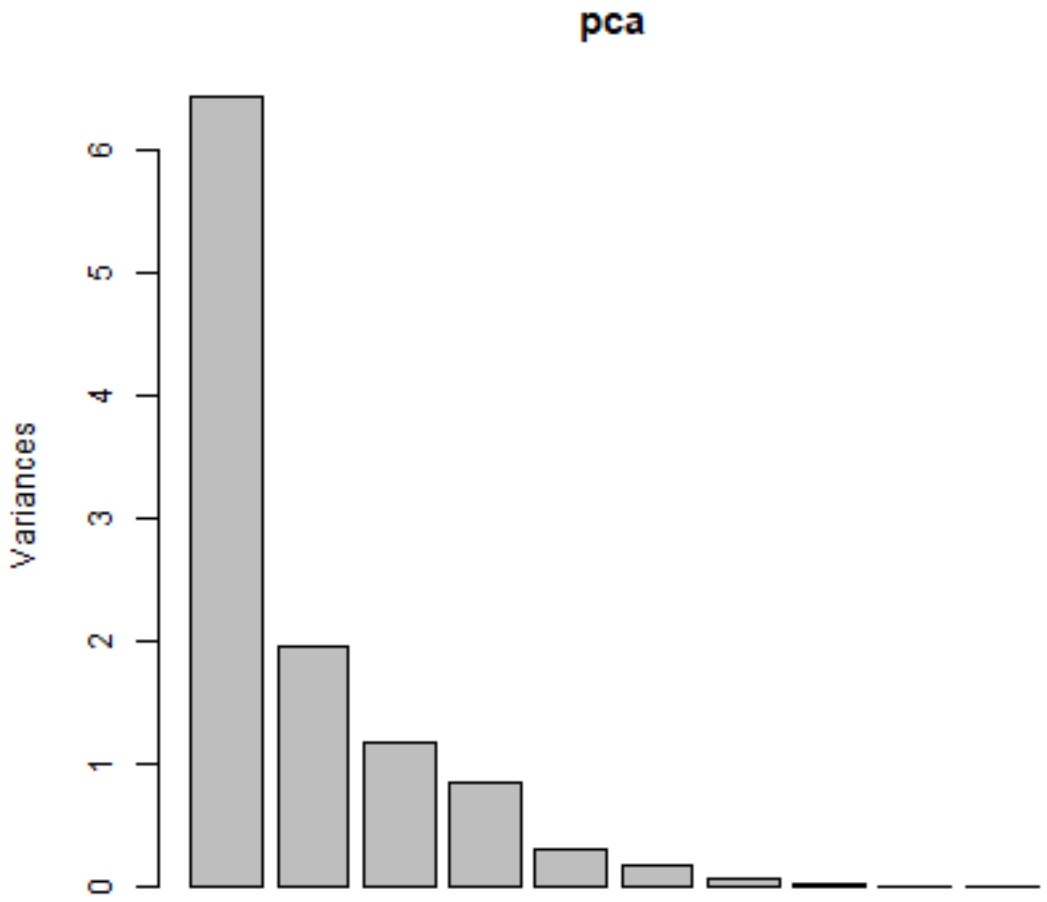
This is known as vegetation and soil-line plot (Same as the scatter plot in earlier section).

```
pca <- prcomp(sr, scale = TRUE)
pca
## Standard deviations (1, .., p=11):
##  [1] 2.53668771 1.40078059 1.08362915 0.92501695 0.54958227 0.41473655
##  [7] 0.27030406 0.12220817 0.08661844 0.04763013 0.03609876
##
## Rotation (n x k) = (11 x 11):
##                                PC1        PC2         PC3         PC4
## LC08_044034_20170614_B1  0.2937880  0.3642123 -0.28481332 -0.06676012
## LC08_044034_20170614_B2  0.3357803  0.3382035 -0.15857739 -0.03635911
## LC08_044034_20170614_B3  0.3612145  0.2650391  0.07275822  0.04141597
## LC08_044034_20170614_B4  0.3676650  0.1641879  0.10447910  0.03578747
## LC08_044034_20170614_B5  0.1591361 -0.1852694  0.71300561  0.32594984
## LC08_044034_20170614_B6  0.3472817 -0.2134275  0.22853576  0.15831890
## LC08_044034_20170614_B7  0.3499019 -0.2367876 -0.11725588  0.06147126
```

(continues on next page)

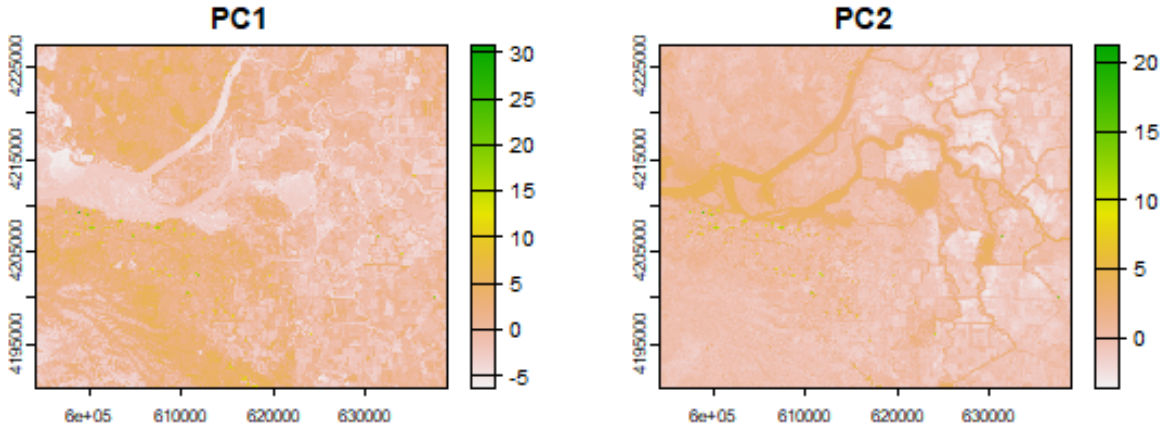
(continued from previous page)

```
## LC08_044034_20170614_B8  0.3496278  0.1964745  0.08489385  0.02433066
## LC08_044034_20170614_B9  0.1325628 -0.1117693  0.33293128 -0.92447725
## LC08_044034_20170614_B10 0.2534434 -0.4831768 -0.30276292 -0.02013211
## LC08_044034_20170614_B11 0.2507597 -0.4850418 -0.30570381 -0.02197088
##                                PC5         PC6         PC7          PC8
## LC08_044034_20170614_B1  -0.49633695 -0.17556592 -0.23610914  0.219084848
## LC08_044034_20170614_B2  -0.22593198 -0.09672106  0.06516109  0.191800572
## LC08_044034_20170614_B3  -0.06775249  0.01134076  0.29589321 -0.502516608
## LC08_044034_20170614_B4   0.33203379  0.06966965  0.60728435  0.005032259
## LC08_044034_20170614_B5  -0.51515049  0.06796061 -0.07734807 -0.094468368
## LC08_044034_20170614_B6   0.28840709 -0.33603872 -0.01251499  0.639691364
## LC08_044034_20170614_B7   0.24635064 -0.53401272 -0.39446741 -0.489769588
## LC08_044034_20170614_B8   0.33312920  0.65599341 -0.53808102  0.022845490
## LC08_044034_20170614_B9  -0.04429935 -0.04329372 -0.02425017 -0.001980250
## LC08_044034_20170614_B10 -0.16429483  0.24521079  0.13682816  0.057604000
## LC08_044034_20170614_B11 -0.19644322  0.24453814  0.11434312 -0.028609160
##                                 PC9          PC10         PC11
## LC08_044034_20170614_B1  -0.1023702943  0.5365796233  0.127065695
## LC08_044034_20170614_B2  -0.1218742739 -0.7691469170 -0.196283512
## LC08_044034_20170614_B3   0.6640135096  0.0551533999  0.059318851
## LC08_044034_20170614_B4  -0.5347992170  0.2350928333  0.021665820
## LC08_044034_20170614_B5  -0.1995974299 -0.0311387747  0.004029261
## LC08_044034_20170614_B6   0.3828069102  0.0639560393 -0.021372642
## LC08_044034_20170614_B7  -0.2438645765 -0.0551104634  0.011338146
## LC08_044034_20170614_B8   0.0005337710  0.0005066492 -0.001773935
## LC08_044034_20170614_B9  -0.0009025984 -0.0006276713  0.001823281
## LC08_044034_20170614_B10  0.0039273450 -0.1709760476  0.686896213
## LC08_044034_20170614_B11  0.0433142125  0.1576520508 -0.684766019
screeplot(pca)
```



We use a function to restrict prediction to the first two principal components

```
pca_predict2 <- function(model, data, ...) {
  predict(model, data, ...)[,1:2]
}
pci <- predict(landsat, pca, fun=pca_predict2)
plot(pci)
```

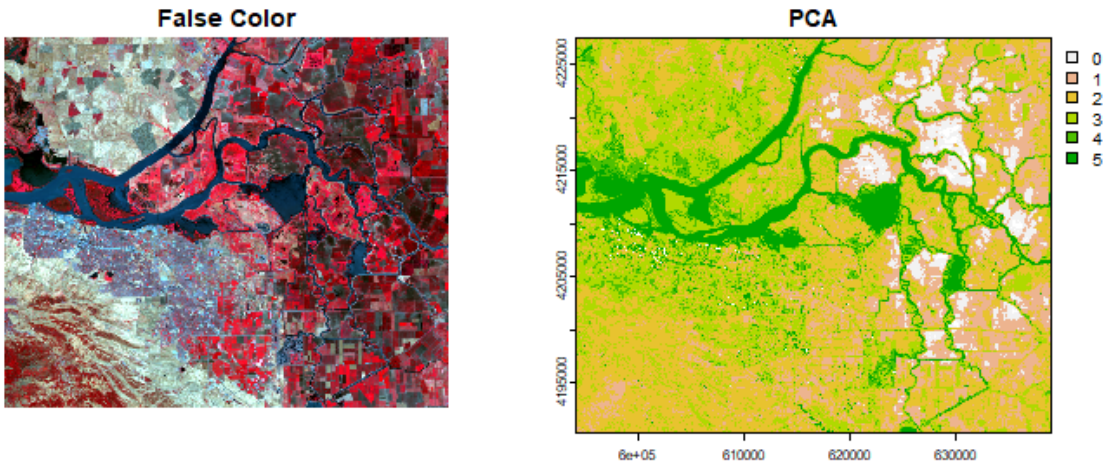


The first principal component highlights the boundaries between different land use classes. it is difficult to understand what the second principal component is highlighting. Lets try thresholding again:

```
# quick look at the histogram of second component
hist <- pci[[2]]
m <- c(-Inf,-3,NA, -3,-2,0, -2,-1,1, -1,0,2, 0,1,3, 1,2,4, 2,6,5, 6,Inf,NA)
rcl <- matrix(m, ncol = 3, byrow = TRUE)
rcl
##      [,1] [,2] [,3]
## [1,] -Inf   -3   NA
## [2,]   -3   -2    0
## [3,]   -2   -1    1
## [4,]   -1    0    2
## [5,]    0    1    3
## [6,]    1    2    4
## [7,]    2    6    5
## [8,]    6  Inf   NA
pcClass <- classify(pci[[2]], rcl)
```

Now plot the results

```
par(mfrow=c(1,2))
plotRGB(landsatFCC, stretch = "lin", main="False Color", mar=c(3.1, 3.1, 2.1, 2.1))
plot(pcClass, main="PCA")
```



To learn more about the information contained in the vegetation and soil line plots read this paper by Gitelson et al. Details about PCA and an extension of PCA in remote sensing, Tasseled-cap Transformation.

CHAPTER

FOUR

UNSUPERVISED CLASSIFICATION

In this chapter we explore unsupervised classification. Various unsupervised classification algorithms exist, and the choice of algorithm affects the results. Here we use the k-means algorithm to illustrate the general principle.

For this example, we will follow the National Land Cover Database 2011 (NLCD 2011) classification scheme for a subset of the Central Valley regions. We use cloud-free composite image from Landsat 5 with 6 bands.

```
library(terra)
## terra 1.7.62
landsat5 <- rast('data/rs/centralvalley-2011LT5.tif')
names(landsat5) <- c('blue', 'green', 'red', 'NIR', 'SWIR1', 'SWIR2')
```

Question 1: *Make a 3-band False Color Composite plot of ``landsat5``.*

In unsupervised classification, we use the reflectance data, but we don't supply any response data (that is, we do not identify any pixel as belonging to a particular class). This may seem odd, but it can be useful when we don't have much prior knowledge of a study area. Or if you have broad knowledge of the distribution of land cover classes of interest, but no specific ground data.

The algorithm groups pixels with similar spectral characteristics into groups.

Learn more about K-means and other unsupervised-supervised algorithms here.

We will perform unsupervised classification on a spatial subset of the `ndvi` layer. Here is yet another way to compute ndvi. In this case we do not use a separate function, but we use a direct algebraic notation.

```
ndvi <- (landsat5[['NIR']] - landsat5[['red']]) / (landsat5[['NIR']] + landsat5[['red']])
```

We will do `kmeans` clustering of the `ndvi` data. First we use `crop` to make a spatial subset of the `ndvi`, to allow for faster processing (you can select any `SpatExtent` using the `draw` function).

4.1 kmeans classification

```
# SpatExtent to crop ndvi layer
e <- ext(-121.807, -121.725, 38.004, 38.072)

# crop landsat by the extent
ndvi <- crop(ndvi, e)
ndvi
## class       : SpatRaster
## dimensions  : 252, 304, 1  (nrow, ncol, nlyr)
## resolution  : 0.0002694946, 0.0002694946  (x, y)
```

(continues on next page)

(continued from previous page)

```
## extent      : -121.807, -121.725, 38.00413, 38.07204  (xmin, xmax, ymin, ymax)
## coord. ref. : lon/lat WGS 84 (EPSG:4326)
## source(s)   : memory
## varname     : centralvalley-2011LT5
## name        :        NIR
## min value   : -0.3360085
## max value   :  0.7756007

# convert the raster to a data.frme
nr <- as.data.frame(ndvi, cell=TRUE)
str(nr)
## 'data.frame':    76608 obs. of  2 variables:
##  $ cell: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ NIR : num  0.245 0.236 0.272 0.277 0.277 ...
```

Please note that `values(ndvi)` converted the `ndvi` SpatRaster to an array (matrix). Now we will perform the `kmeans` clustering on the matrix and inspect the output.

```
# It is important to set the seed generator because `kmeans` initiates the centers in␣
↪random locations
set.seed(99)

# Create 10 clusters, allow 500 iterations, start with 5 random sets using "Lloyd"␣
↪method.
# Do not use the first column (cell number).
kmncluster <- kmeans(nr[,-1], centers=10, iter.max = 500, nstart = 5, algorithm="Lloyd")

# kmeans returns an object of class "kmeans"
str(kmncluster)
## List of 9
##  $ cluster     : int [1:76608] 4 4 3 3 3 3 3 4 4 4 ...
##  $ centers     : num [1:10, 1] 0.55425 0.00498 0.29997 0.20892 -0.20902 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:10] "1" "2" "3" "4" ...
##   .. ..$ : NULL
##  $ totss       : num 6459
##  $ withinss    : num [1:10] 5.69 6.13 4.91 4.9 5.75 ...
##  $ tot.withinss: num 55.8
##  $ betweenss   : num 6403
##  $ size        : int [1:10] 8932 4550 7156 6807 11672 8624 8736 5040 9893 5198
##  $ iter        : int 108
##  $ ifault      : NULL
##  - attr(*, "class")= chr "kmeans"
```

`kmeans` returns an object with 9 elements. The length of the `cluster` element within `kmncluster` is 76608 which same as length of `nr` created from the `ndvi`. The cell values of `kmncluster$cluster` range between 1 to 10 corresponding to the input number of cluster we provided in the `kmeans` function. `kmncluster$cluster` indicates the cluster label for corresponding pixel. We need to convert the `kmncluster$cluster` values back to a `SpatRaster` of the same dimension as the `ndvi`.

```
# Use the ndvi object to set the cluster values to a new raster
knr <- rast(ndvi, nlyr=1)
```

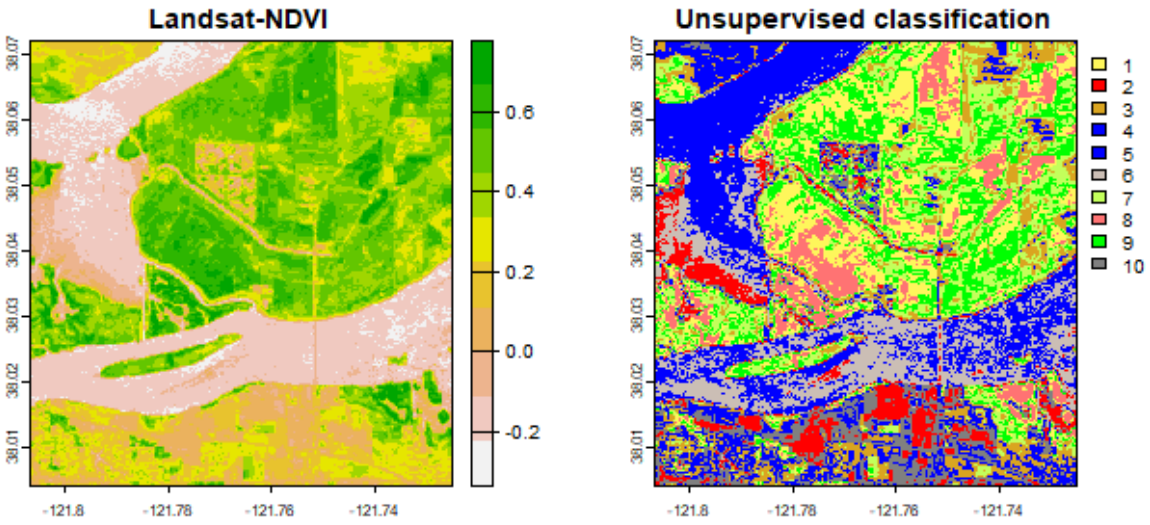
(continues on next page)

(continued from previous page)

```
knr[nr$cell] <- kmncluster$cluster
knr
## class       : SpatRaster
## dimensions  : 252, 304, 1  (nrow, ncol, nlyr)
## resolution  : 0.0002694946, 0.0002694946  (x, y)
## extent      : -121.807, -121.725, 38.00413, 38.07204  (xmin, xmax, ymin, ymax)
## coord. ref. : lon/lat WGS 84 (EPSG:4326)
## source(s)   : memory
## varname     : centralvalley-2011LT5
## name        : NIR
## min value   :   1
## max value   :  10
```

We can see that knr is a SpatRaster but we do not know which cluster (1-10) belongs to what land cover class (and if it does belong to a class that we would recognize). You can find that out by plotting them side-by-side with a reference layers and using unique color for each cluster.

```
# Define a color vector for 10 clusters (learn more about setting the color later)
mycolor <- c("#fef65b","#ff0000", "#daa520","#0000ff","#0000ff","#00ff00","#cbbeb5",
             "#c3ff5b", "#ff7373", "#00ff00", "#808080")

par(mfrow = c(1,2))
plot(ndvi, col = rev(terrain.colors(10)), main = "Landsat-NDVI")
plot(knr, main = 'Unsupervised classification', col = mycolor, type="classes")
```



While for other purposes it is usually better to define more classes (and possibly merge classes later), a simple classification like this one could be useful, e.g., merge cluster 4 and 5 to construct a water mask for the year 2011.

You can change the colors in my mycolor. Learn more about selecting colors in R here and here.

Question 2:*Plot a true-color image of 'landsat5' for the subset (extent 'e') and result of 'kmeans' clustering side-by-side and make a table of land-use land-cover labels for the clusters (based on visual inspection). E.g. cluster 4 and 5 are water.*

CHAPTER

FIVE

SUPERVISED CLASSIFICATION

Here we explore supervised classification for a simple land use land cover (LULC) mapping task. Various supervised classification algorithms exist, and the choice of algorithm can affect the results. Here we explore two related algorithms (CART and RandomForest).

In supervised classification, we have prior knowledge about some of the land-cover types through, for example, field-work, reference spatial data or interpretation of high resolution imagery (such as available on Google maps). Specific sites in the study area that represent homogeneous examples of these known land-cover types are identified. These areas are commonly referred to as training sites because the spectral properties of these sites are used to train the classification algorithm.

The following examples uses a Classification and Regression Trees (CART) classifier (Breiman et al. 1984) (further reading to predict land use land cover classes in the study area.

We will take the following steps:

- Create sample sites used for classification
- Extract cell values from Landsat data for the sample sites
- Train the classifier using training samples
- Classify the Landsat data using the trained model
- Evaluate the accuracy of the model

5.1 Landsat data to classify

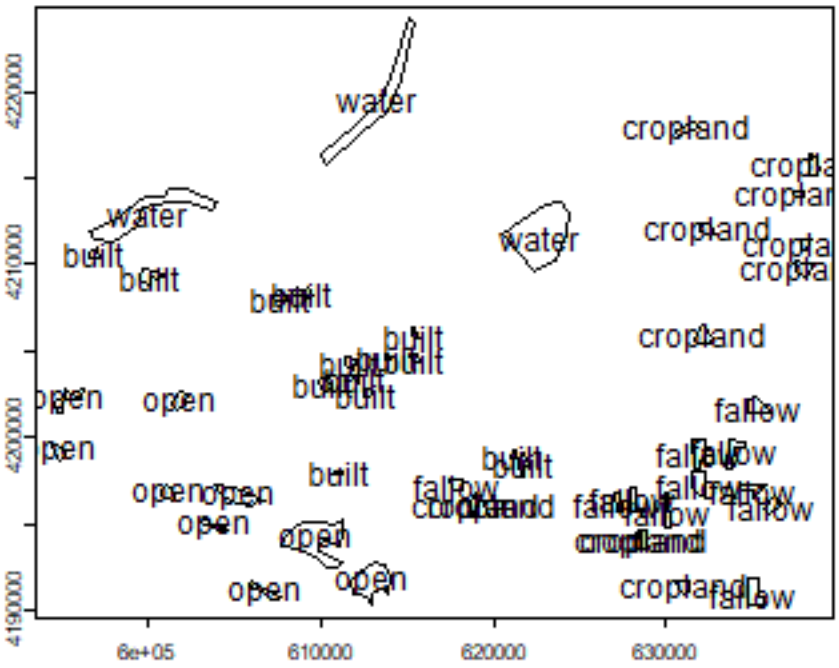
Here is our Landsat data.

```
library(terra)
## terra 1.7.62

# We read the 6 bands from the Landsat image we previously used
raslist <- paste0('data/rs/LC08_044034_20170614_B', 2:7, ".tif")
landsat <- rast(raslist)
names(landsat) <- c('blue', 'green', 'red', 'NIR', 'SWIR1', 'SWIR2')
```

5.2 Reference data

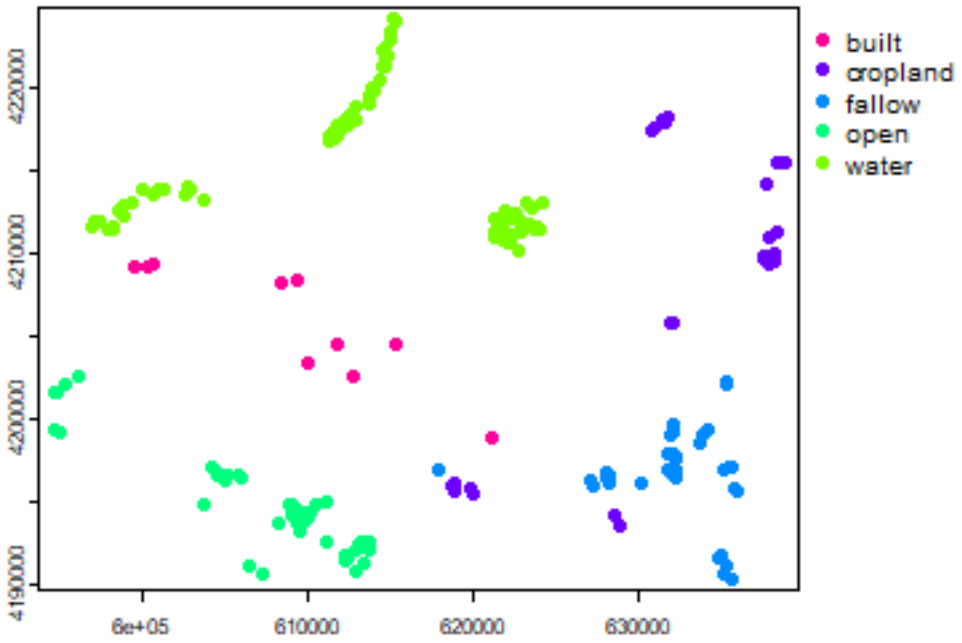
Training and/or validation data can come from a variety of sources. In this example, we use some training polygons we have already collected from other sources. We have already used this for making the spectral plots.There are 5 distinct classes – built,cropland,fallow,open and, water and we hope to find the pixels under this categroies based on our knowledge of training sites.

```
# load polygons with land use land cover information
samp <- readRDS("data/rs/lcsamples.rds")

# check the distribution of the polygons
plot(samp)
text(samp, samp$class)
```



Next we generate random points within each polygons.

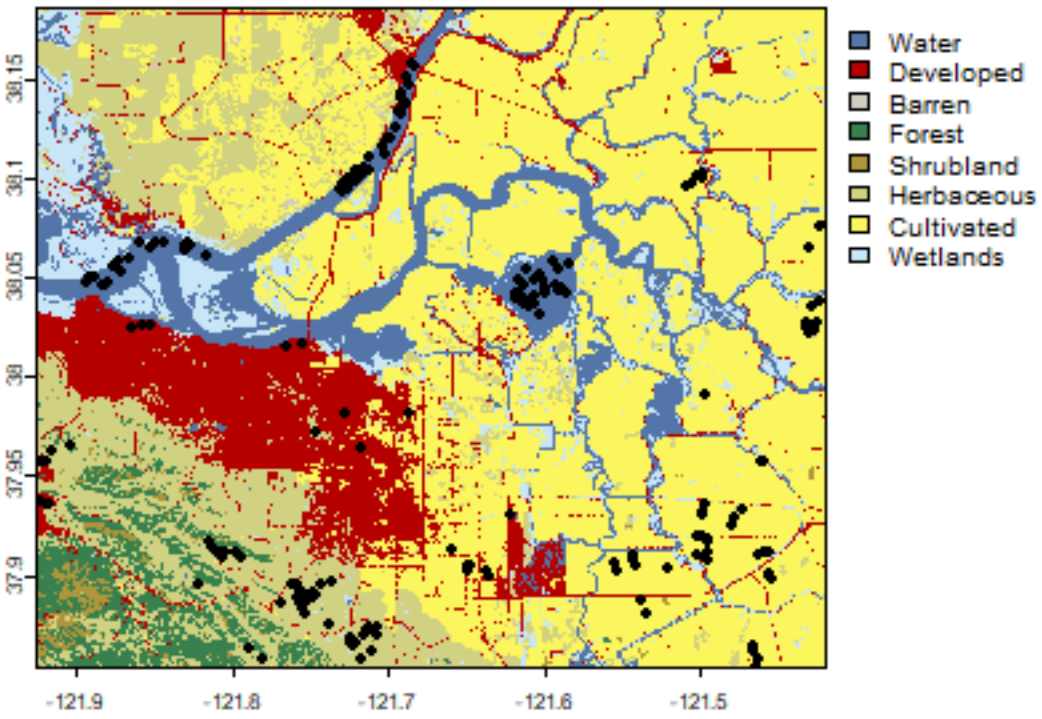
```
set.seed(1)
# generate point samples from the polygons
ptsamp <- spatSample(samp, 200, method="random")
plot(ptsamp, "class")
```



Alternatively, we can generate the training and validation sample sites using a reference land use land cover data. For example, the National Land Cover Database 2011 (NLCD 2011) is a land cover product for the United States. NLCD is a 30-m Landsat-based land cover database spanning 4 epochs (1992, 2001, 2006 and 2011). NLCD 2011 is based primarily on a decision-tree classification of circa 2011 Landsat data.

Detailes of the class mapped in NCLD 2011 can be found here (here)[https://www.mrlc.gov/nlcd11_leg.php]. It has two pairs of class values and names that correspond to the levels of land use and land cover classification system. These levels usually represent the level of complexity, level I being the simplest with broad land use land cover categories. Read this report by Anderson et al to learn more about this land use and land cover classification system.

```
nlcd <- rast('data/rs/nlcd-L1.tif')
names(nlcd) <- c("nlcd2001", "nlcd2011")
nlcd2011 <- nlcd[[2]]

# assign class names as categories (levels)
nlcdclass <- c("Water", "Developed", "Barren", "Forest", "Shrubland", "Herbaceous",
↪"Cultivated", "Wetlands")
classdf <- data.frame(value = c(1,2,3,4,5,7,8,9), names = nlcdclass)
levels(nlcd2011) <- classdf

# colors (as hexidecimal codes)
classcolor <- c("#5475A8", "#B50000", "#D2CDC0", "#38814E", "#AF963C", "#D1D182", "
↪#FBF65D", "#C8E6F8")

# plot the locations on top of the original nlcd raster
plot(nlcd2011, col=classcolor)
ptlonlat <- project(ptsamp, crs(nlcd2011))
points(ptlonlat)
```



Generate sample sites from the NLCD SpatRaster

```
# Sampling
samp2011 <- spatSample(nlcd2011, size = 200, method="regular")

# Number of samples in each class
table(samp2011[,1])
##
##      Water  Developed     Barren     Forest  Shrubland Herbaceous Cultivated
##         19         29          1          6          3         35        107
##   Wetlands
##         16
```

5.3 Extract reflectance values for the training sites

Once we have the training sites, we can extract the cell values from each layer in `landsat`. These values will be the predictor variables and "class" from `ptsamp` will be the response variable.

```
# extract the reflectance values for the locations
df <- extract(landsat, ptsamp, ID=FALSE)

# Quick check for the extracted values
head(df)
##         blue      green        red        NIR       SWIR1       SWIR2
## 1 0.09797934 0.08290726 0.05811966 0.01958285 0.006245691 0.003816809
```

(continues on next page)

(continued from previous page)

```
## 2 0.11986095 0.09904197 0.07978442 0.05723051 0.040965684 0.033939276
## 3 0.12838373 0.13727517 0.18810819 0.31336907 0.315103978 0.180648044
## 4 0.15295446 0.18123358 0.25232175 0.40375817 0.401546150 0.237791821
## 5 0.13992092 0.14987499 0.19207680 0.28279120 0.356763631 0.232044920
## 6 0.12341753 0.10114556 0.08288557 0.06377982 0.046972826 0.038883783

# combine lulc class information with extracted values
sampdata <- data.frame(class = ptsamp$class, df)
```

We often find classnames are provided as string labels (e.g. water, crop, vegetation) that need to be 'relabelled' to integer or factors if only string labels are supplied before using them as response variable in the classification. There are several approaches that could be used to convert these classes to integer codes. We can make a function that will reclassify the character strings representing land cover classes into integers based on the existing factor levels.

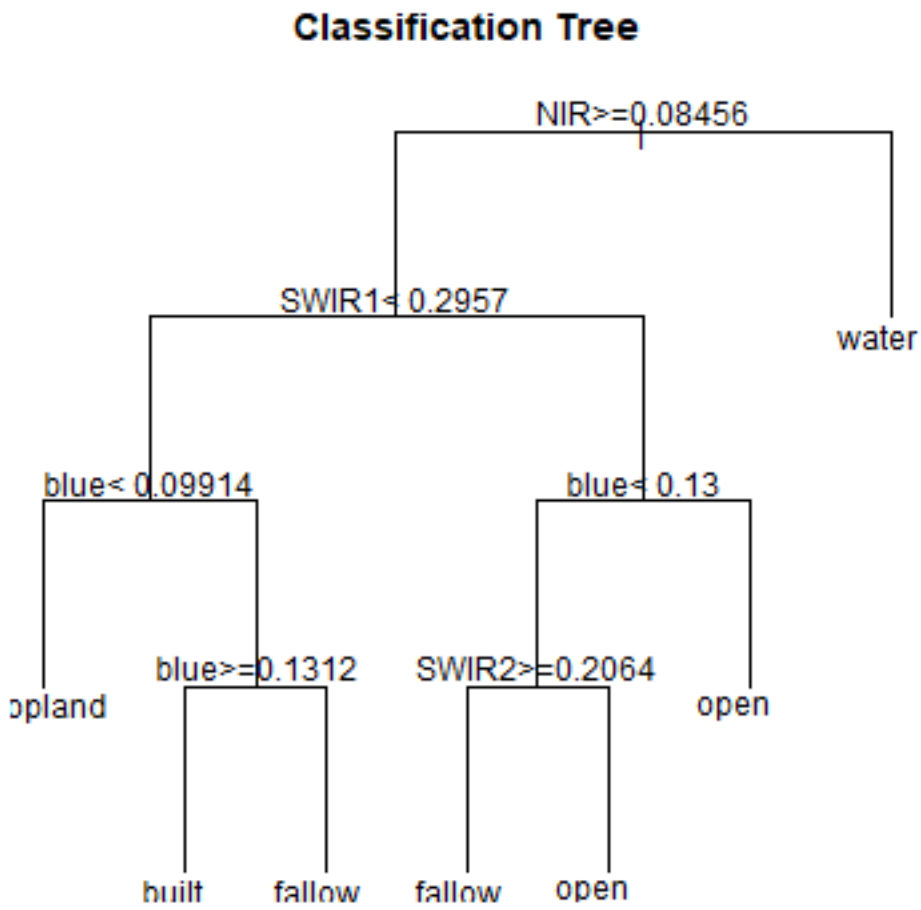
5.4 Train the classifier

Now we will train the classification algorithm using `sampdata` dataset.

```
library(rpart)

# Train the model
cartmodel <- rpart(as.factor(class)~., data = sampdata, method = 'class', minsplit = 5)
```

One of the primary reasons behind choosing `cart` model is to have a closer look at the classification model. Unlike other models, `cart` provides very simple way of inspecting and plotting the model structure.

```
# print trained model
print(cartmodel)
## n= 200
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 200 122 water (0.055 0.12 0.19 0.24 0.39)
##    2) NIR>=0.08455543 122  73 open (0.09 0.2 0.31 0.4 0)
##      4) SWIR1< 0.2957055 70  35 fallow (0.14 0.34 0.5 0.014 0)
##        8) blue< 0.09913956 24   0 cropland (0 1 0 0 0) *
##        9) blue>=0.09913956 46  11 fallow (0.22 0 0.76 0.022 0)
##         18) blue>=0.1312463 10   0 built (1 0 0 0 0) *
##         19) blue< 0.1312463 36   1 fallow (0 0 0.97 0.028 0) *
##      5) SWIR1>=0.2957055 52   4 open (0.019 0 0.058 0.92 0)
##       10) blue< 0.1299668 13   3 open (0 0 0.23 0.77 0)
##         20) SWIR2>=0.2063899 3   0 fallow (0 0 1 0 0) *
##         21) SWIR2< 0.2063899 10   0 open (0 0 0 1 0) *
##       11) blue>=0.1299668 39   1 open (0.026 0 0 0.97 0) *
##    3) NIR< 0.08455543 78   0 water (0 0 0 0 1) *

# Plot the trained classification tree
plot(cartmodel, uniform=TRUE, main="Classification Tree")
text(cartmodel, cex = 1)
```



See `?rpart.control` to set different parameters for building the model.

You can print/plot more about the `cartmodel` created in the previous example. E.g. you can use `plotcp(cartmodel)` to learn about the cost-complexity (`cp` argument in `rpart`).

5.5 Classify

Now that we have our trained classification model (`cartmodel`), we can use it to make predictions, that is, to classify all cells in the `landsat5` SpatRaster.

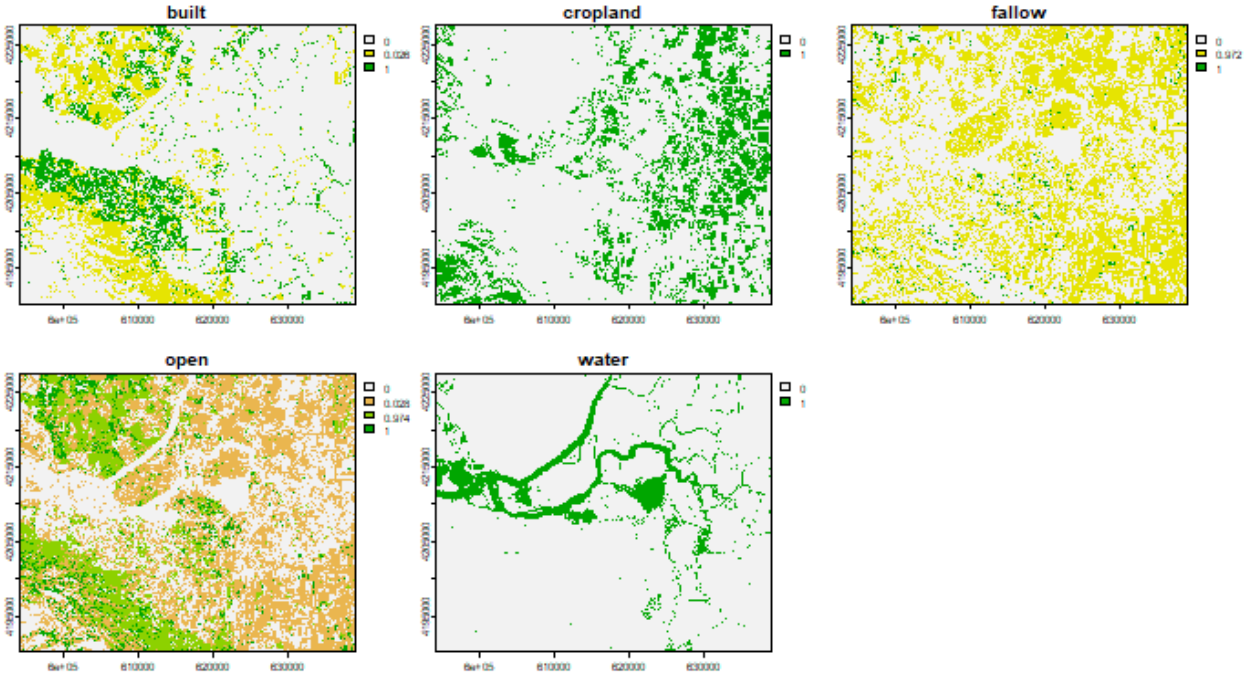
Important The layer names in the SpatRaster should exactly match those that were used to train the model. This will be the case if the same SpatRaster object was used (via extract) to obtain the values to fit the model. Otherwise you need to specify the matching names.

```
classified <- predict(landsat, cartmodel, na.rm = TRUE)
classified
```

(continues on next page)

(continued from previous page)

```
## class       : SpatRaster
## dimensions  : 1245, 1497, 5  (nrow, ncol, nlyr)
## resolution  : 30, 30  (x, y)
## extent      : 594090, 639000, 4190190, 4227540  (xmin, xmax, ymin, ymax)
## coord. ref. : WGS 84 / UTM zone 10N (EPSG:32610)
## source(s)   : memory
## names       : built, cropland, fallow, open, water
## min values  :     0,        0,      0,    0,     0
## max values  :     1,        1,      1,    1,     1
plot(classified)
```



Observe that there are 5 layers in the `classified` object, each of the layer representing the probability of a particular LULC class. Below, we make a SpatRaster that shows, for each grid cell, the LULC class with the highest probability.

```
lulc <- which.max(classified)
lulc
## class       : SpatRaster
## dimensions  : 1245, 1497, 1  (nrow, ncol, nlyr)
## resolution  : 30, 30  (x, y)
## extent      : 594090, 639000, 4190190, 4227540  (xmin, xmax, ymin, ymax)
## coord. ref. : WGS 84 / UTM zone 10N (EPSG:32610)
## source(s)   : memory
## name        : which.max
## min value   :         1
## max value   :         5
```

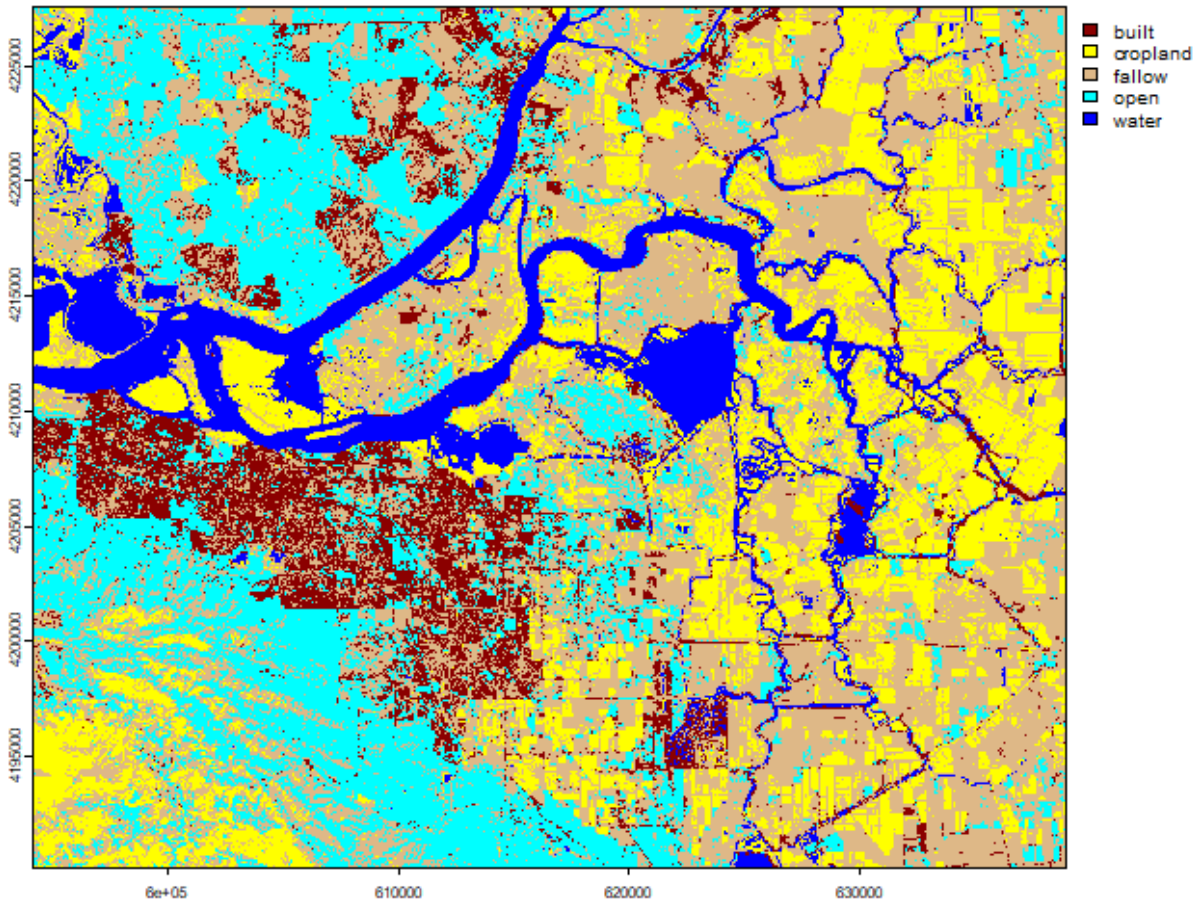
To make a nice map, we make the raster categorical, using `levels<-`; and we provide custom colors.

```
cls <- c("built","cropland","fallow","open","water")
df <- data.frame(id = 1:5, class=cls)
```

(continues on next page)

(continued from previous page)

```
levels(lulc) <- df
lulc
## class       : SpatRaster
## dimensions  : 1245, 1497, 1  (nrow, ncol, nlyr)
## resolution  : 30, 30  (x, y)
## extent      : 594090, 639000, 4190190, 4227540  (xmin, xmax, ymin, ymax)
## coord. ref. : WGS 84 / UTM zone 10N (EPSG:32610)
## source(s)   : memory
## categories  : class
## name        : class
## min value   : built
## max value   : water

mycolor <- c("darkred", "yellow", "burlywood", "cyan", "blue")
plot(lulc, col=mycolor)
```



If you are not satisfied with the results, you can select more samples and use additional predictor variables to see if you can improve the classification. The choice of classifier (algorithm) also plays an important role. Next we show how to test the performance the classification model.

5.6 Model evaluation

This section discusses how to assess the accuracy of the model to get an idea of how accurate the classified map might be. Two widely used measures in remote sensing are "overall accuracy" and "kappa". You can perform the accuracy assessment using the independent samples.

To evaluate any model, you can use k-fold cross-validation (you can also do single-fold). In this technique the data used to fit the model is split into `k` groups (typically 5 groups). In turn, one of the groups will be used for model testing, while the rest of the data is used for model training (fitting).

```
set.seed(99)

# number of folds
k <- 5
j <- sample(rep(1:k, each = round(nrow(sampdata))/k))
table(j)
## j
##  1  2  3  4  5
## 40 40 40 40 40
```

Now we train and test the model five times, each time computing the predictions and storing that with the actual values in a list. Later we use the list to compute the final accuarcy.

```
x <- list()

for (k in 1:5) {
    train <- sampdata[j!= k, ]
    test <- sampdata[j == k, ]
    cart <- rpart(as.factor(class)~., data=train, method = 'class',
                  minsplit = 5)
    pclass <- predict(cart, test, na.rm = TRUE)
    # assign class to maximum probablity
    pc <- apply(pclass, 1, which.max)
    # use labels instead of numbers
    pc <- colnames(pclass)[pc]
    # create a data.frame using the reference and prediction
    x[[k]] <- cbind(test$class, pc)
}
```

Now combine the five list elements into a single data.frame, using `do.call` and compute a confusion matrix.

```
y <- do.call(rbind, x)
y <- data.frame(y)
colnames(y) <- c('observed', 'predicted')

# confusion matrix
conmat <- table(y)
print(conmat)
##           predicted
## observed   built cropland fallow open water
##   built        8        0      1    2     0
##   cropland     0       24      0    0     0
##   fallow       2        0     33    3     0
```

(continues on next page)

(continued from previous page)

```
##   open          0        0      2   47     0
##   water         0        0      0    0    78
```

Question 1:*Comment on the miss-classification between different classes.*

Question 2:*Can you think of ways to to improve the accuracy.*

Compute the overall accuracy and the "kappa" statistic.

Overall accuracy:

```
# number of total cases/samples
n <- sum(conmat)
n
## [1] 200

# number of correctly classified cases per class
diag <- diag(conmat)

# Overall Accuracy
OA <- sum(diag) / n
OA
## [1] 0.95
```

Kappa:

```
# observed (true) cases per class
rowsums <- apply(conmat, 1, sum)
p <- rowsums / n

# predicted cases per class
colsums <- apply(conmat, 2, sum)
q <- colsums / n

expAccuracy <- sum(p*q)
kappa <- (OA - expAccuracy) / (1 - expAccuracy)
kappa
## [1] 0.9317732
```

Producer and user accuracy

```
# Producer accuracy
PA <- diag / colsums

# User accuracy
UA <- diag / rowsums

outAcc <- data.frame(producerAccuracy = PA, userAccuracy = UA)
outAcc
##          producerAccuracy userAccuracy
## built           0.8000000    0.7272727
## cropland        1.0000000    1.0000000
## fallow          0.9166667    0.8684211
```

(continues on next page)

(continued from previous page)

```
## open            0.9038462    0.9591837
## water           1.0000000    1.0000000
```

Question 3:*Perform the classification using Random Forest classifiers from the ``randomForest`` package*

Question 4:*Plot the results of rpart and Random Forest classifier side-by-side.*

Question 5 (optional):*Repeat the steps for other years using Random Forest.* For example you can use the cloud-free composite image `data/centralvalley-2001LE7.tif`. This data is collected by the Landsat 7 platform. You can use the National Land Cover Database 2001 (NLCD 2001) subset of the California Central Valley for generating training sites.

Question 6 (optional):*We have trained the classifiers using unequal samples for each class. Investigate the effect of sample size on classification. Repeat the steps with different subsets, e.g. a sample size of 100, 50, 25 per class, and compare the results. Use the same holdout samples for model evaluation.*