Remote Sensing Image Analysis with R

Aniruddah Ghosh and Robert J. Hijmans

May 20, 2021

CONTENTS

Aniruddha Ghosh and Robert J. Hijmans

CHAPTER

ONE

INTRODUCTION

This book provides a short introduction to satellite data analysis with R. Before reading this you should first learn the basics of the raster 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 remote sensing data.

It is possible to download some satellite data using R-packages. For example, you can use the MODIS or MODISTools 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 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).

The data are normally stored as raster data (referred to as "images"). Each separate image (for a place and time) is referred to as a s "scene". 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" (shorthand for "bandwidth"), and grid cells are referred to as "pixel".

1.2 Data

You can download all the data required for the examples used in this book here. Unzip the file contents and save the data to the R working directory of your computer.

You can also use the below script to download the data.

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

1.4 R packages

Here is a list of some R packages for analyzing remote sensing data

- RStoolbox
- landsat
- hsdar
- rasterVis for visualization

CHAPTER

TWO

EXPLORATION

In this chapter we describe how to access and 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 image 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 zipped file, which contains separate files for each band.

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 RasterLayer objects for single Landsat layers (bands)

```
library(raster)

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

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

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

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

Print the variables to check. E.g.

```
b2
## class      : RasterLayer
## dimensions : 1245, 1497, 1863765  (nrow, ncol, ncell)
## resolution : 30, 30  (x, y)
## extent     : 594090, 639000, 4190190, 4227540  (xmin, xmax, ymin, ymax)
```

(continues on next page)

(continued from previous page)

```
## crs        : +proj=utm +zone=10 +datum=WGS84 +units=m +no_defs
## source     : LC08_044034_20170614_B2.tif
## names      : LC08_044034_20170614_B2
## values     : 0.0748399, 0.7177562  (min, max)
```

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 Raster* object (this is the same for any raster data set).

```
# coordinate reference system (CRS)
crs(b2)
## CRS arguments:
##  +proj=utm +zone=10 +datum=WGS84 +units=m +no_defs

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

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

# Number of bands
nlayers(b2)
## [1] 1

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

You can create a RasterStack (an object with multiple layers) from the existing RasterLayer (single band) objects.

```
s <- stack(b5, b4, b3)
# Check the properties of the RasterStack
s
## class      : RasterStack
## dimensions : 1245, 1497, 1863765, 3  (nrow, ncol, ncell, nlayers)
## resolution : 30, 30  (x, y)
## extent     : 594090, 639000, 4190190, 4227540  (xmin, xmax, ymin, ymax)
## crs        : +proj=utm +zone=10 +datum=WGS84 +units=m +no_defs
## names      : LC08_044034_20170614_B5, LC08_044034_20170614_B4, LC08_044034_20170614_B3
## min values :            0.0008457669,            0.0208406653,            0.0425921641
## max values :               1.0124315,               0.7861769,               0.6924697
```

You can also create the RasterStack using the filenames.

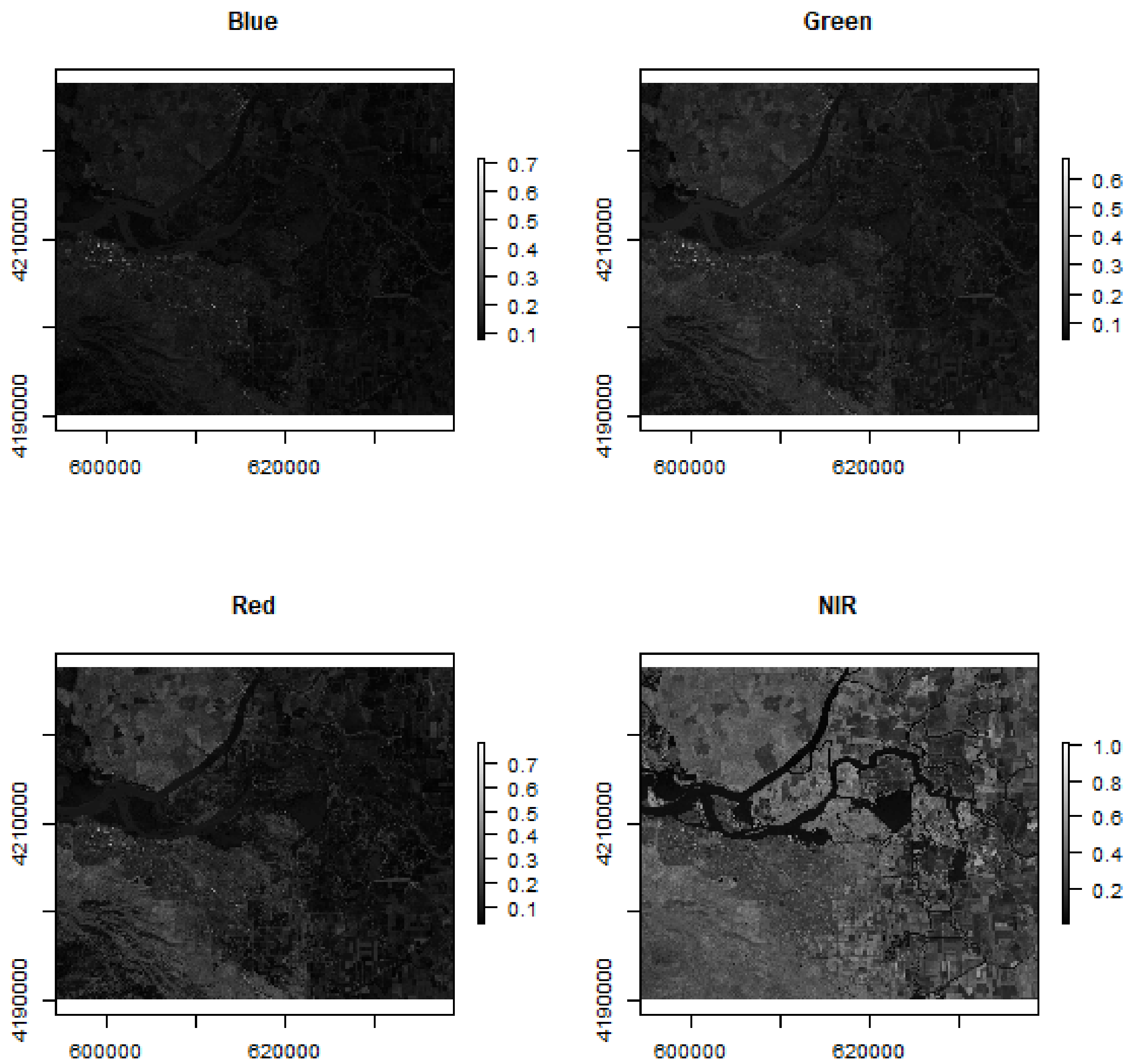
```
# first create a list of raster layers to use
filenames <- paste0('data/rs/LC08_044034_20170614_B', 1:11, ".tif")
filenames
##  [1] "data/rs/LC08_044034_20170614_B1.tif"
##  [2] "data/rs/LC08_044034_20170614_B2.tif"
##  [3] "data/rs/LC08_044034_20170614_B3.tif"
##  [4] "data/rs/LC08_044034_20170614_B4.tif"
##  [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 <- stack(filenames)
landsat
## class      : RasterStack
## dimensions : 1245, 1497, 1863765, 11  (nrow, ncol, ncell, nlayers)
## resolution : 30, 30  (x, y)
## extent     : 594090, 639000, 4190190, 4227540  (xmin, xmax, ymin, ymax)
## crs        : +proj=utm +zone=10 +datum=WGS84 +units=m +no_defs
## names      : LC08_044034_20170614_B1, LC08_044034_20170614_B2, LC08_044034_20170614_
↪B3, LC08_044034_20170614_B4, LC08_044034_20170614_B5, LC08_044034_20170614_B6, LC08_
↪044034_20170614_B7, LC08_044034_20170614_B8, LC08_044034_20170614_B9, LC08_044034_
↪20170614_B10, LC08_044034_20170614_B11
## min values :            9.641791e-02,            7.483990e-02,            4.259216e-
↪02,            2.084067e-02,            8.457669e-04,           -7.872183e-03,         ␣
↪  -5.052945e-03,            3.931751e-02,           -4.337332e-04,             2.
↪897978e+02,            2.885000e+02
## max values :              0.73462820,              0.71775615,              0.
↪69246972,              0.78617686,              1.01243150,              1.04320455, ␣
↪              1.11793602,              0.82673049,              0.03547901,         ␣
↪322.43139648,            317.99530029
```

Above we created a RasterStack 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. We won't use the last four layers and you will see how to remove those in following sections.

2.3 Single band and composite maps

You can plot individual layers of a RasterStack 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))
```

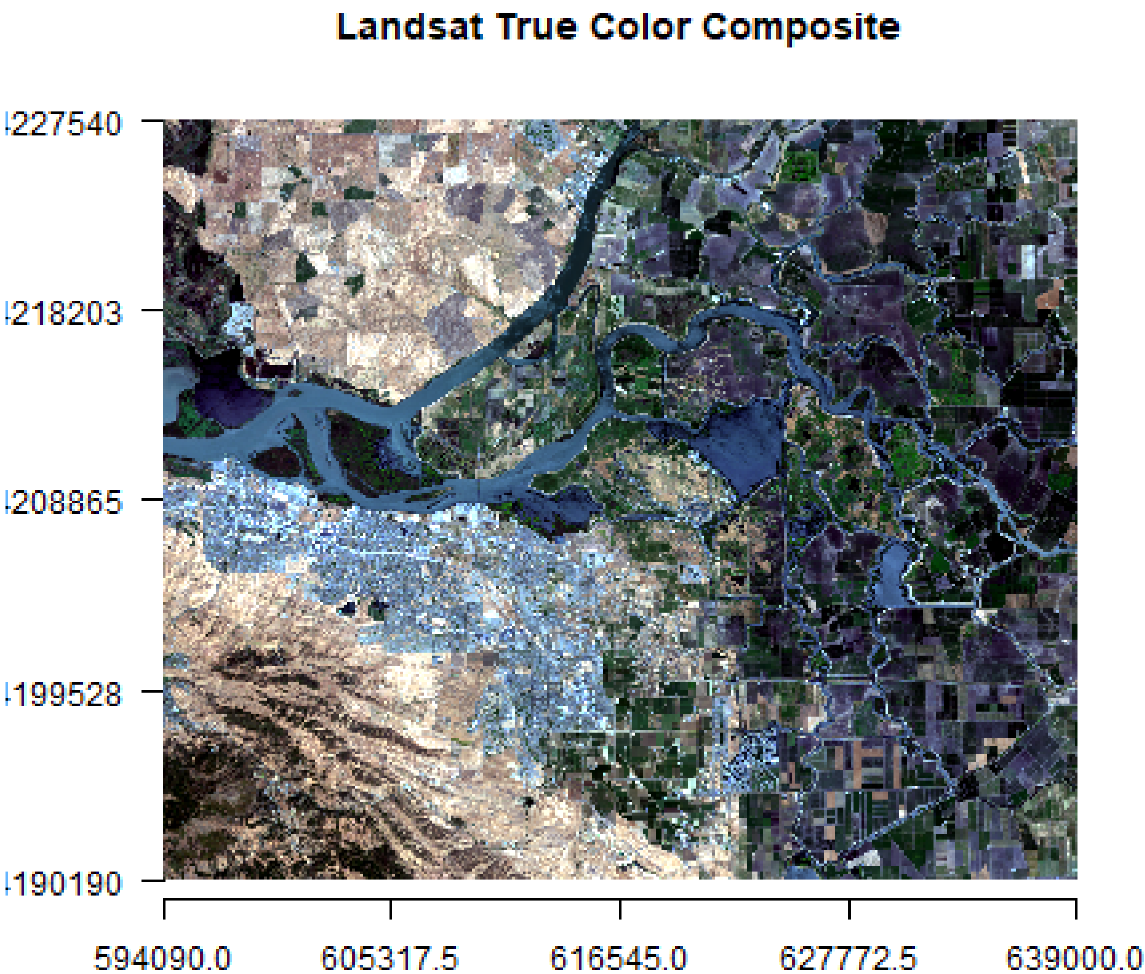


Have a look at the legends of the maps created above. They can range between 0 and 1. Notice the difference in shading and range of legends between the different bands. This is because different surface features reflect the incident solar radiation differently. Each layer represent how much incident 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.

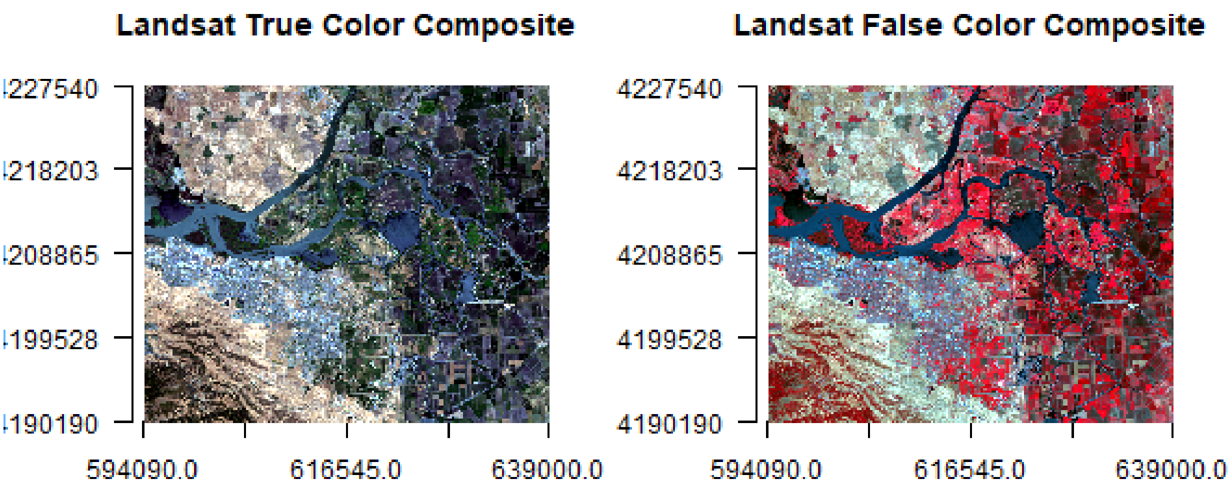
To make a "true (or natural) color" image, that is, something that looks like a normal photograph (vegetation in green, water blue etc), we need bands in the red, green and blue regions. For this Landsat image, band 4 (red), 3 (green), and 2 (blue) can be used. The `plotRGB` method can be used to combine them into a single composite. You can also supply additional arguments to `plotRGB` to improve the visualization (e.g. a linear stretch of the values, using `strecth = "lin"`).

```
landsatRGB <- stack(b4, b3, b2)
plotRGB(landsatRGB, axes = TRUE, stretch = "lin", main = "Landsat True Color Composite")
```



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).

```
par(mfrow = c(1,2))
plotRGB(landsatRGB, axes=TRUE, stretch="lin", main="Landsat True Color Composite")

landsatFCC <- stack(b5, b4, b3)
plotRGB(landsatFCC, axes=TRUE, stretch="lin", main="Landsat False Color Composite")
```



Note: Always check for package documentation (`help(plotRGB)`) for other arguments that can be added (like scale) to improve or modify the image.

Question 1: *Use the plotRGB function with RasterStack ``landsat`` to create a true and false color composite (hint remember the position of the bands in the stack).*

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
nlayers(landsat)
## [1] 11
nlayers(landsatsub1)
## [1] 3
nlayers(landsatsub2)
## [1] 3
```

We won't use the last four bands in `landsat`. You can remove those using

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

For clarity, it is useful to set the names of the bands.

```
names(landsat)
## [1] "LC08_044034_20170614_B1" "LC08_044034_20170614_B2"
## [3] "LC08_044034_20170614_B3" "LC08_044034_20170614_B4"
```

(continues on next page)

(continued from previous page)

```
## [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 an `extent` object, or another spatial object from which an Extent can be extracted.

```
# Using extent
extent(landsat)
## class      : Extent
## xmin       : 594090
## xmax       : 639000
## ymin       : 4190190
## ymax       : 4227540
e <- extent(624387, 635752, 4200047, 4210939)

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

Question 2: *Interactive selection from the image is also possible. Use ``drawExtent`` and ``drawPoly`` to select an area of interest*

Question 3: *Use the RasterStack ``landsatcrop`` to create a true and false color composite*

2.6 Saving results to disk

At this stage we may want to save the raster to disk using the function `writeRaster`. Multiple file types are supported. We will use the commonly used GeoTiff format. While the layer order is preserved, layer names are unfortunately lost in the GeoTiff format.

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

Alternatively you can used the 'raster-grd' format.

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

An advantage of this format is that it saves the layer names. The disadvantage of this format is that not many other programs can read the data, in contrast to the GeoTiff format.

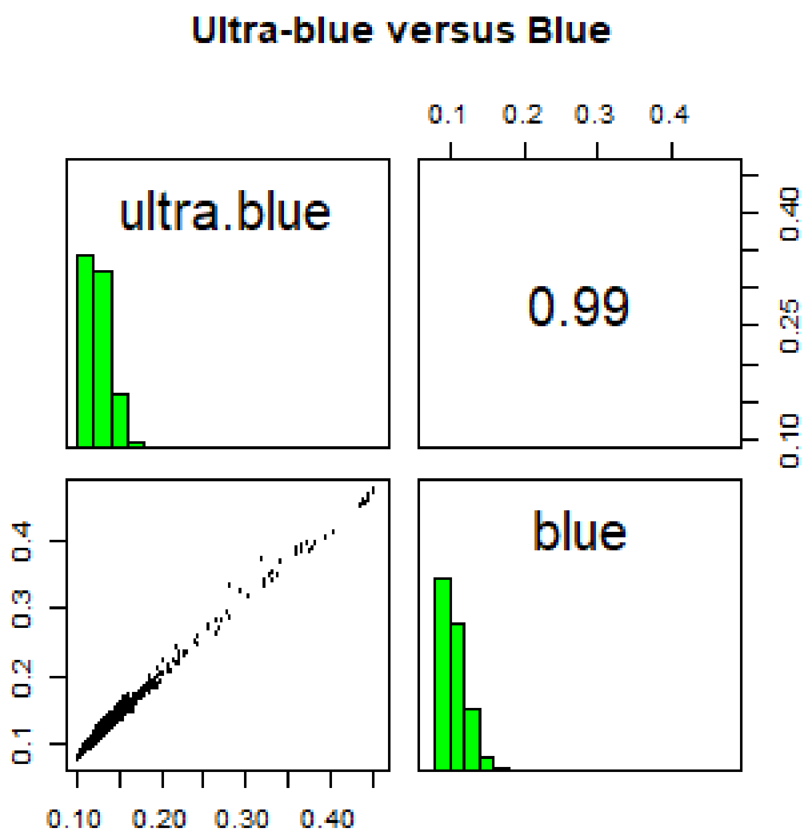
Note: Check for package documentation (`help(writeRaster)`) for additional helpful arguments that can be added.

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 of the raster package.

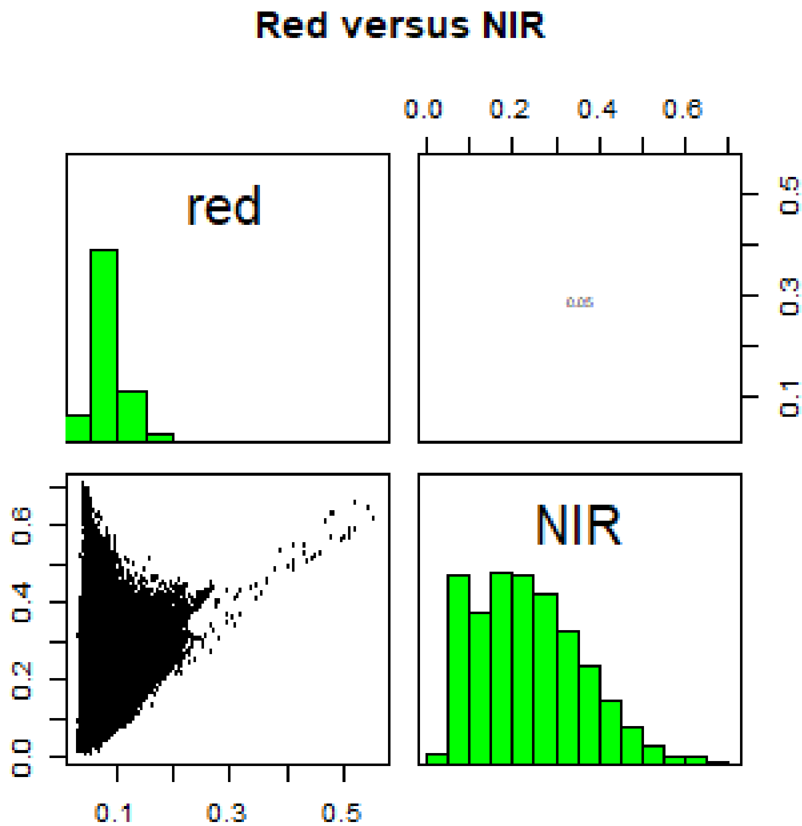
Plot of reflection in the ultra-blue wavelength against reflection in the blue wavelength.

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



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.

2.8 Extract pixel values

Often we want to get the values of raster cells 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 `Raster*` object for the cells in which a set of points fall.

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

# generate 300 point samples from the polygons
ptsamp <- spsample(samp, 300, type='regular')
## Warning in proj4string(obj): CRS object has comment, which is lost in output
```

(continues on next page)

(continued from previous page)

```
# add the land cover class to the points
ptsamp$class <- over(ptsamp, samp)$class

# extract values with points
df <- extract(landsat, ptsamp)

# To see some of the reflectance values
head(df)
##      ultra.blue      blue     green       red       NIR     SWIR1     SWIR2
## [1,]  0.1410052 0.1248705 0.1113816 0.1148080 0.1793469 0.2322618 0.1928575
## [2,]  0.1395523 0.1239814 0.1177140 0.1251308 0.1995803 0.2612348 0.2238691
## [3,]  0.1372535 0.1197959 0.1036178 0.1024034 0.1818842 0.2088187 0.1686338
## [4,]  0.1277982 0.1195357 0.1204682 0.1599375 0.3001404 0.3075571 0.1824047
## [5,]  0.1289693 0.1201212 0.1211838 0.1598074 0.2862176 0.3166437 0.1924021
## [6,]  0.1480100 0.1449739 0.1573568 0.2008598 0.3047379 0.3821368 0.2652685
```

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, 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     0.1791182 0.17209275 0.17608798 0.19163815 0.24237949 0.24805444
## cropland  0.1117013 0.08931219 0.08450462 0.05233991 0.48128346 0.15164212
## fallow    0.1346741 0.11946550 0.10820704 0.11711890 0.18396671 0.23850747
## open      0.1390056 0.13789844 0.15295334 0.20732350 0.34415621 0.35790739
## water     0.1333515 0.11632029 0.09899055 0.07830379 0.04851571 0.03296793
##               SWIR2
## built    0.20686696
## cropland 0.06555377
## fallow   0.19976847
## open     0.21372804
## water    0.02668551
```

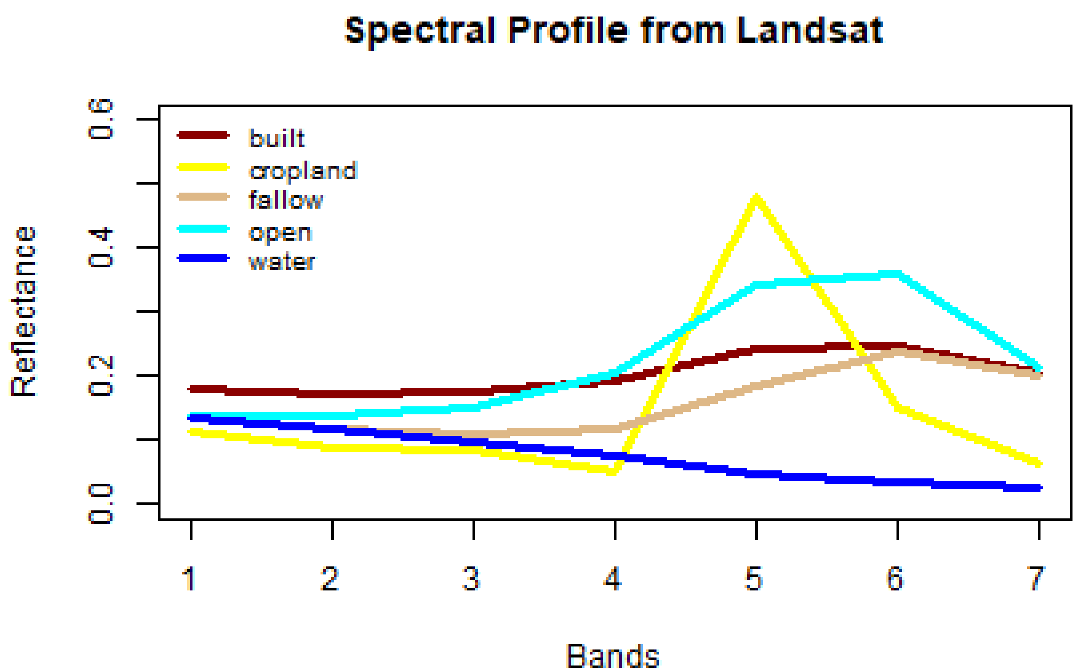
Now we plot the mean spectra of these features.

```
# 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
```

(continues on next page)

(continued from previous page)

```
ms <- as.matrix(ms)

# 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 Profile from Landsat", font.main = 2)

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



The spectral 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 raster 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.

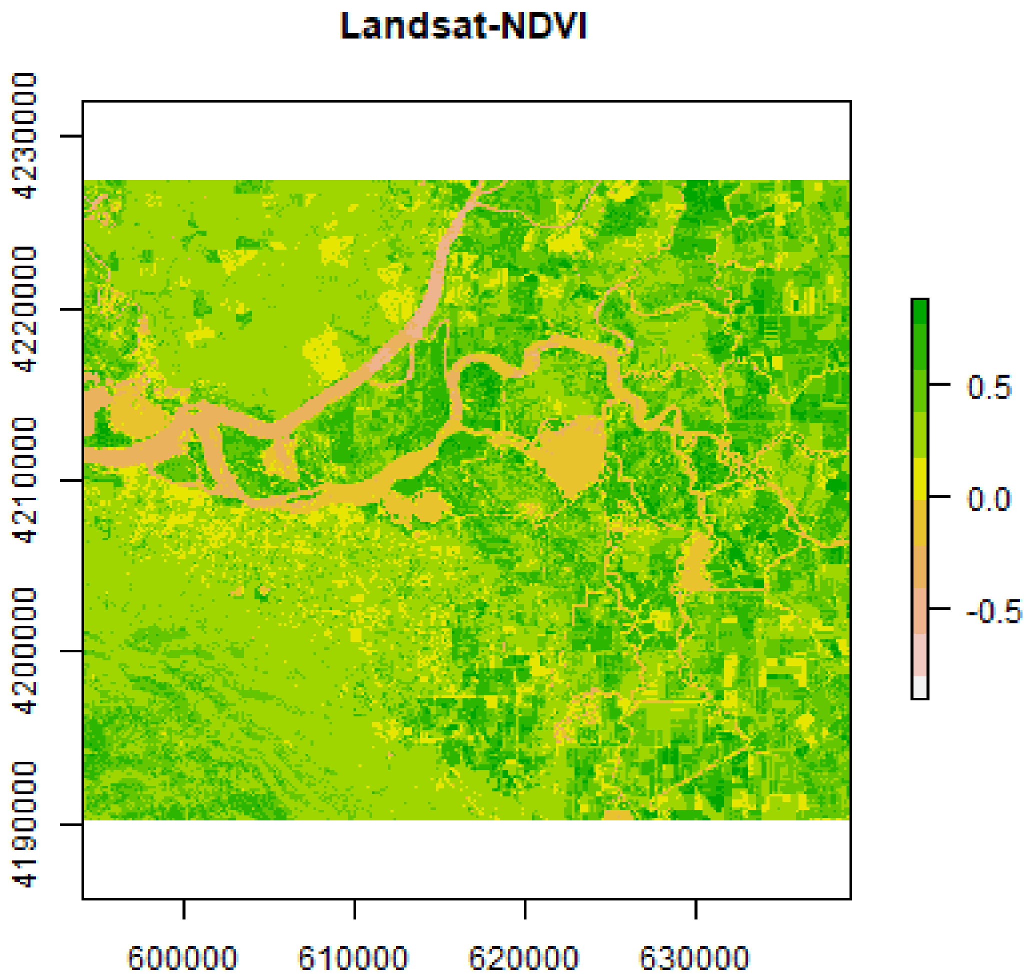
```
library(raster)
raslist <- paste0('data/rs/LC08_044034_20170614_B', 1:11, ".tif")
landsat <- stack(raslist)
landsatRGB <- landsat[[c(4,3,2)]]
landsatFCC <- landsat[[c(5,4,3)]]
```

3.1 Vegetation indices

Let's define a general function for a ratio based (vegetation) index. In the function below, img is a mutilayer Raster* 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)
}
```

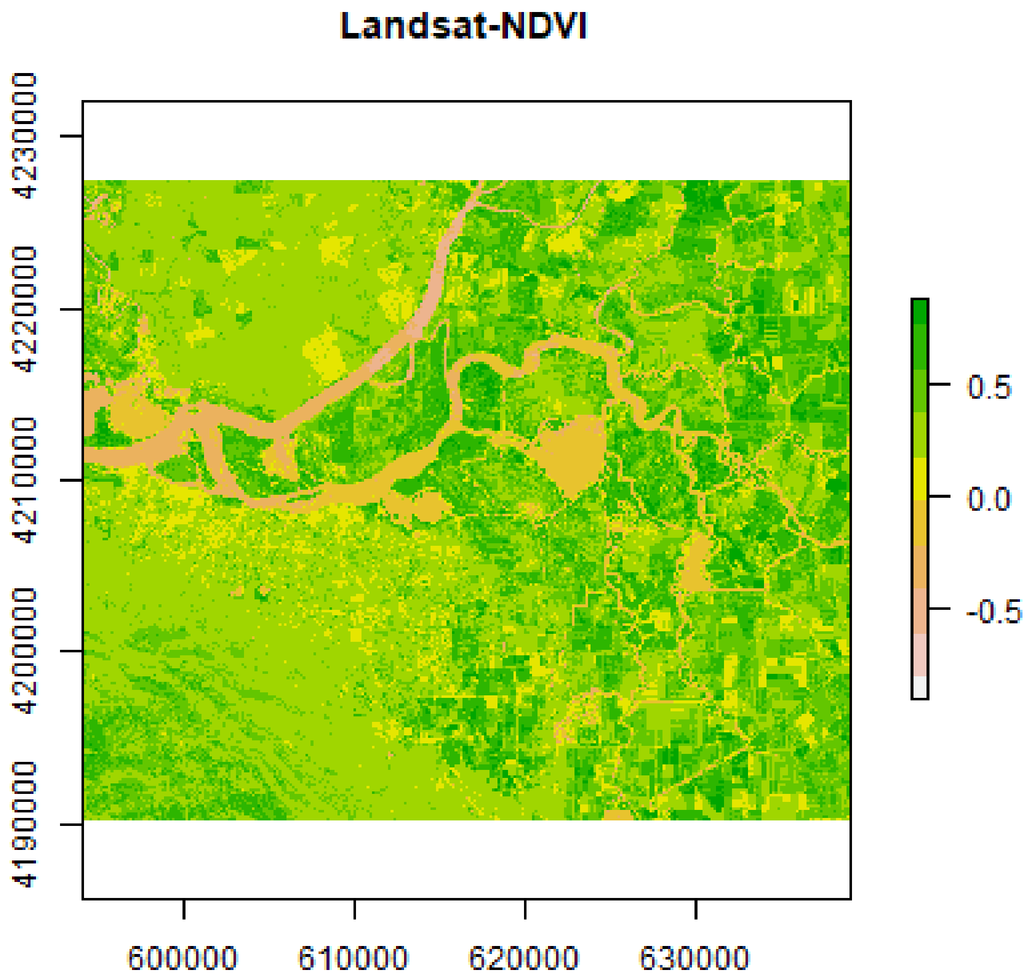
```
# For Landsat NIR = 5, red = 4.
ndvi <- vi(landsat, 5, 4)
plot(ndvi, col = rev(terrain.colors(10)), main = "Landsat-NDVI")
```



You can see the variation in greenness from the plot.

An alternative way to accomplish this is like this

```
vi2 <- function(x, y) {
    (x - y) / (x + y)
}
ndvi2 <- overlay(landsat[[5]], landsat[[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.*

3.2 Histogram

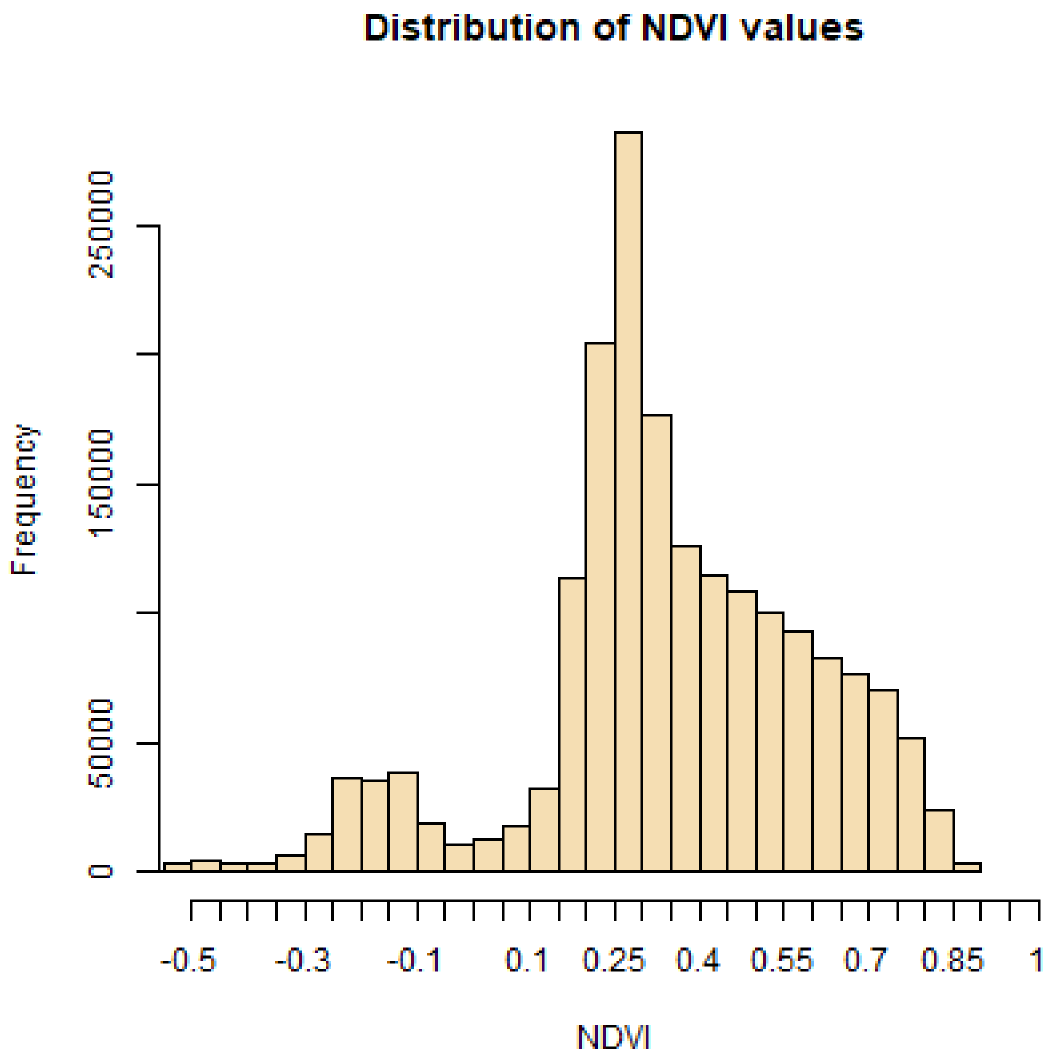
We can explore the distribution of values contained within our raster using the hist() function which produces a histogram. Histograms are often useful in identifying outliers and bad data values in our raster data.

```
# view histogram of data
hist(ndvi,
     main = "Distribution of NDVI values",
     xlab = "NDVI",
     ylab= "Frequency",
     col = "wheat",
     xlim = c(-0.5, 1),
```

(continues on next page)

(continued from previous page)

```
     breaks = 30,
     xaxt = 'n')
axis(side=1, at = seq(-0.5,1, 0.05), labels = seq(-0.5,1, 0.05))
```



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

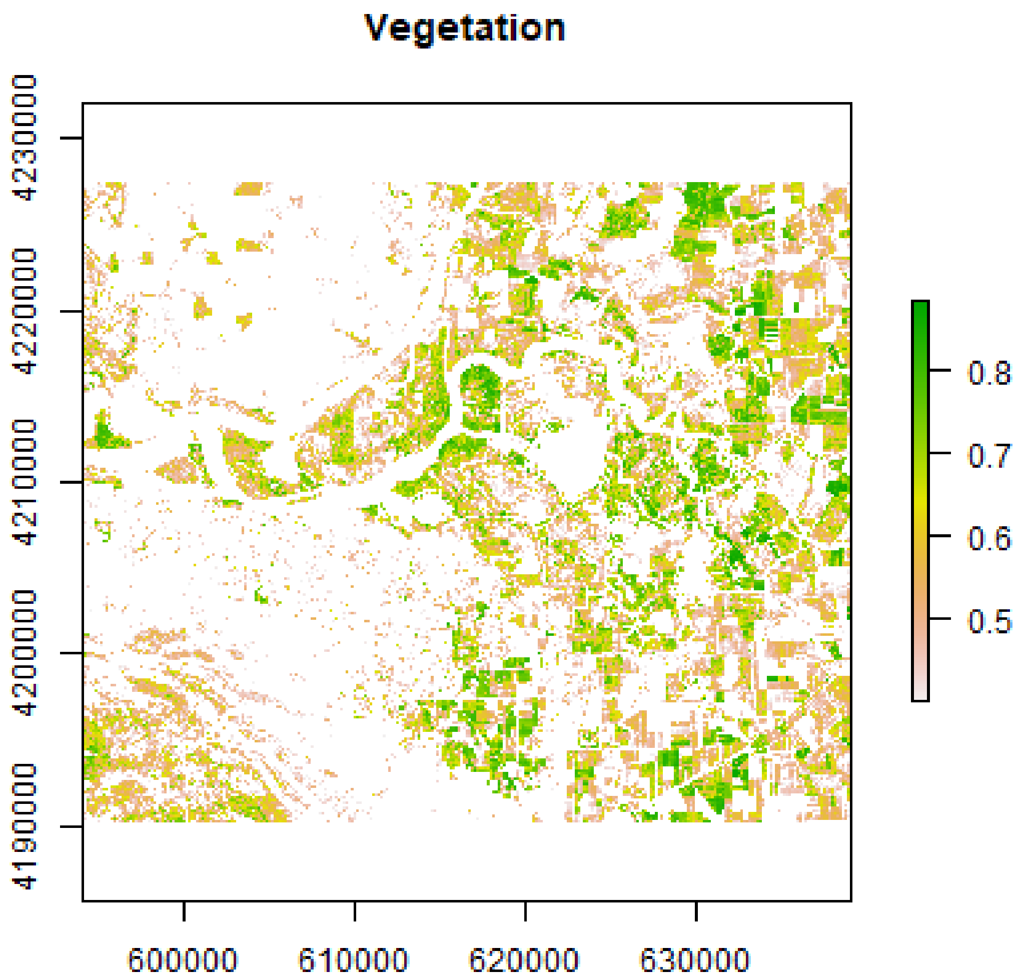
Question 2: *Make histograms of the values the vegetation indices developed in question 1.*

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.

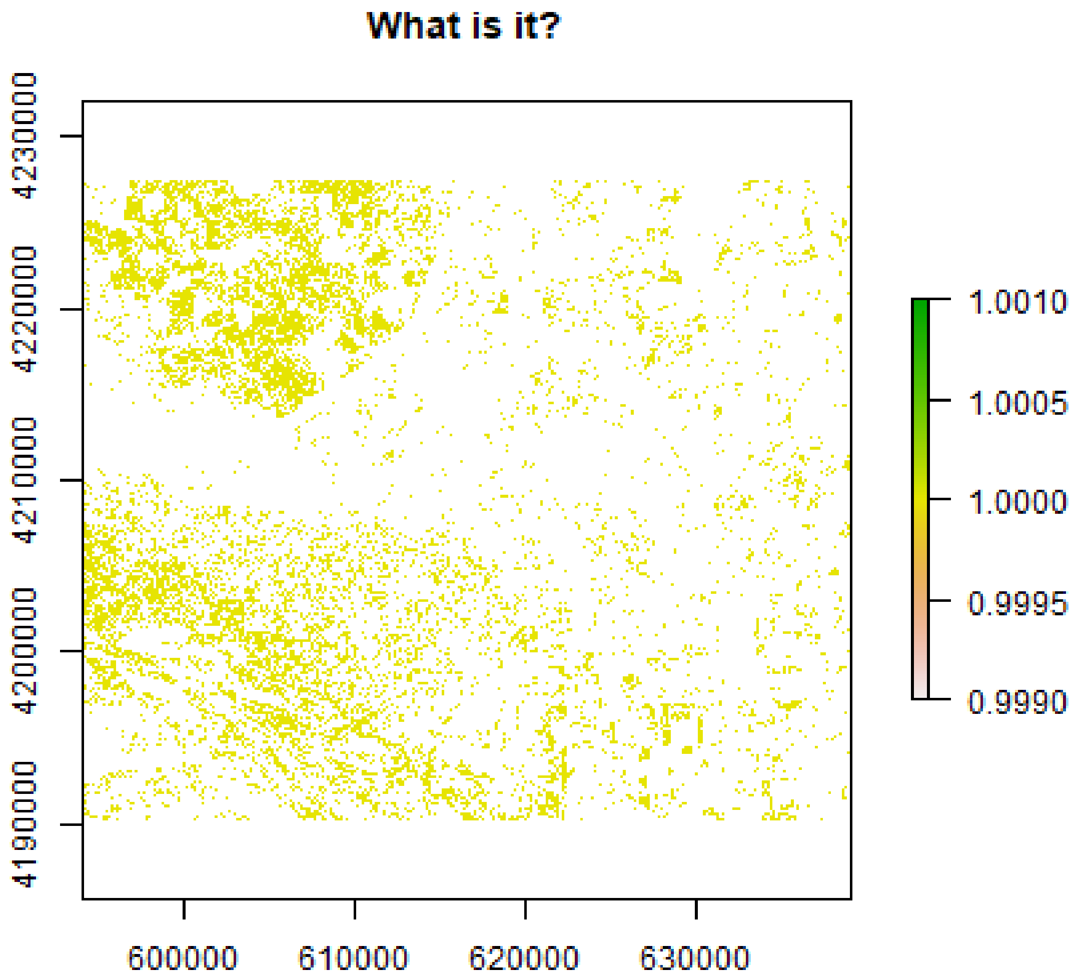
Cells with NDVI values greater than 0.4 are definitely vegetation. The following operation masks all cells that are perhaps not vegetation.

```
veg <- reclassify(ndvi, cbind(-Inf, 0.4, NA))
plot(veg, main='Vegetation')
```



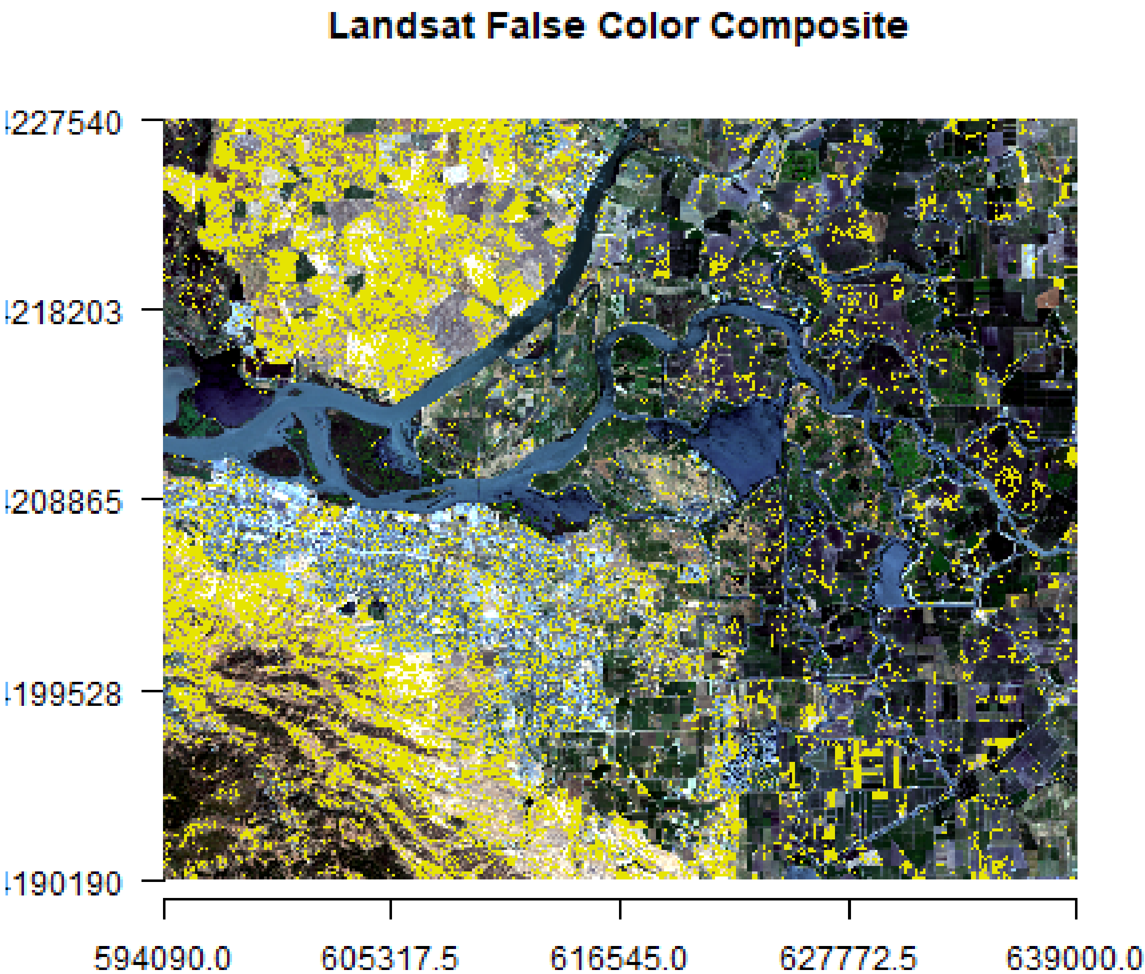
Let's map the area that corresponds to the peak between 0.25 and 0.3 in the NDVI histogram.

```
land <- reclassify(ndvi, c(-Inf, 0.25, NA,  0.25, 0.3, 1,  0.3, Inf, NA))
plot(land, main = 'What is it?')
```



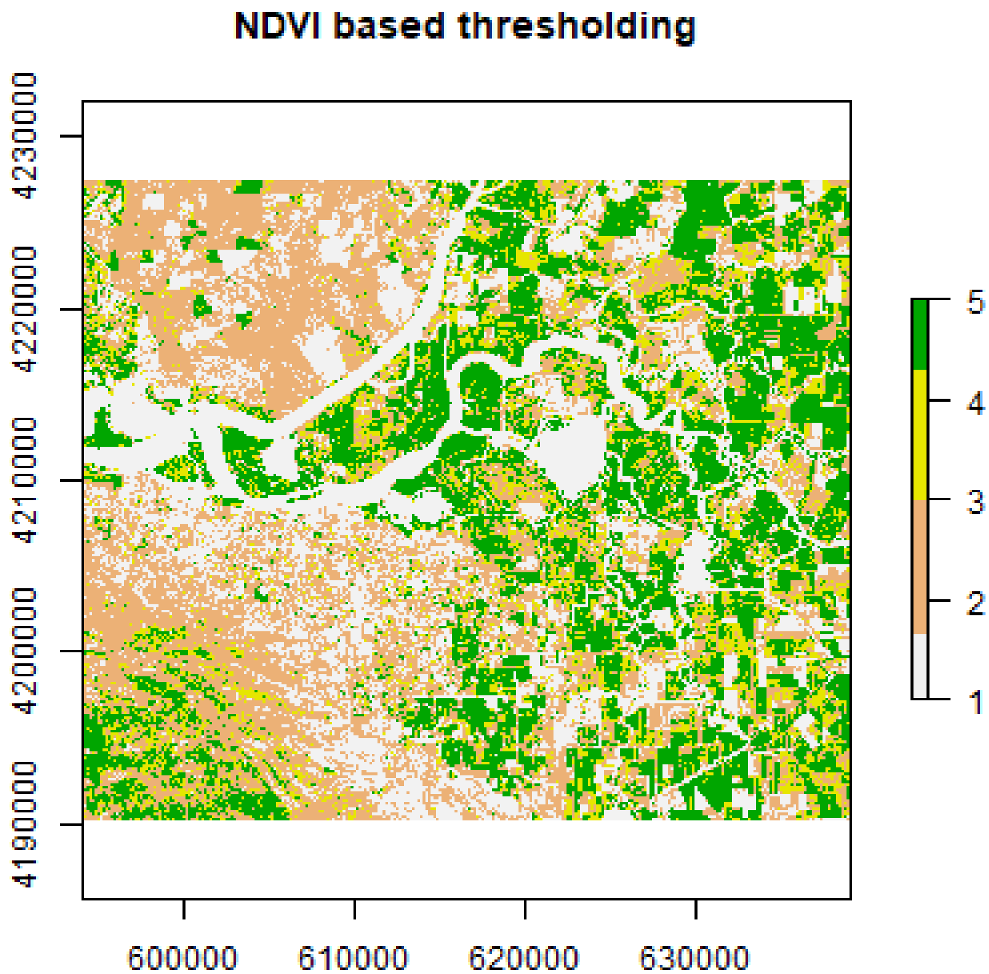
These may be the open areas. You can plot `land` over original `landsatFCC` raster to find out more.

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



You can also create classes for different amount of vegetation presence.

```
vegc <- reclassify(ndvi, c(-Inf,0.25,1, 0.25,0.3,2, 0.3,0.4,3, 0.4,0.5,4, 0.5,Inf, 5))
plot(vegc,col = rev(terrain.colors(4)), main = 'NDVI based thresholding')
```



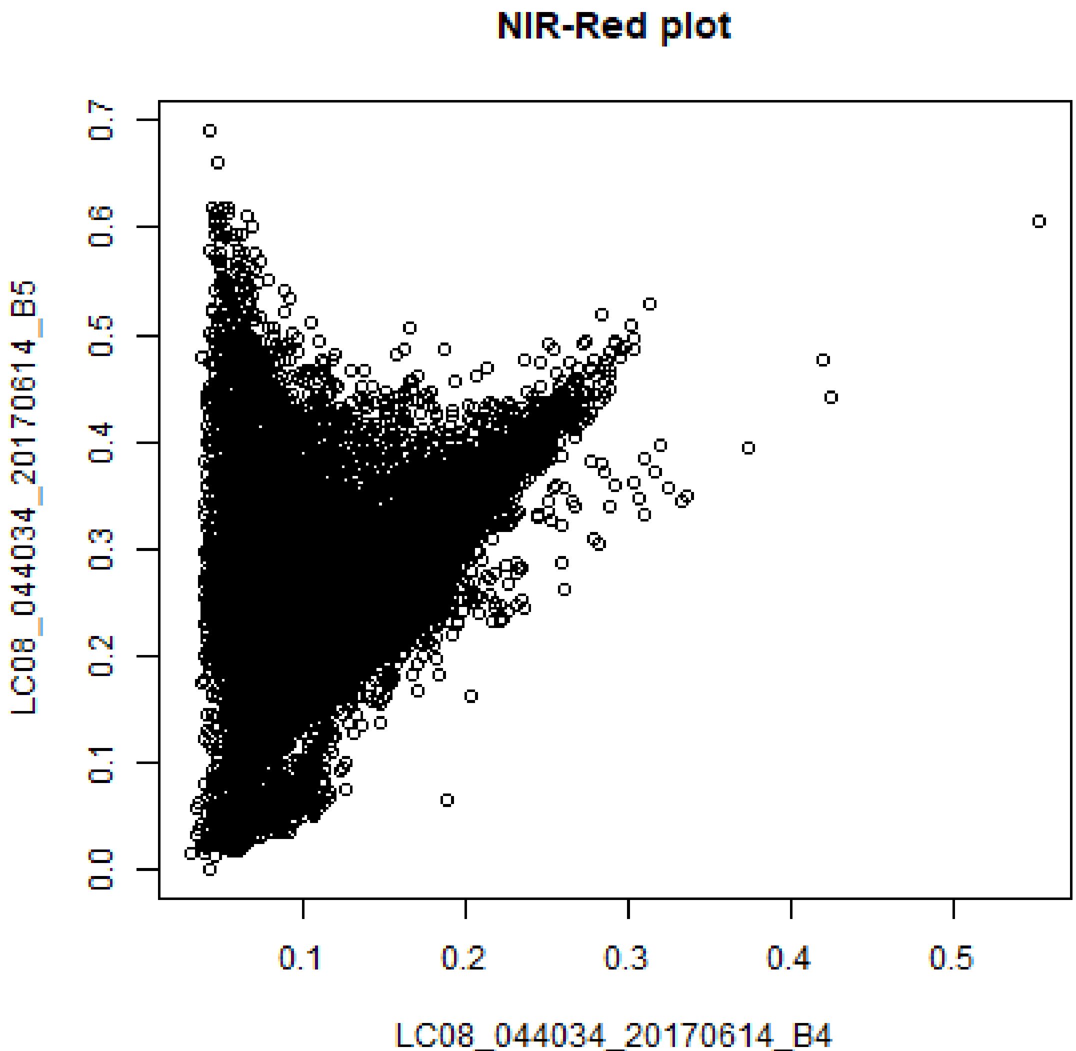
Question 3: *Is it possible to find water using thresholding of NDVI or any other indices?*

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 <- sampleRandom(landsat, 10000)
plot(sr[,c(4,5)], main = "NIR-Red plot")
```



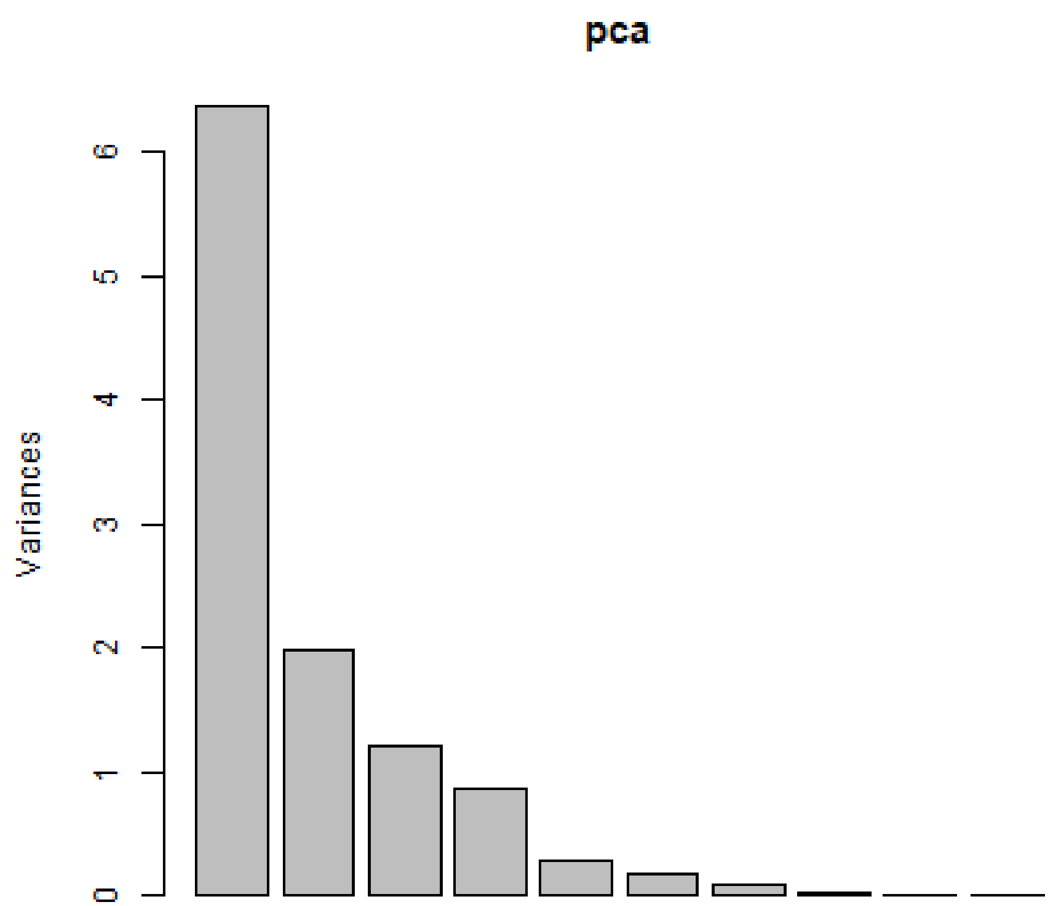
This is known as vegetation and soil-line plot (Same as the scatter plot in earlier section).

```
pca <- prcomp(sr, scale = TRUE)
pca
## Standard deviations (1, .., p=11):
##  [1] 2.52687022 1.40533428 1.09901821 0.92507423 0.53731672 0.42657919
##  [7] 0.28002273 0.12466139 0.09031384 0.04761419 0.03609857
##
## Rotation (n x k) = (11 x 11):
##                                 PC1        PC2         PC3         PC4
## LC08_044034_20170614_B1  0.2973469 -0.3516438 -0.29454767 -0.06983456
## LC08_044034_20170614_B2  0.3387629 -0.3301194 -0.16407835 -0.03803678
## LC08_044034_20170614_B3  0.3621173 -0.2641152  0.07373240  0.04090884
## LC08_044034_20170614_B4  0.3684797 -0.1659582  0.10260552  0.03680852
## LC08_044034_20170614_B5  0.1546322  0.1816252  0.71354112  0.32017620
## LC08_044034_20170614_B6  0.3480230  0.2108982  0.23064060  0.16598851
## LC08_044034_20170614_B7  0.3496281  0.2384417 -0.11662258  0.07600209
```

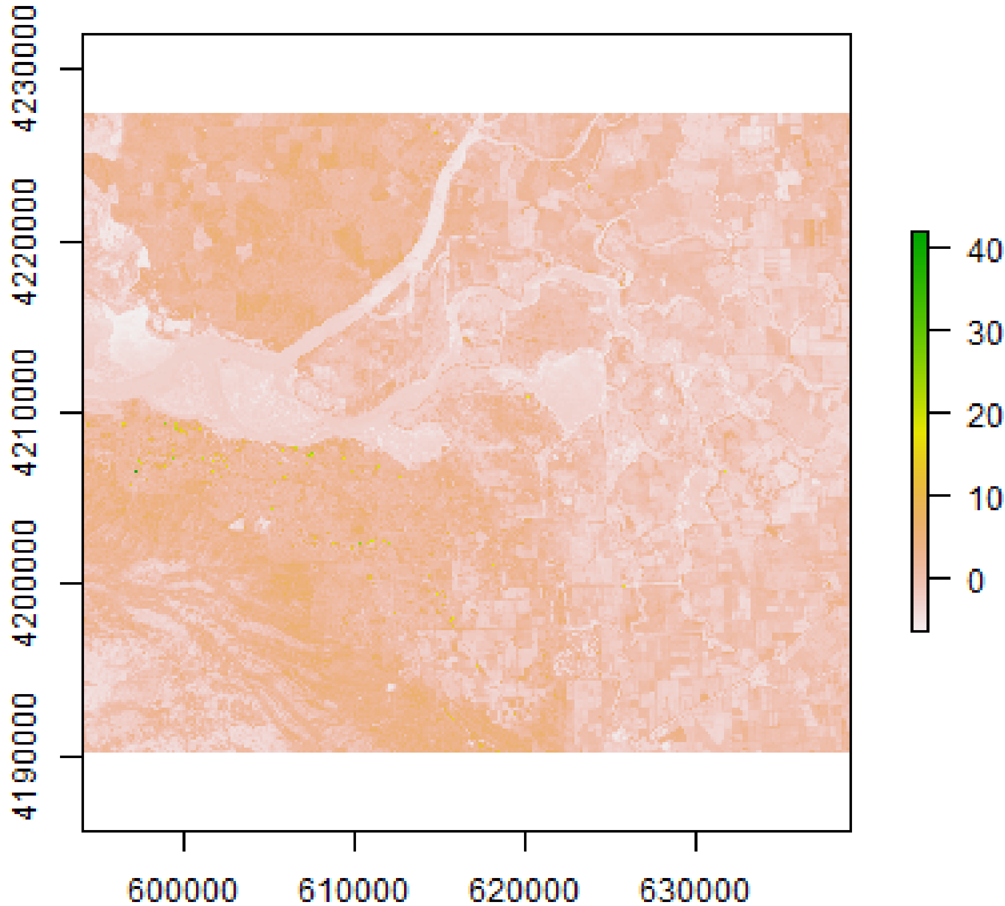
(continues on next page)

(continued from previous page)

```
## LC08_044034_20170614_B8  0.3490464 -0.2007305  0.08765521  0.02303421
## LC08_044034_20170614_B9  0.1314827  0.1047365  0.33741447 -0.92325315
## LC08_044034_20170614_B10 0.2497611  0.4912132 -0.29286315 -0.02950655
## LC08_044034_20170614_B11 0.2472765  0.4931489 -0.29515754 -0.03176549
##                                 PC5          PC6         PC7          PC8
## LC08_044034_20170614_B1   0.49474685  0.175510232 -0.23948553  0.215745065
## LC08_044034_20170614_B2   0.22121122  0.094184121  0.06447037  0.216537517
## LC08_044034_20170614_B3   0.08482031  0.009040232  0.30511210 -0.518233675
## LC08_044034_20170614_B4  -0.33835490 -0.066844529  0.60174786  0.012437959
## LC08_044034_20170614_B5   0.51960822 -0.059561476 -0.07348455 -0.083217504
## LC08_044034_20170614_B6  -0.29437062  0.317984598 -0.02106132  0.632178645
## LC08_044034_20170614_B7  -0.25404931  0.525411720 -0.40543545 -0.478543437
## LC08_044034_20170614_B8  -0.31407992 -0.673584139 -0.52642131  0.003527306
## LC08_044034_20170614_B9   0.03040161  0.059642905 -0.03152221 -0.002775800
## LC08_044034_20170614_B10  0.16317572 -0.243735973  0.14341520  0.041736319
## LC08_044034_20170614_B11  0.19294569 -0.241611777  0.11997475 -0.022446494
##                                  PC9         PC10          PC11
## LC08_044034_20170614_B1   0.122812108  0.535959306  0.1203473847
## LC08_044034_20170614_B2   0.091063964 -0.773112627 -0.1817872036
## LC08_044034_20170614_B3  -0.644305383  0.070860458  0.0540175730
## LC08_044034_20170614_B4   0.543822097  0.218324141  0.0135097158
## LC08_044034_20170614_B5   0.209682702 -0.040186292 -0.0004965182
## LC08_044034_20170614_B6  -0.397210135  0.089423617 -0.0045305608
## LC08_044034_20170614_B7   0.248871690 -0.074907393  0.0003958921
## LC08_044034_20170614_B8  -0.012642132  0.002411975  0.0007749022
## LC08_044034_20170614_B9   0.003176077 -0.000832654  0.0019986985
## LC08_044034_20170614_B10 -0.003574004 -0.158727672  0.6900281980
## LC08_044034_20170614_B11 -0.043475408  0.148102091 -0.6878990264
screeplot(pca)
```

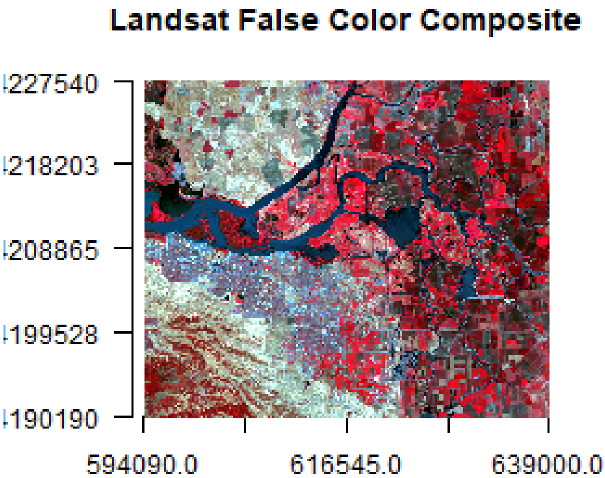



```
pci <- predict(landsat, pca, index = 1:2)
plot(pci[[1]])
```



The first principal component highlights the boundaries between land use classes or spatial details, which is the most common information among all wavelengths. it is difficult to understand what the second principal component is highlighting. Lets try thresholding again:

```
pc2 <- reclassify(pci[[2]], c(-Inf,0,1,0,Inf,NA))
par(mfrow = c(1,2))
plotRGB(landsatFCC, r = 1, g = 2, b = 3, axes = TRUE, stretch = "lin", main = "Landsat␣
↪False Color Composite")
plotRGB(landsatFCC, r = 1, g = 2, b = 3, axes = TRUE, stretch = "lin", main = "Landsat␣
↪False Color Composite")
plot(pc2, legend = FALSE, add = TRUE)
```



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

CHAPTER

FOUR

UNSUPERVISED CLASSIFICATION

In this chapter we explore unsupervised classification. Various unsupervised classification algorithms exist, and the choice of algorithm can affect the results. We will explore only one algorithm (k-means) 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(raster)
landsat5 <- stack('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 `extent` using the `drawExtent()` function).

4.1 kmeans classification

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

# crop landsat by the extent
ndvi <- crop(ndvi, e)
ndvi
## class      : RasterLayer
## dimensions : 252, 304, 76608  (nrow, ncol, ncell)
## 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)
## crs         : +proj=longlat +datum=WGS84 +no_defs
## source      : memory
## names       : layer
## values      : -0.3360085, 0.7756007  (min, max)

# convert the raster to vecor/matrix
nr <- getValues(ndvi)
str(nr)
##  num [1:76608] 0.245 0.236 0.272 0.277 0.277 ...
```

Please note that `getValues` converted the `ndvi` RasterLayer 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)

# We want to create 10 clusters, allow 500 iterations, start with 5 random sets using
↪"Lloyd" method
kmncluster <- kmeans(na.omit(nr), 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 RasterLayer of the same dimension as the `ndvi`.

```
# Use the ndvi object to set the cluster values to a new raster
knr <- setValues(ndvi, kmncluster$cluster)

# You can also do it like this
knr <- raster(ndvi)
values(knr) <- kmncluster$cluster
```

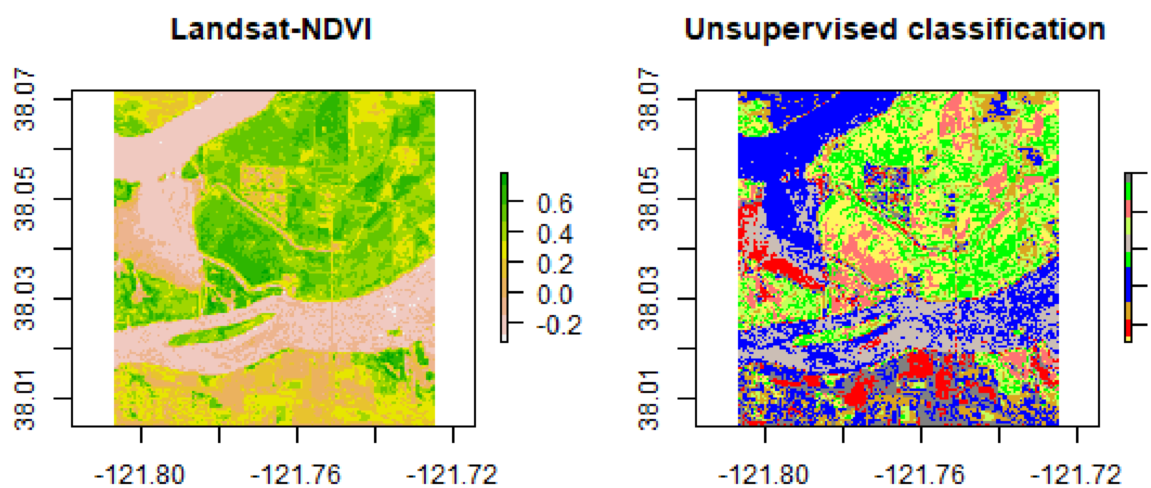
(continues on next page)

(continued from previous page)

```
knr
## class      : RasterLayer
## dimensions : 252, 304, 76608  (nrow, ncol, ncell)
## resolution : 0.0002694946, 0.0002694946  (x, y)
## extent     : -121.807, -121.725, 38.00413, 38.07204  (xmin, xmax, ymin, ymax)
## crs        : +proj=longlat +datum=WGS84 +no_defs
## source     : memory
## names      : layer
## values     : 1, 10  (min, max)
```

We can see that knr is a RasterLayer 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 )
```



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 3-band RGB 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. E.g. cluster 4 and 5 are water.*

CHAPTER

FIVE

SUPERVISED CLASSIFICATION

Here we explore supervised classification. 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, fieldwork, 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 perform the following steps:

- Generate sample sites based on a reference raster
- 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 Reference data

The National Land Cover Database 2011 (NLCD 2011) is a land cover product for the USA. 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.

You can find the class names in NCLD 2011 (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.

```
library(raster)
nlcd <- brick('data/rs/nlcd-L1.tif')
names(nlcd) <- c("nlcd2001", "nlcd2011")

# The class names and colors for plotting
nlcdclass <- c("Water", "Developed", "Barren", "Forest", "Shrubland", "Herbaceous",
↪"Planted/Cultivated", "Wetlands")
classdf <- data.frame(classvalue1 = c(1,2,3,4,5,7,8,9), classnames1 = nlcdclass)
```

(continues on next page)

(continued from previous page)

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

# Now we ratify (RAT = "Raster Attribute Table") the ncld2011 (define RasterLayer as a␣
↪categorical variable). This is helpful for plotting.
nlcd2011 <- nlcd[[2]]
nlcd2011 <- ratify(nlcd2011)
rat <- levels(nlcd2011)[[1]]

#
rat$landcover <- nlcdclass
levels(nlcd2011) <- rat
```

We did a lot of things here. Take a step back and read more about `ratify`.

Note There is no class with value 6.

5.2 Generate sample sites

As we discussed in the class, training and/or validation data can come from a variety of sources. In this example we will generate the training and validation sample sites using the NLCD reference RasterLayer. Alternatively, you can use predefined sites that you may have collected from other sources. We will generate the sample sites following a stratified random sampling to ensure samples from each LULC class.

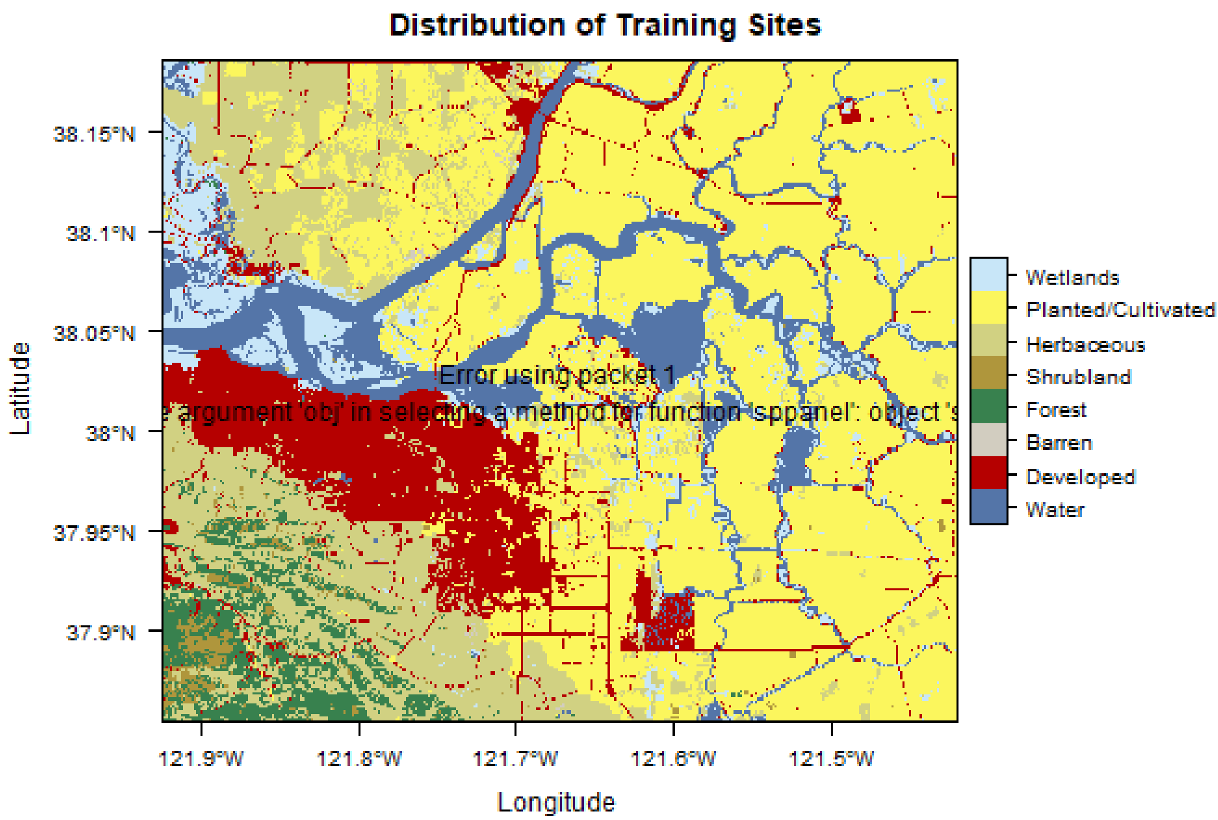
```
# Load the training sites locations
# Set the random number generator to reproduce the results
set.seed(99)

# Sampling
samp2011 <- sampleStratified(nlcd2011, size = 200, na.rm = TRUE, sp = TRUE)
samp2011
## class       : SpatialPointsDataFrame
## features    : 1600
## extent      : -121.9257, -121.4225, 37.85415, 38.18536  (xmin, xmax, ymin, ymax)
## crs         : +proj=longlat +datum=WGS84 +no_defs
## variables   : 2
## names       :    cell, nlcd2011
## min values  :     413,        1
## max values  : 2307837,        9
# Number of samples in each class
table(samp2011$nlcd2011)
##
##   1   2   3   4   5   7   8   9
## 200 200 200 200 200 200 200 200
```

You can see there are two variables in `samp2011`. The `cell` column contains cell numbers of `nlcd2011` sampled. `nlcd2011` column contains the class values (1-9). We will drop the `cell` column later.

Here `nlcd` has integer values between 1-9. You will often find classnames are provided as string labels (e.g. water, crop, vegetation). You will need to 'relabel' class names 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.

Let's plot the training sites over the `nlcd2011` RasterLayer to visualize the distribution of sampling locations.

```
library(rasterVis)
plt <- levelplot(nlcd2011, col.regions = classcolor, main = 'Distribution of Training␣
↪Sites')
print(plt + layer(sp.points(samp2011, pch = 3, cex = 0.5, col = 1)))
```



`rasterVis` offers more advanced (trellis/lattice) plotting of Raster* objects. Please install the package if it is not available for your machine.

5.3 Extract values for sites

Here is our Landsat data.

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

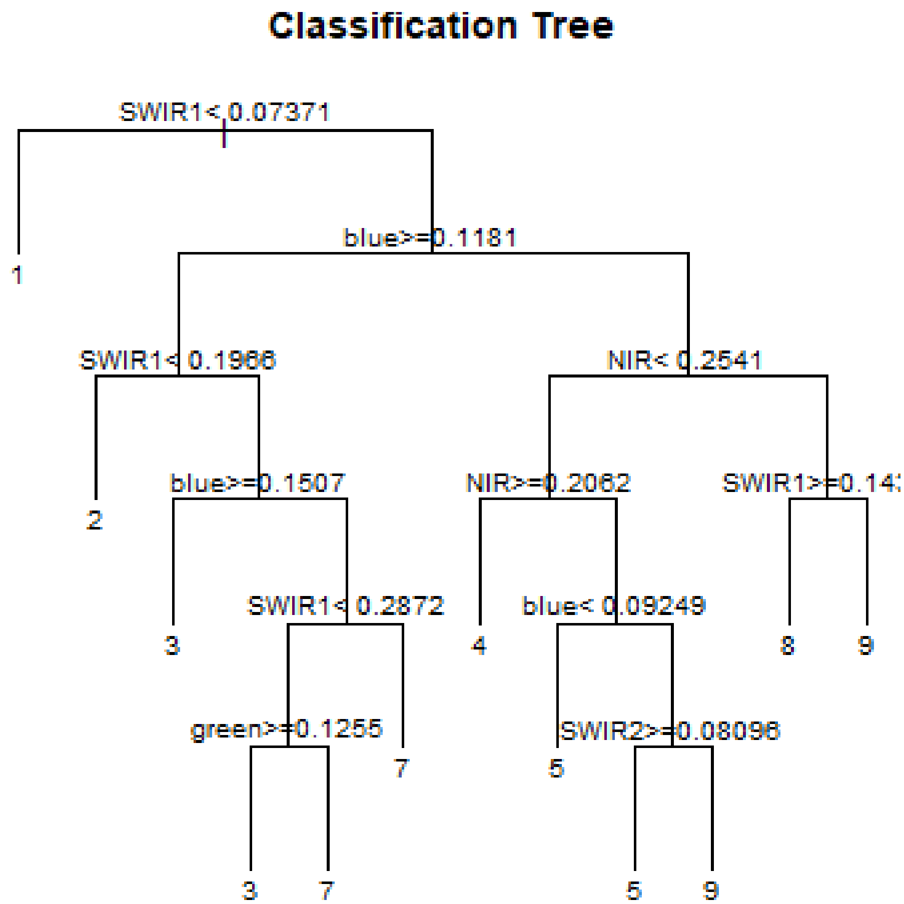
Once we have the sites, we can extract the cell values from `landsat5` RasterStack. These band values will be the predictor variables and "classvalues" from `nlcd2011` will be the response variable.

```
# Extract the layer values for the locations
sampvals <- extract(landsat5, samp2011, df = TRUE)

# sampvals no longer has the spatial information. To keep the spatial information you␣
↪use `sp=TRUE` argument in the `extract` function.

# drop the ID column
sampvals <- sampvals[, -1]

# combine the class information with extracted values
sampdata <- data.frame(classvalue = samp2011@data$nlcd2011, sampvals)
```

5.4 Train the classifier

Now we will train the classification algorithm using `training2011` dataset.

```
library(rpart)

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

# print(model.class)

# Plot the trained classification tree
plot(cart, uniform=TRUE, main="Classification Tree")
text(cart, cex = 0.8)
```



In the classification tree plot classvalues are printed at the leaf nodes. You can find the corresponding land use land cover names from the `classdf` data.frame.

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

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

5.5 Classify

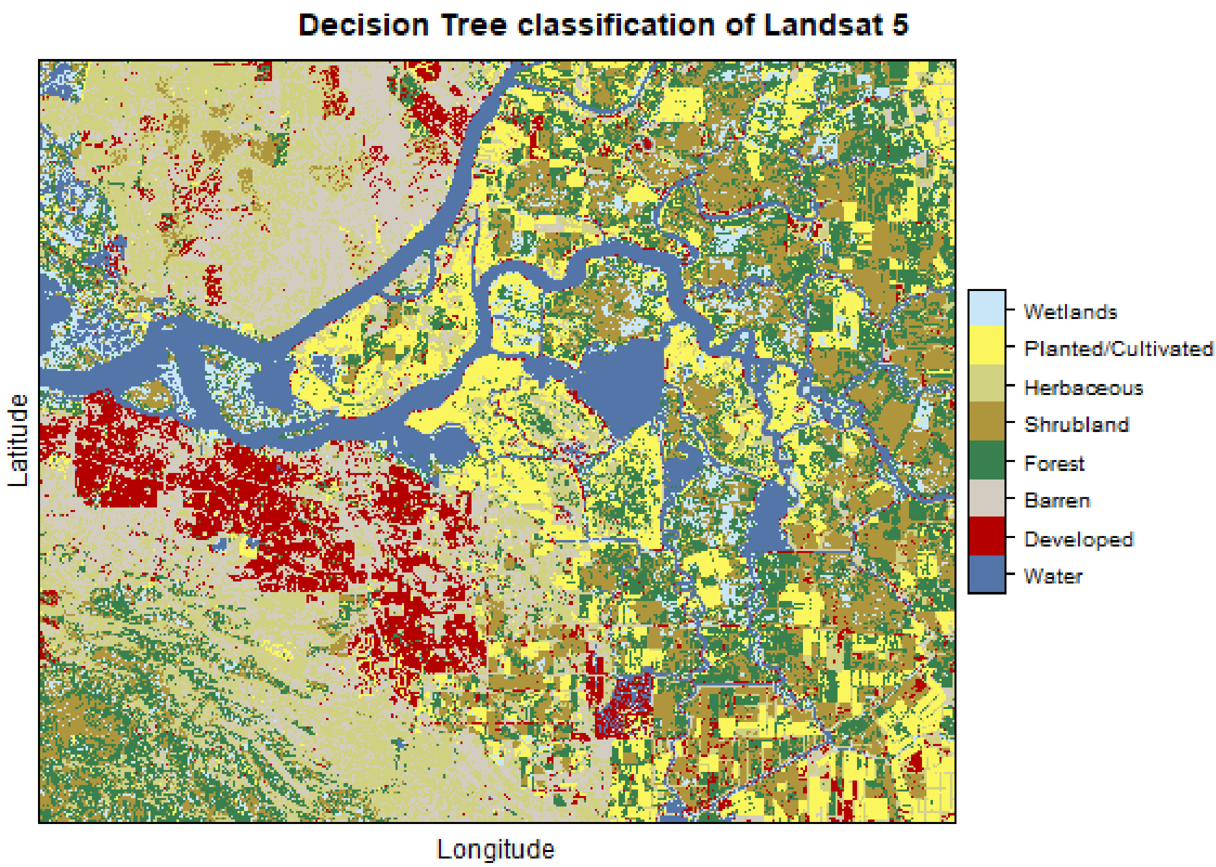
Now we have our trained classification model (`cart`), we can use it to make predictions, that is, to classify all cells in the `landsat5` RasterStack.

Important The names in the Raster object to be classified should exactly match those expected by the model. This will be the case if the same Raster object was used (via extract) to obtain the values to fit the model.

```
# Now predict the subset data based on the model; prediction for entire area takes␣
↪longer time
pr2011 <- predict(landsat5, cart, type='class')
pr2011
## class      : RasterLayer
## dimensions : 1230, 1877, 2308710  (nrow, ncol, ncell)
## resolution : 0.0002694946, 0.0002694946  (x, y)
## extent     : -121.9258, -121.42, 37.85402, 38.1855  (xmin, xmax, ymin, ymax)
## crs        : +proj=longlat +datum=WGS84 +no_defs
## source     : memory
## names      : layer
## values     : 1, 9  (min, max)
## attributes :
##        ID value
##  from:  1     1
##   to :  8     9
```

Now plot the classification result using `rasterVis`. See will set the `classnames` for the `classvalues`.

```
pr2011 <- ratify(pr2011)

rat <- levels(pr2011)[[1]]

rat$legend <- classdf$classnames

levels(pr2011) <- rat

levelplot(pr2011, maxpixels = 1e6,
          col.regions = classcolor,
          scales=list(draw=FALSE),
          main = "Decision Tree classification of Landsat 5")
```



Question 1:*Plot ``nlcd2011`` and ``pr2011`` side-by-side and comment about the accuracy of the prediction (e.g. mixing between cultivated crops, pasture, grassland and shrubs).*

You may need to select more samples and use additional predictor variables. The choice of classifier also plays an important role.

5.6 Model evaluation

Now let's 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 (`validation2011`).

To evaluate any model, you can use k-fold cross-validation. 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).

```
library(dismo)
set.seed(99)
j <- kfold(sampdata, k = 5, by=sampdata$classvalue)
table(j)
## j
##   1   2   3   4   5
## 320 320 320 320 320
```

Now we train and test the model five times, each time computing a confusion matrix that we store in a list.

```
x <- list()

for (k in 1:5) {
    train <- sampdata[j!= k, ]
    test <- sampdata[j == k, ]
    cart <- rpart(as.factor(classvalue)~., data=train, method = 'class', minsplit = 5)
    pclass <- predict(cart, test, type='class')
    # create a data.frame using the reference and prediction
    x[[k]] <- cbind(test$classvalue, as.integer(pclass))
}
```

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')


conmat <- table(y)
# change the name of the classes
colnames(conmat) <- classdf$classnames
rownames(conmat) <- classdf$classnames
conmat
##                     predicted
## observed              Water Developed Barren Forest Shrubland Herbaceous
##   Water                 175         6      0      3         0          0
##   Developed               2        90     51      8        10         22
##   Barren                  7        39     82      4        19         38
##   Forest                  0         2      1    106        57          1
##   Shrubland               0         3      5     59       102         12
##   Herbaceous              0         9     36     10        27        109
##   Planted/Cultivated      0         7     11     34        42         19
##   Wetlands               18        10      6     36        29          5
##                     predicted
```

(continues on next page)

(continued from previous page)

```
## observed             Planted/Cultivated Wetlands
##   Water                               7        9
##   Developed                          11        6
##   Barren                              5        6
##   Forest                              6       27
##   Shrubland                          12        7
##   Herbaceous                          8        1
##   Planted/Cultivated                 69       18
##   Wetlands                           33       63
```

Question 2:*Comment on the miss-classification between different classes.*

Question 3:*Can you think of ways to to improve the accuracy.*

Compute the overall accuracy and the "Kappa" statistic.

Overall accuracy:

```
# number of cases
n <- sum(conmat)
n
## [1] 1600

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

# Overall Accuracy
OA <- sum(diag) / n
OA
## [1] 0.4975
```

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] 0.4257143
```

Producer and user accuracy

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

# User accuracy
UA <- diag / rowsums
```

(continues on next page)

(continued from previous page)

```
outAcc <- data.frame(producerAccuracy = PA, userAccuracy = UA)

outAcc
##                    producerAccuracy userAccuracy
## Water                     0.8663366        0.875
## Developed                 0.5421687        0.450
## Barren                    0.4270833        0.410
## Forest                    0.4076923        0.530
## Shrubland                 0.3566434        0.510
## Herbaceous                0.5291262        0.545
## Planted/Cultivated        0.4569536        0.345
## Wetlands                  0.4598540        0.315
```

Question 4:*Perform the classification using Random Forest classifiers from the ``randomForest`` package*

Question 5:*Plot the results of rpart and Random Forest classifier side-by-side.*

Question 6 (optional):*Repeat the steps for the year 2001 using Random Forest*. Use the cloud-free composite image `data/centralvalley-2001LE7.tif`. This is Landsat 7 data . Use as reference data the National Land Cover Database 2001 (NLCD 2001) for the subset of the California Central Valley.*

Question 7 (optional):*We have trained the classifiers using 200 samples for each class. Investigate the effect of sample size on classification. Repeat the steps with different subsets, e.g. a sample size of 150, 100, 50 per class, and compare the results. Use the same holdout samples for model evaluation.*