

# Introduction to remote sensing data analysis using R

## Remote sensing image sources

Getting remote sensing image for a specific project remains one of the most challenging steps in the workflow. You have to find the data most suitable for your particular objective. For example, MODIS data at 250 m spatial resolution is not sufficient for mapping agricultural plots in developing countries. Few important properties to consider while searching the remote sensing data are (not in the order of importance):

1. spatial resolution or pixel size
2. date or time of the year/season
3. cloud-cover
4. wavelengths to measure different physical properties
5. availability of historical data
6. noise or artifacts in data (read about problems in Landsat ETM+)

There are many numerous choices of remote sensing data. However, in this tutorial we'll only focus freely available Sentinel, Landsat 8 and MODIS data. You can access these data from following sources:

- 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](https://lpdaac.usgs.gov/data_access/data_pool)
- v. <https://scihub.copernicus.eu/>
- vi. <https://aws.amazon.com/public-data-sets/landsat/>

This site lists 15 sources of freely available remote sensing image.

Advanced users can also consider using Google Earth Engine and MODISTools to access different satellite data.

## Basic image manipulation and visualization

In this section we will learn about reading remote sensing image into R-environment. These image products are available in gridded or raster format. Please remember from its inception, R was designed for statistical data analysis and modeling. Therefore you will not get all the options available in geospatial image processing software. One of R's unique strength lies in its capability to bring a wide variety of statistical tools to analyze raster image.

### Read image and display basic image properties

Multi-band raster data (for example multi-spectral Landsat image) can be read as a `RasterStack` or `RasterBrick` in R. You can read multi-band data using:

```
> library(raster)
> #load Landsat data
> r <- brick('landsat8-2016march.tif')
>
> #load Sentinel data
> s <- brick('sentinel.tif')
```

```

>
> r
class      : RasterBrick
dimensions  : 642, 593, 380706, 6 (nrow, ncol, ncell, nlayers)
resolution  : 30, 30 (x, y)
extent     : 251940, 269730, -382140, -362880 (xmin, xmax, ymin, ymax)
coord. ref. : +proj=utm +zone=37 +datum=WGS84 +units=m +no_defs +ellps=WGS84 +towgs84=0,0,0
data source : C:\Users\anigh\Google Drive\teaching\remotesensing-tza-ani\data\landsat8-2016march.tif
names       : landsat8.2016march.1, landsat8.2016march.2, landsat8.2016march.3, landsat8.2016march.4, 1
> s
class      : RasterBrick
dimensions  : 1934, 1786, 3454124, 10 (nrow, ncol, ncell, nlayers)
resolution  : 10, 10 (x, y)
extent     : 251900.9, 269760.9, -382182.3, -362842.3 (xmin, xmax, ymin, ymax)
coord. ref. : +proj=utm +zone=37 +datum=WGS84 +units=m +no_defs +ellps=WGS84 +towgs84=0,0,0
data source : C:\Users\anigh\Google Drive\teaching\remotesensing-tza-ani\data\sentinel.tif
names       : sentinel.1, sentinel.2, sentinel.3, sentinel.4, sentinel.5, sentinel.6, sentinel.7, sentinel.8
min values  : 575.5850, 401.0961, 235.0271, 307.0000, 464.0000, 489.0000, 470.8530, 371.0000
max values  : 6397.991, 6278.674, 6372.010, 5482.000, 6586.000, 7242.000, 11990.447, 7095.0000

```

The last argument prints the properties of `lsat1` object. You can see the spatial resolution, extent, bands, projection and minimum-maximum values.

Single band raster data can also be read using `raster` function.

### Image information and statistics

Following examples show how various properties can be accessed from the image.

```

> # Summarize a Raster* object. A sample is used for very large files.
> # summary(lsat1, maxsamp = 5000)
> # number of bands
> nlayers(r)
[1] 6
> nlayers(s)
[1] 10
>
> # coordinate reference system (CRS)
> crs(r)
CRS arguments:
+proj=utm +zone=37 +datum=WGS84 +units=m +no_defs +ellps=WGS84
+towgs84=0,0,0
> crs(s)
CRS arguments:
+proj=utm +zone=37 +datum=WGS84 +units=m +no_defs +ellps=WGS84
+towgs84=0,0,0
>
> # get spatial resolution
> xres(r)
[1] 30
>
> yres(r)
[1] 30

```

```

>
> res(r)
[1] 30 30
>
> res(s)
[1] 10 10
>
> # Number of rows, columns, or cells
> ncell(r)
[1] 380706
> dim(s)
[1] 1934 1786   10

```

The data products have different number of bands. For Landsat and Sentinel, these are different spectral bands. The term ‘band’ is frequently used in remote sensing to refer to a variable (layer) in a multi-variable dataset as these variables typically represent reflection in different bandwidths in the electromagnetic spectrum.

Some of these functions can also be used to set image properties also.

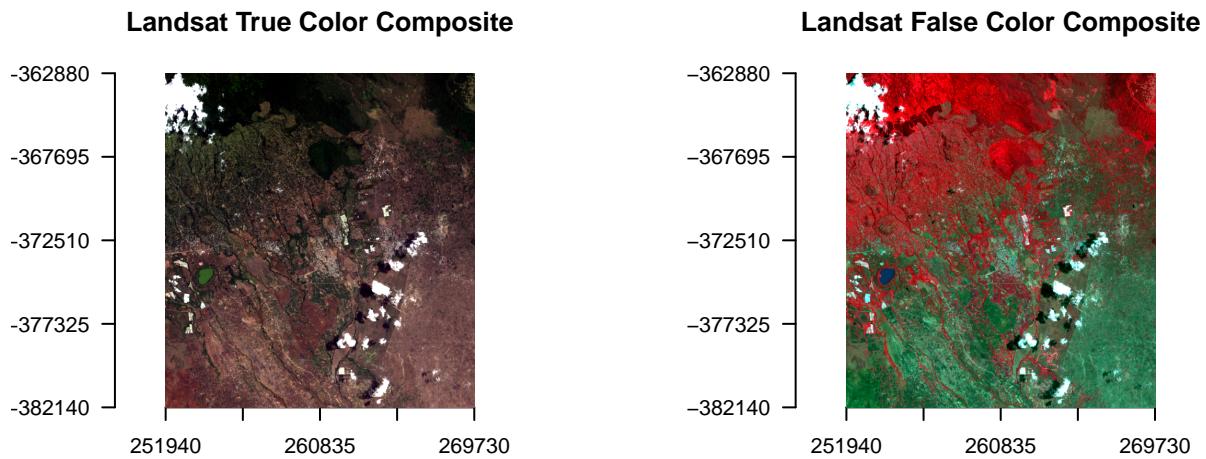
### Visualize single and multi-band imagery

To display 3-band color image, we use `plotRGB`. We have select the index of bands we want to render in the red, green and blue regions. For this Landsat image, r = 3 (red), g = 2(green), b = 1(blue) will plot the true color composite (vegetation in green, water blue etc). Selecting r = 4 (NIR), g = 3 (red), b = 2(green) will plot the false color composite (very popular in remote sensing with vegetation as red). You can find more about the visualization here.

```

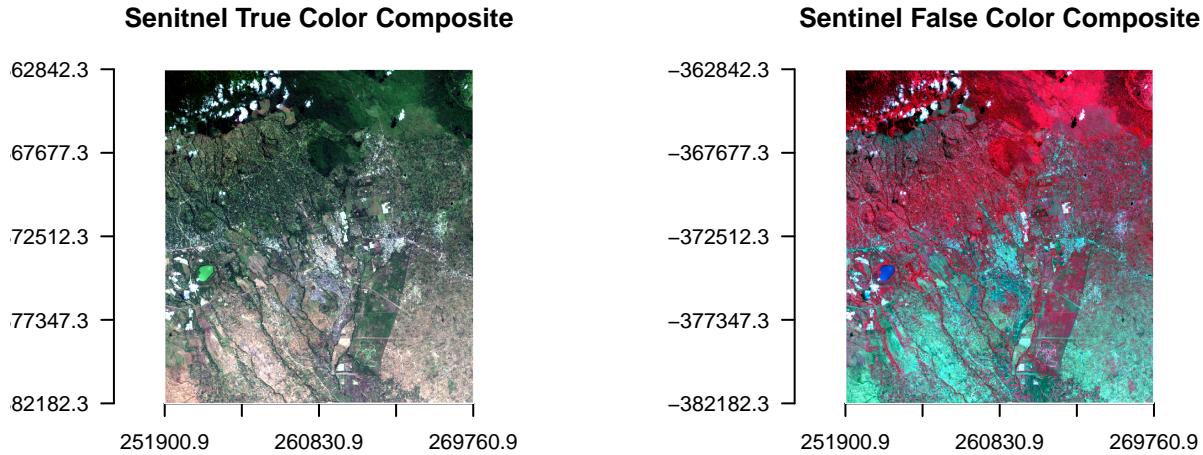
> nf <- layout(matrix(c(1,0,2), 1, 3, byrow = TRUE), width = c(1,0.2,1), respect = TRUE)
> plotRGB(r, r = 3, g = 2, b = 1, axes = TRUE, stretch = "lin", main = "Landsat True Color Composite")
> plotRGB(r, r = 4, g = 3, b = 2, axes = TRUE, stretch = "lin", main = "Landsat False Color Composite")

```



```
> dev.off()
null device
  1
```

```
> nf <- layout(matrix(c(1,0,2), 1, 3, byrow = TRUE), width = c(1,0.2,1), respect = TRUE)
> plotRGB(s, r = 3, g = 2, b = 1, axes = TRUE, stretch = "lin", main = "Sentinel True Color Composite")
> plotRGB(s, r = 7, g = 3, b = 2, axes = TRUE, stretch = "lin", main = "Sentinel False Color Composite")
```



```
> dev.off()
null device
  1
```

You can see the effect of differences in spatial resolution. Sentinel image has 10 m pixel size compared to 30 m pixels of Landsat.

You can also supply additional visualization arguments to `plotRGB` function to achieve desirable visualization effects.

### Subset and rename spectral bands

You can select specific bands using `subset` function.

```
> # select first 3 bands only
> rsub <- subset(r, 1:3)
> # Number of bands in orginal and new data
> nlayers(r)
[1] 6
> nlayers(rsub)
[1] 3
```

Set the names of the bands using the following:

```

> # For LANDSAT
> names(r)
[1] "landsat8.2016march.1" "landsat8.2016march.2" "landsat8.2016march.3"
[4] "landsat8.2016march.4" "landsat8.2016march.5" "landsat8.2016march.6"
> names(r) <- c('blue','green','red','NIR','SWIR1','SWIR2')
>
> # For SENTINEL
> names(s)
[1] "sentinel.1"   "sentinel.2"   "sentinel.3"   "sentinel.4"   "sentinel.5"
[6] "sentinel.6"   "sentinel.7"   "sentinel.8"   "sentinel.9"   "sentinel.10"
> names(s) <- c('blue','green','red','rededge1','rededge2','rededge3','NIR','rededge4','SWIR1','SWIR2')

```

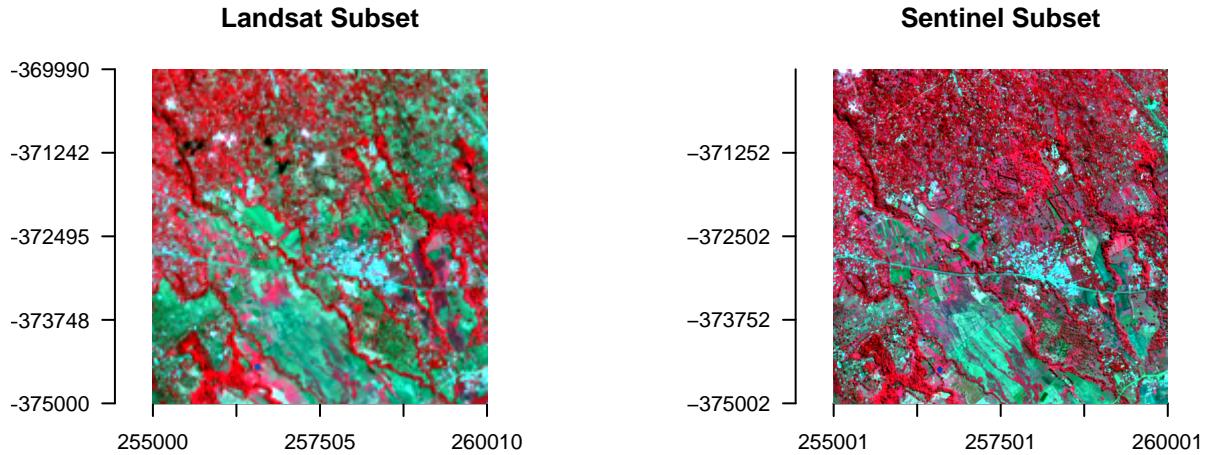
## Spatial subset/crop

Spatial subsetting can be used to limit analysis to a geographic subset of the image. Spatial subsets can be selected using the following methods: using `extent` object, using another `spatial` object from which an Extent can be extracted/created or entering row and column numbers.

```

> # Using extent
> extent(r)
class      : Extent
xmin      : 251940
xmax      : 269730
ymin      : -382140
ymax      : -362880
> e <- extent(255000, 260000, -375000, -370000)
>
> # crop LANDSAT
> rr <- crop(r, e)
>
> # crop SENTINEL
> ss <- crop(s, e)
>
> # Now plot the subsets side-by-side to compare the resolution
> nf <- layout(matrix(c(1,0,2), 1, 3, byrow = TRUE), width = c(1,0.2,1), respect = TRUE)
> plotRGB(rr, r = 4, g = 3, b = 2, axes = TRUE, stretch = "lin", main = "Landsat Subset")
> plotRGB(ss, r = 7, g = 3, b = 2, axes = TRUE, stretch = "lin", main = "Sentinel Subset")

```



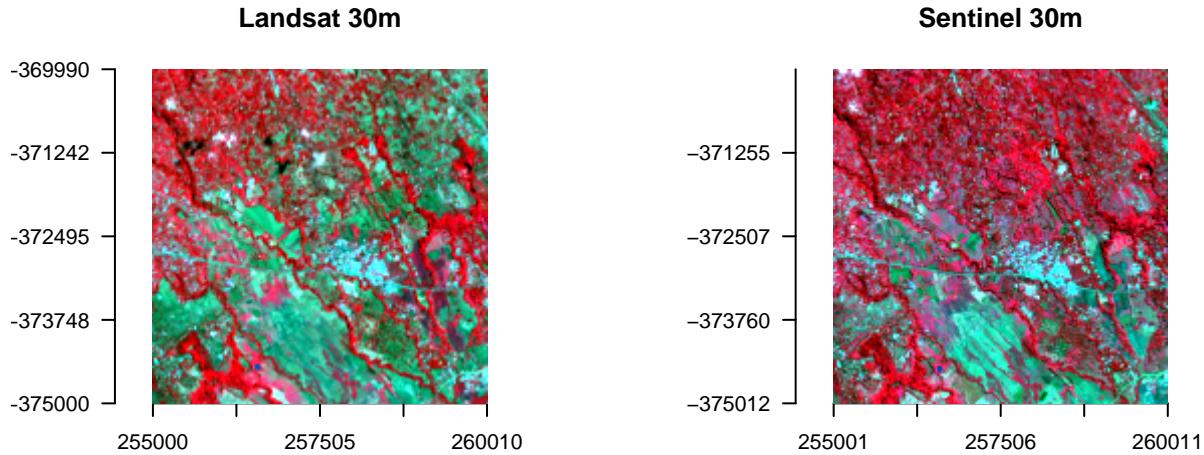
```
> dev.off()
null device
  1
```

**Ex 1** Interactive selection from the image is also possible. Use `drawExtent` and `drawPoly` to select area of your interests. Also use the `boundary` file to crop both Landsat and Sentinel data and plot them.

### Change spatial resolution or pixel size

`aggregate` creates a lower resolution (larger cells) image from finer pixels. Aggregation groups rectangular areas to create larger cells. Reverse process is known as `disaggregate` which creates higher resolution (smaller cells) from larger cells. These methods are particularly useful to compare (or analyze) data from different sources; for example, MODIS data 250 m with Landsat 30 m.

```
> ssa <- aggregate(ss, fact=3)
>
> nf <- layout(matrix(c(1,0,2), 1, 3, byrow = TRUE), width = c(1,0.2,1), respect = TRUE)
> plotRGB(rr, r = 4, g = 3, b = 2, axes = TRUE, stretch = "lin", main = "Landsat 30m")
> plotRGB(ssa, r = 7, g = 3, b = 2, axes = TRUE, stretch = "lin", main = "Sentinel 30m")
```



```
> dev.off()
null device
  1
```

### Extract raster values

Often we require value(s) of raster cell(s) for a geographic location/area. `extract` function is used to get raster values at the locations of other spatial data. You can use coordinates (points), lines, polygons or an Extent (rectangle) object. You can also use cell numbers to extract values. If using points, `extract` returns the values of a Raster\* object for the cells in which a set of points fall.

```
> # extract values with points
> samp <- readRDS('samples.rds')
> df <- extract(s,samp)
>
> # To see the reflectance values
> head(df)
      blue    green     red rededge1 rededge2 rededge3      NIR
[1,] 1531.706 1466.710 1399.800 1428.046 2236.057 2477.101 2626.426
[2,] 2735.996 2643.311 2737.977 2908.017 3597.897 3957.831 3979.245
[3,] 2487.741 2424.093 2505.525 2699.732 3583.802 3971.218 3929.050
[4,] 2262.449 2195.070 2176.647 2424.051 3416.316 3811.576 3733.057
[5,] 2299.131 2219.972 2194.162 2381.674 3268.081 3622.779 3632.741
[6,] 1962.716 2012.093 2078.173 2367.000 3553.000 4057.000 3810.069
rededge4     SWIR1     SWIR2
```

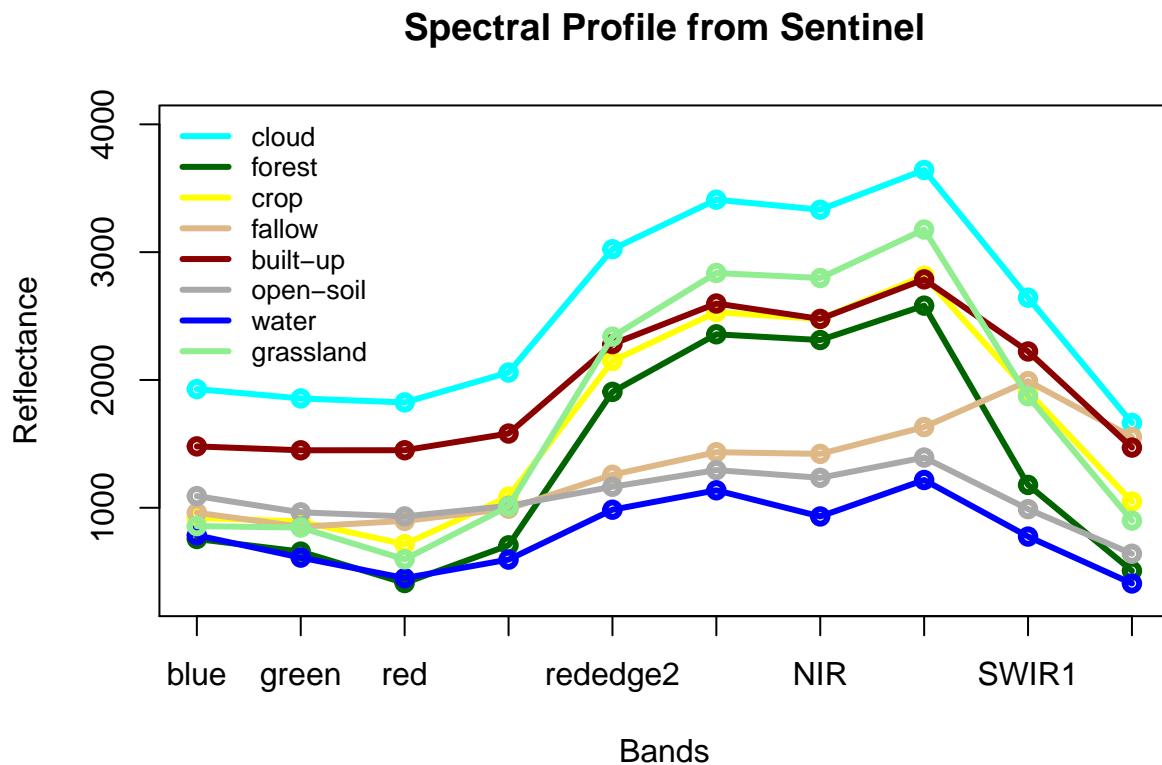
```
[1,] 2698.097 1831.076 1083.050
[2,] 4163.326 3667.964 2560.216
[3,] 4230.438 3371.761 2221.030
[4,] 4067.465 3254.127 2198.960
[5,] 3860.959 2988.901 1980.681
[6,] 4398.000 3109.000 1950.000
```

**Ex 2** Explore interactive options to extract values of a Raster objects. For example draw your own line or polygon boundaries and extract values for cells falling within the boundaries.

### Plot spectra

Plot of the spectrum (all bands) for the pixel/area is also known as spectral profiles. These profiles demonstrate the differences in spectral properties of various Earth surface features and constitute basis for numerous techniques for remote sensing 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: cloud, forest, crop, fallow, built-up, open-soil, water and grassland. In the next example, we will plot the mean spectra of these features.

```
> df <- round(df)
>
> # create an empty data.frame to store the mean spectra
> ms <- matrix(NA, nrow = length(unique(samp$id)), ncol = nlayers(ss))
>
> for (i in unique(samp$id)){
+   x <- df[samp$id==i,]
+   ms[i,] <- colMeans(x)
+ }
>
> # Specify the row- and column names
> rownames(ms) <- unique(samp$class)
> colnames(ms) <- names(ss)
>
> # We will create a vector of color for the land use land cover classes and will reuse it for other p
> mycolor <- c('cyan', 'darkgreen', 'yellow', 'burlywood', 'darkred', 'darkgray', 'blue', 'lightgreen')
>
>
> # First create an empty plot
> plot(1, ylim=c(300, 4000), xlim = c(1,10), xlab = "Bands", ylab = "Reflectance", xaxt='n')
>
> # Custom X-axis
> axis(1, at=1:10, lab=colnames(ms))
> # add the other spectra
> for (i in 1:nrow(ms)){
+   lines(ms[i,], type = "o", lwd = 3, lty = 1, col = mycolor[i])
+ }
>
> # Title
> title(main="Spectral Profile from Sentinel", font.main = 2)
>
> # Finally the legend
> legend("topleft", rownames(ms),
+        cex=0.8, col=mycolor, lty = 1, lwd =3, bty = "n")
```



Above example shows use of various plot parameters. The spectral profile shows (dis)similarity between different features. Clouds are generally bright and highly reflective in all wavelengths. Crop and forest show similar spectral feature (also see built-up and open areas). Water shows relatively low reflection.

**Ex 3** Make similar spectral plot using Landsat data (`rr`).

#### Write raster data

You often need to save an output raster resulting from any analysis to the disk and that can be accomplished by the function called `writeRaster`.

```
> writeRaster(ss, filename="sentinel-subset-area.tif", format="GTiff", overwrite=TRUE)
```

Be careful with the use of `overwrite`. Using `compress` can significantly reduce the filesize. GeoTiff doesn't preserve band/layer names. Use '`grd`' to save the associated band/layer names.

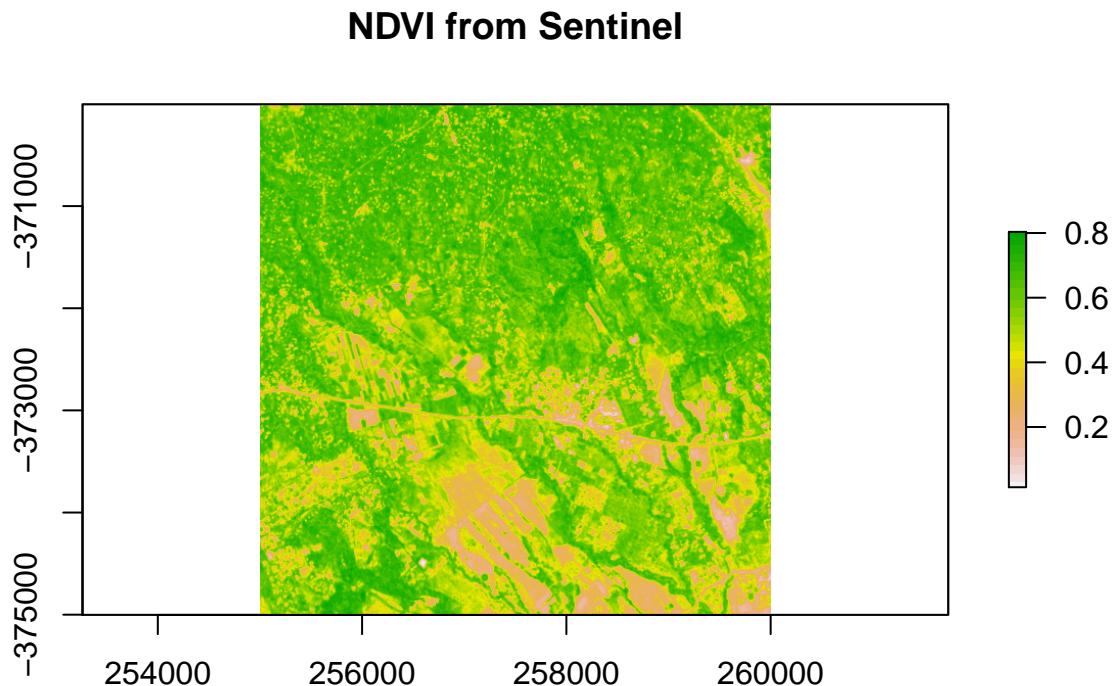
#### Basic mathematical operations

`raster` package supports many mathematical operations. Math operations are generally performed per pixel. First we will learn about basic arithmetic operations on bands. First example is a custom math function that calculates the Normalized Difference Vegetation Index (NDVI). Learn more about [vegetation indices] (<http://www.un-spider.org/links-and-resources/data-sources/daotm/daotm-vegetation>) and NDVI.

## Compute vegetation indices

Let's define a general function for ratio based vegetation index.

```
> # i and k are the index of bands to be used for the indices computation
>
> vi <- function(img, i, k){
+ bi <- img[[i]]
+ bk <- img[[k]]
+ vi <- (bk-bi)/(bk+bi)
+ return(vi)
+ }
>
> # For Sentinel NIR = 7, red = 3.
>
> ndvi <- vi(ss, 3,7)
> plot(ndvi, col = rev(terrain.colors(30)), main = 'NDVI from Sentinel')
```



You can see the variation in greenness from the plot.

**Ex 4** Repeat this steps for Landsat (For the current Landsat data, band 3 is red and band 4 is NIR)'

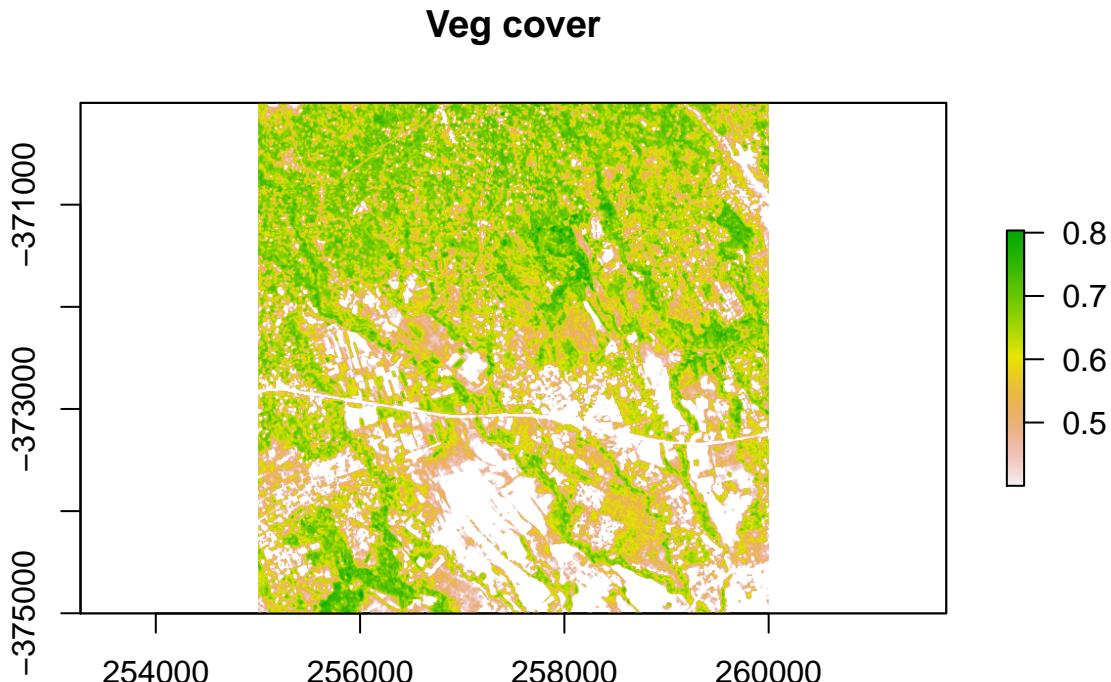
**Ex 5** Adapt the function to compute indices which will highlight i) water and ii) built-up water. Hint: Use the spectral profile plot to find the bands having maximum and minimum reflectance.

## 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.

Pixels having NDVI values greater than 0.4 are definitely vegetation. Following operation masks all non-vegetation pixels.

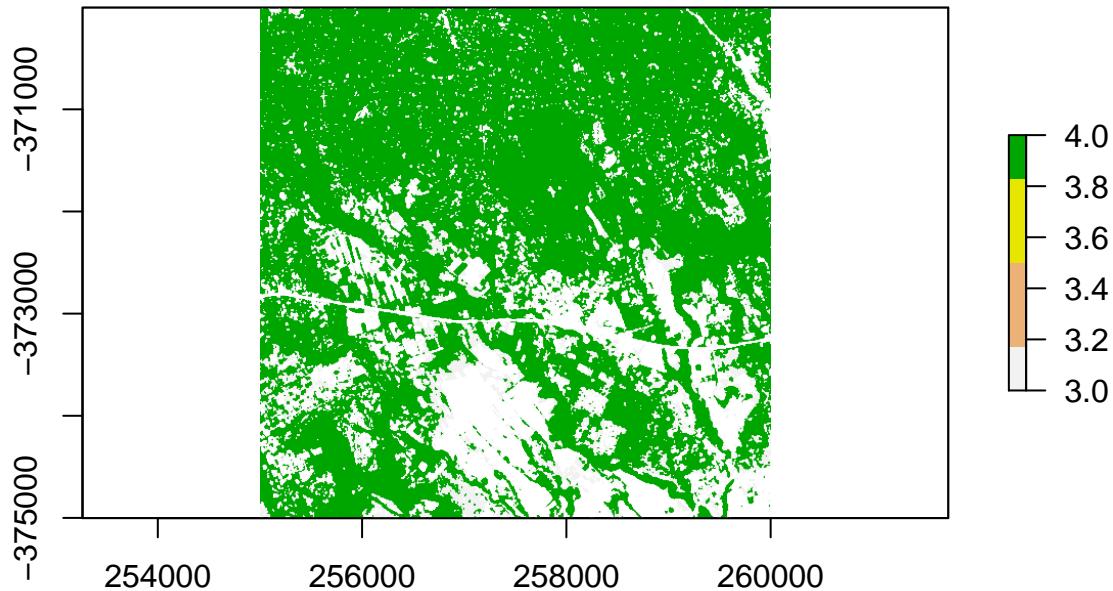
```
> veg <- calc(ndvi, function(x){x[x < 0.4] <- NA; return(x)})  
> plot(veg, main = 'Veg cover')
```



You can also create classes for different density of vegetation, 0 being no-vegetation.

```
> vegc <- reclassify(veg, c(-Inf,0.2,0, 0.2,0.3,1, 0.3,0.4,2, 0.4,0.5,3, 0.5, Inf, 4))  
> plot(vegc,col = rev(terrain.colors(4)), main = 'NDVI based thresholding')
```

## NDVI based thresholding

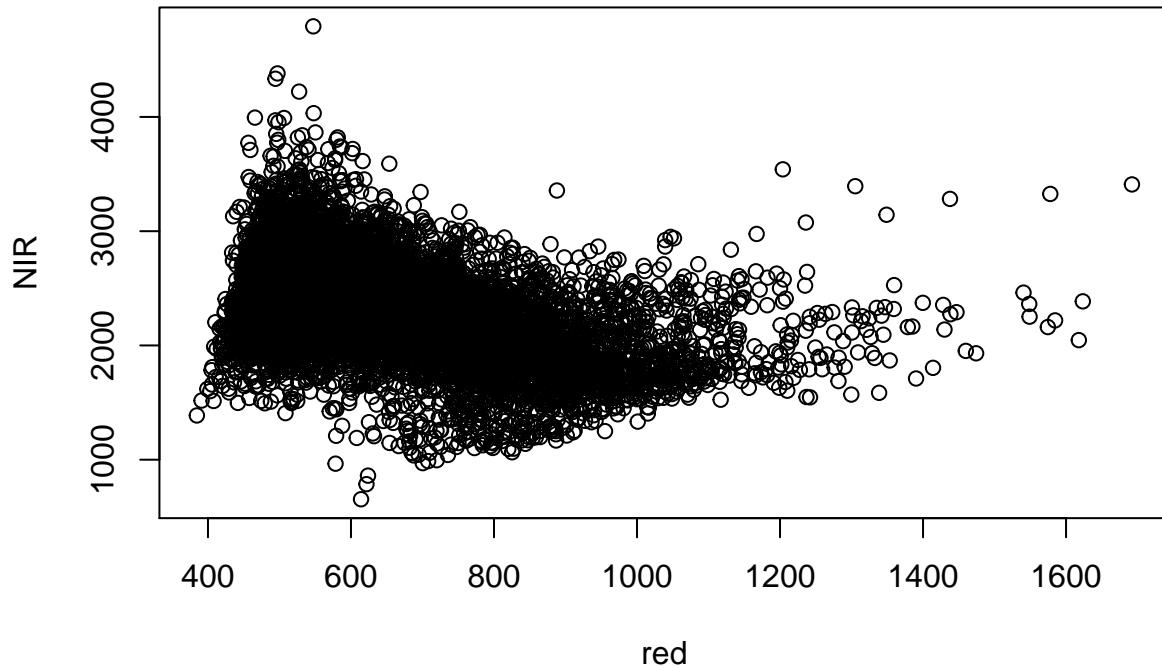


**Ex 6** Find water, cloud and built-up features using thresholding.

### Principal component analysis

The principal components (PC) transform (also known as the Karhunen-Loeve transform) is a spectral transformation which takes spectrally correlated image bands and generates uncorrelated bands. This process also helps to reduce the dimensionality and noise in the data. You can calculate the same number of principal components as the number of input bands. The first PC band explains the largest percentage of variance and other bands explain the variance in decreasing order. Last few bands appear noisy because they contain variance from the noise in original bands.

```
> set.seed(1)
> sr <- sampleRandom(ss, 10000)
> plot(sr[,c(3,7)])
```



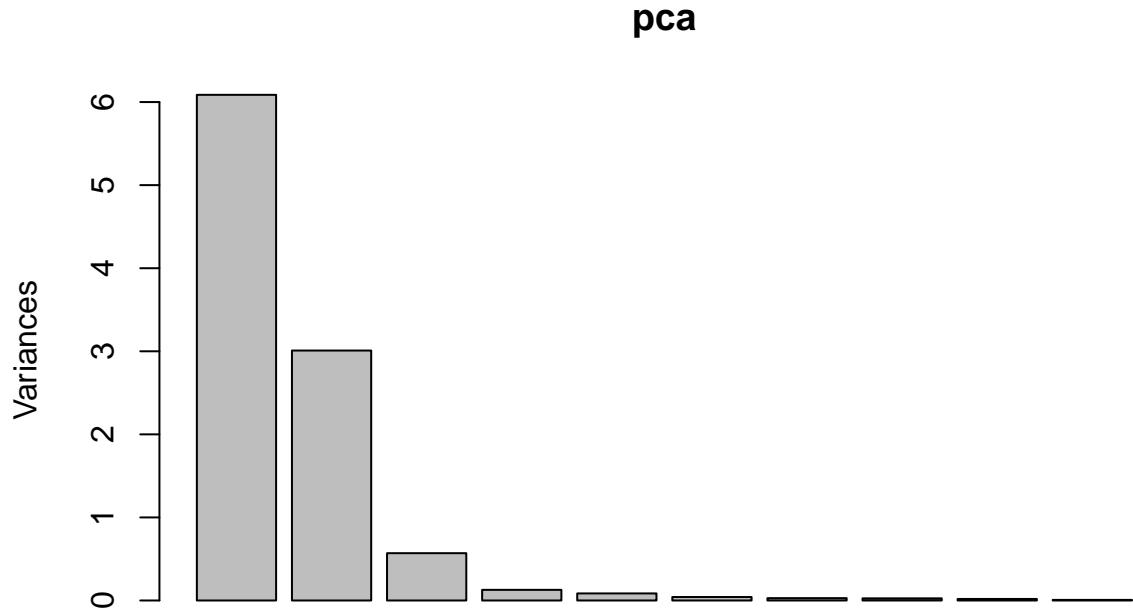
```

>
> # This is known as vegetation and soil-line plot. Can you guess the directions of 'principal component'
>
> pca <- prcomp(sr, scale = TRUE)
> pca
Standard deviations:
[1] 2.46718768 1.73462142 0.75463824 0.35782252 0.29161689 0.20465741
[7] 0.16810992 0.16053714 0.13572464 0.08472482

Rotation:
          PC1        PC2        PC3        PC4        PC5
blue      0.3397900 -0.1927183 -0.517510399 -0.35137577 -0.20051905
green     0.3168016 -0.3034097 -0.401064897  0.10887651  0.21281386
red       0.3805443 -0.1486060 -0.186778648 -0.11685673 -0.11072305
rededge1  0.3210704 -0.3076936  0.126436188  0.74109404  0.16396219
rededge2 -0.2509966 -0.4427533 -0.010637943  0.23521235 -0.22165289
rededge3 -0.2921145 -0.3927221 -0.027889327 -0.05493350 -0.31546205
NIR      -0.2754878 -0.3958728 -0.005447105 -0.31166346  0.78722555
rededge4 -0.2890050 -0.3964942  0.034374101 -0.08948546 -0.31671179
SWIR1    0.3058531 -0.2649557  0.594645623 -0.21606434 -0.07865375
SWIR2    0.3674040 -0.1402051  0.405896435 -0.30271605 -0.02227181
          PC6        PC7        PC8        PC9        PC10
blue     0.418361230  0.43460333 -0.11223178  0.19743930 -0.02069227
green    0.163741396 -0.58900910  0.18119621 -0.42550518  0.04418968
red      -0.800331121 -0.08713177  0.10527775  0.33595706 -0.01511829
rededge1 -0.007627127  0.45758270  0.01200449 -0.02139775 -0.01031098

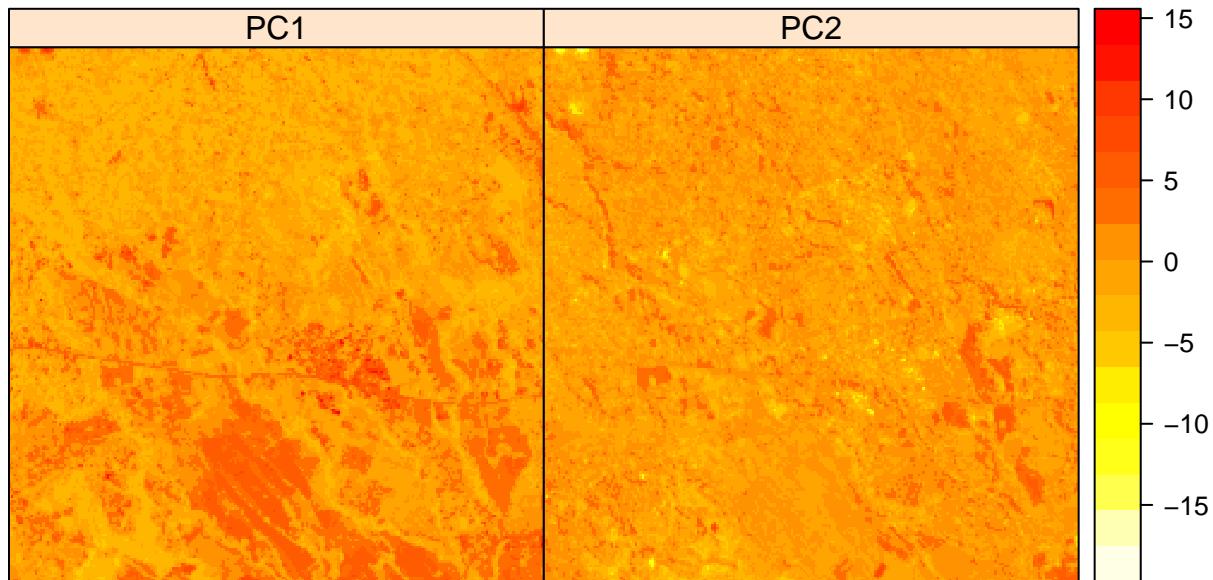
```

```
rededge2  0.126338483 -0.38148372 -0.50927098  0.45669100  0.08359020  
rededge3 -0.150689315  0.11457871  0.04093835 -0.35511730 -0.70250041  
NIR      -0.083787102  0.15645215 -0.02764396  0.13374798 -0.01879166  
rededge4 -0.092491834  0.17954615  0.34198944 -0.21531708  0.66758003  
SWIR1    0.301195608 -0.17563161  0.44240419  0.29176975 -0.16541595  
SWIR2    -0.112303455  0.03155452 -0.60734598 -0.42729794  0.15300794  
> plot(pca)  
> screeplot(pca)
```



```
>  
> pci <- predict(ss, pca, index = 1:2)  
> spplot(pci, col.regions = rev(heat.colors(20)), main = list(label="First 2 principal components from S
```

## First 2 principal components from Sentinel



To learn more about the information contained in the vegetation and soil line plots read this paper by [Gitelson et al] ([http://www.tandfonline.com/doi/abs/10.1080/01431160110107806#V6hp\\_LgrKhd](http://www.tandfonline.com/doi/abs/10.1080/01431160110107806#V6hp_LgrKhd)). An extension of PCA in remote sensing is known as Tasseled-cap Transformation.

## Image classification

We will explore two classification methods: unsupervised and supervised. Various unsupervised and supervised classification algorithms may be used. Different choice of classifier may produce different results. We will explore two k-means (unsupervised) and decision tree (supervised) algorithms.

### Unsupervised classification

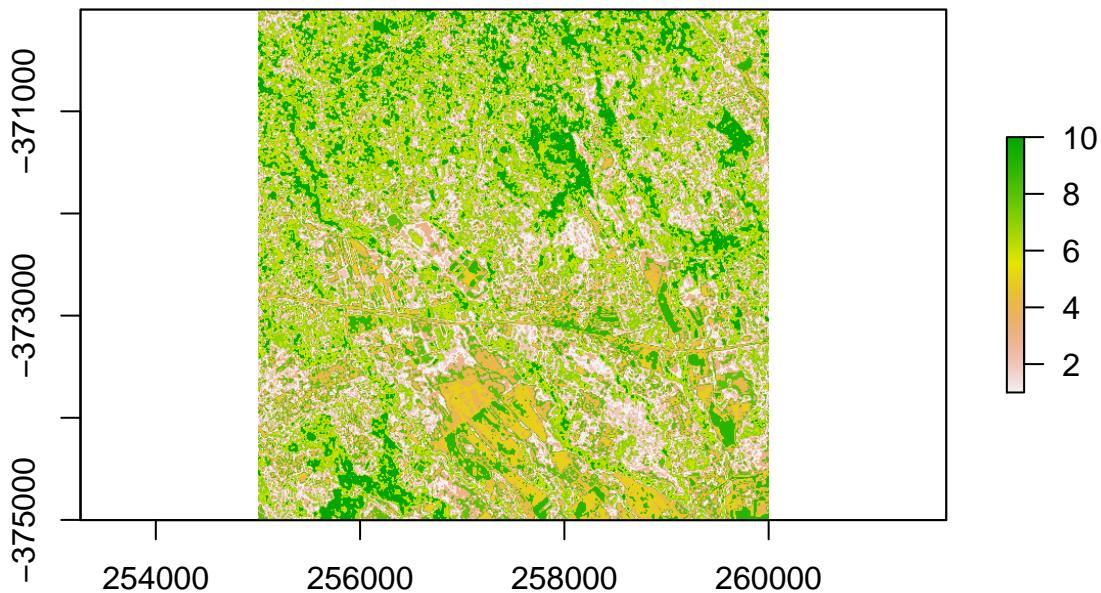
In unsupervised classification, we don't supply any training data. This particularly is useful when we don't have prior knowledge of the study area. The algorithm groups pixels with similar spectral characteristics into unique clusters/classes/groups following some statistically determined conditions. You have to re-label and combine these spectral clusters into information classes (for e.g. land-use land-cover).

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

We will perform unsupervised classification of the ndvi layer generated previously.

```
> nr <- getValues(ndvi)
> nr.km <- kmeans(na.omit(nr), centers = 10, iter.max = 500, nstart = 3, algorithm="Lloyd")
> knr <- ndvi
> knr[] <- nr.km$cluster
> plot(knr, main = 'Unsupervised classification of Sentinel data')
```

## Unsupervised classification of Sentinel data



To perform unsupervised classification, a good practice is to start with large number of centers (more clusters) and merge/group/recode similar clusters by inspecting the original imagery. Unsupervised algorithms are often referred as clustering.

## Supervised classification

In a supervised classification, we have prior knowledge about some of the land-use and land-cover types through a combination of fieldwork, interpretation of high resolution imagery and personal experience. 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 image is classified using the trained algorithm.

To collect training data interactively, you can use Google Earth (drawing point or polygon) or desktop GIS software. We will use a training site already collected through interpretation of high resolution image. The following example uses a Classification and Regression Trees (CART) classifier (Breiman et al. 1984) to predict different LULC around Arusha. In remote sensing literature CART is better known as decision tree classifiers when used for image classification. Read this article to know more.

```
> library(rpart)
>
> # For training the model, we will use the data used for plotting the spectra
>
> df <- data.frame(samp$class, df)
>
> # Train the model
> model.class <- rpart(as.factor(samp.class)~., data = df, method = 'class')
```

```

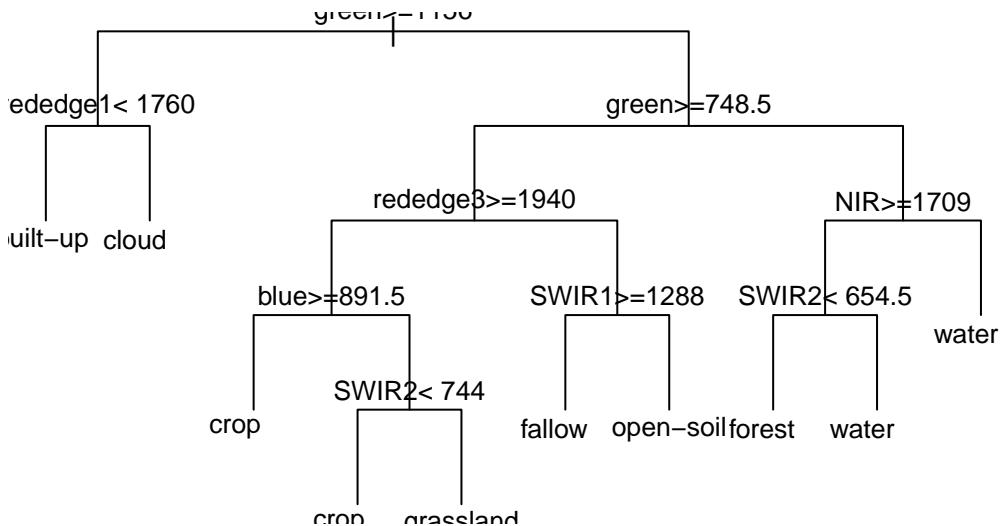
>
> # Print the trained classification tree
> model.class
n= 405

node), split, n, loss, yval, (yprob)
  * denotes terminal node

1) root 405 336 grassland (0.16 0.091 0.13 0.12 0.12 0.17 0.086 0.12)
  2) green>=1156 101 37 built-up (0.63 0.37 0 0 0 0 0 0)
    4) rededge1< 1760 73 11 built-up (0.85 0.15 0 0 0 0 0 0) *
    5) rededge1>=1760 28 2 cloud (0.071 0.93 0 0 0 0 0 0) *
  3) green< 1156 304 235 grassland (0 0 0.17 0.16 0.16 0.23 0.12 0.16)
    6) green>=748.5 207 138 grassland (0 0 0.26 0.24 0.0048 0.33 0.17 0)
      12) rededge3>=1940.5 123 54 grassland (0 0 0.42 0.0081 0.0081 0.56 0 0)
        24) blue>=891.5 41 1 crop (0 0 0.98 0.024 0 0 0 0) *
        25) blue< 891.5 82 13 grassland (0 0 0.15 0 0.012 0.84 0 0)
          50) SWIR2< 744 7 1 crop (0 0 0.86 0 0.14 0 0 0) *
          51) SWIR2>=744 75 6 grassland (0 0 0.08 0 0 0.92 0 0) *
  13) rededge3< 1940.5 84 36 fallow (0 0 0.012 0.57 0 0 0.42 0)
    26) SWIR1>=1288.5 51 3 fallow (0 0 0.02 0.94 0 0 0.039 0) *
    27) SWIR1< 1288.5 33 0 open-soil (0 0 0 0 0 0 1 0) *
  7) green< 748.5 97 48 water (0 0 0 0 0.49 0 0 0.51)
  14) NIR>=1709 52 7 forest (0 0 0 0 0.87 0 0 0.13)
    28) SWIR2< 654.5 45 0 forest (0 0 0 0 1 0 0 0) *
    29) SWIR2>=654.5 7 0 water (0 0 0 0 0 0 0 1) *
  15) NIR< 1709 45 3 water (0 0 0 0 0.067 0 0 0.93) *
>
> # Much cleaner way is to plot the trained classification tree
> plot(model.class, uniform=TRUE, main="Classification Tree")
> text(model.class, cex=.8)

```

## Classification Tree



```

>
> # Print some model specific parameters
> printcp(model.class)

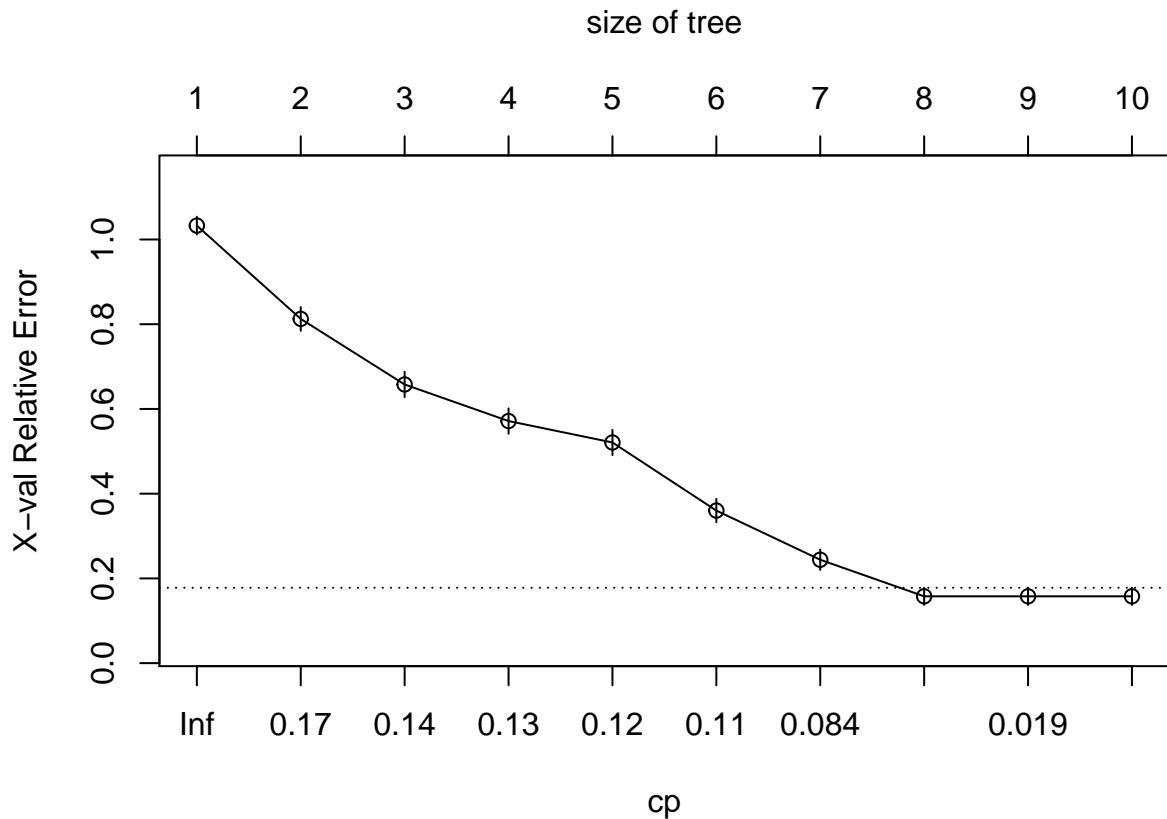
Classification tree:
rpart(formula = as.factor(samp.class) ~ ., data = df, method = "class")

Variables actually used in tree construction:
[1] blue      green     NIR       rededge1  rededge3  SWIR1     SWIR2

Root node error: 336/405 = 0.82963

n= 405

      CP nsplit rel.error xerror      xstd
1 0.190476      0 1.000000 1.03274 0.020980
2 0.145833      1 0.809524 0.81250 0.028074
3 0.142857      2 0.663690 0.65774 0.029822
4 0.119048      3 0.520833 0.57143 0.029907
5 0.113095      4 0.401786 0.52083 0.029670
6 0.098214      5 0.288690 0.36012 0.027415
7 0.071429      6 0.190476 0.24405 0.024068
8 0.020833      7 0.119048 0.15774 0.020200
9 0.017857      8 0.098214 0.15774 0.020200
10 0.010000      9 0.080357 0.15774 0.020200
> plotcp(model.class)
  
```



```

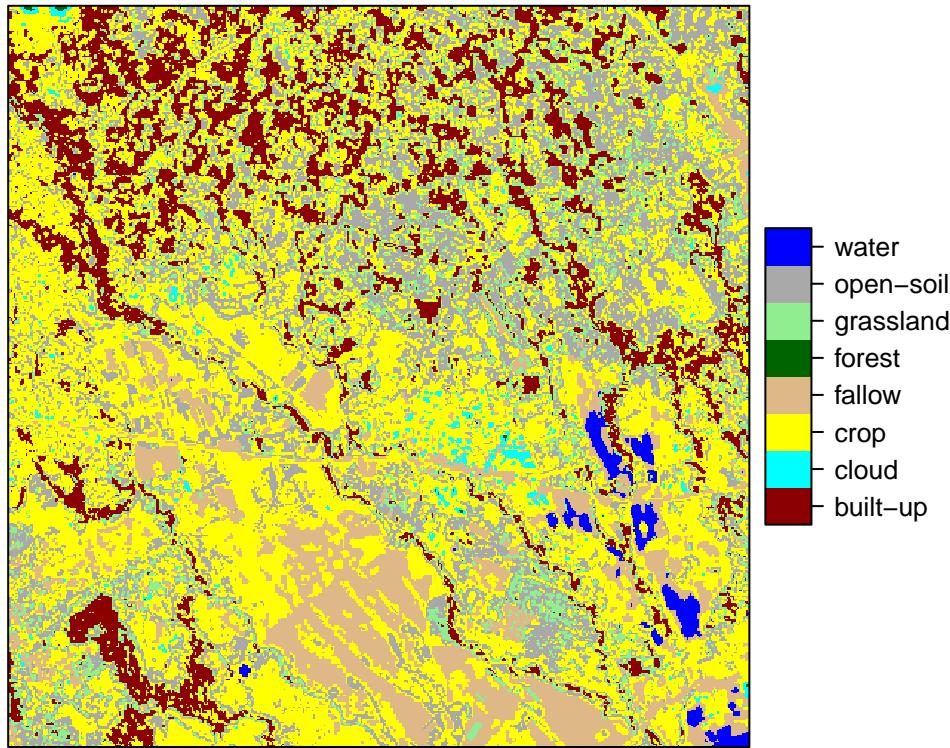
>
> # Now predict the subset data based on the model; prediction for entire area takes longer time
> pr <- predict(ss, model.class, type='class', progress = 'text')

|-----| 0%
|=====| 25%
|=====| 50%
|=====| 75%
|=====| 100%

>
> # Now plot the classification result
> library(rasterVis)
> pr <- ratify(pr)
> rat <- levels(pr)[[1]]
> rat$legend <- c("cloud","forest","crop","fallow","built-up","open-soil","water","grassland")
> levels(pr) <- rat
> levelplot(pr, maxpixels = 1e6,
+             col.regions = c("darkred","cyan","yellow","burlywood","darkgreen","lightgreen","darkgrey","black"),
+             scales=list(draw=FALSE),
+             main = "Supervised Classification of Sentinel data")

```

## Supervised Classification of Sentinel data



You can see the result is not satisfactory. For example, cropland is over-predicted; buildings with high reflection are wrongly labeled as cloud. One possible reason is, the training data is not adequate. Also land use land covers have complex patterns with lot intermixing. So you have carefully select large number of samples. Also choice of classifier plays an important role.

**Ex 7** Repeat the supervised classification with `randomForest` classifier. You can also collect additional samples.

Accuracy assessment of the classified map is required to test the quality of the product. Two widely reported measures in remote sensing community are overall accuracy and kappa statistics value. You can perform the accuracy assessment using hold-out samples.

### Resources:

- 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

### Using packages developed for analyzing remote sensing

Remote sensing R packages:

- RStoolbox
- landsat
- MODISTools

-hsdar

-Raster visualization: rasterVis