Land Use Land Cover classification

CIAT

06 November, 2016, 20:16

Objectives

This manual will help you conduct a land use land cover classification of a 6-band Landsat 8 image. At the end of this session, you will be able to:

1. Import a Landsat image into R
2. Import training data in the form of a shapefile into R
3. Extract pixel data to train and fit a Random forests model
4. Speed up image classification through parallel processing (Bonus)

For more details on source of Landsat data see (http://earthexplorer.usgs.gov/).

Before you start this session, it is important you have (i) the latest R software and (ii) Rstudio installed in your computer.

First, clear your workspace.

```
#clear your workspace
rm(list = ls(all = TRUE))
```

Set the start of spatial data processing.

```
#set the start of spatial data processing
startTime <- Sys.time()
cat("Start time", format(startTime),"\n")
```

```
## Start time 2016-11-06 20:16:35
```

Loading the data in R

Set your working directory. This is where you will save all outputs.

```
#set your working directory
setwd("C:/LDN_Workshop/Sample_dataset/Land_Use_Land_Cover")
```

List down the packages to be used in this session. Packages will be installed if not already installed. They will then be loaded into the session.

```
#load packages
.packages = c("rgdal","raster","caret")
.inst <- .packages %in% installed.packages()
if(length(.packages[!.inst]) > 0) install.packages(.packages[!.inst])
lapply(.packages, require, character.only=TRUE)
```

```
## Loading required package: rgdal
```

```
## Loading required package: sp

## rgdal: version: 1.1-10, (SVN revision 622)
##  Geospatial Data Abstraction Library extensions to R successfully loaded
##  Loaded GDAL runtime: GDAL 2.0.1, released 2015/09/15
##  Path to GDAL shared files: C:/Users/jymutua/Documents/R/win-library/3.3/rgdal/gdal
##  Loaded PROJ.4 runtime: Rel. 4.9.2, 08 September 2015, [PJ_VERSION: 492]
##  Path to PROJ.4 shared files: C:/Users/jymutua/Documents/R/win-library/3.3/rgdal/proj
##  Linking to sp version: 1.2-3

## Loading required package: raster

## Loading required package: caret

## Loading required package: lattice

## Loading required package: ggplot2

## [[1]]
## [1] TRUE
##
## [[2]]
## [1] TRUE
##
## [[3]]
## [1] TRUE
```

If you need to reproduce the results next time, set the seed. You can use any number.

```r
#set random seed
set.seed(322)
```

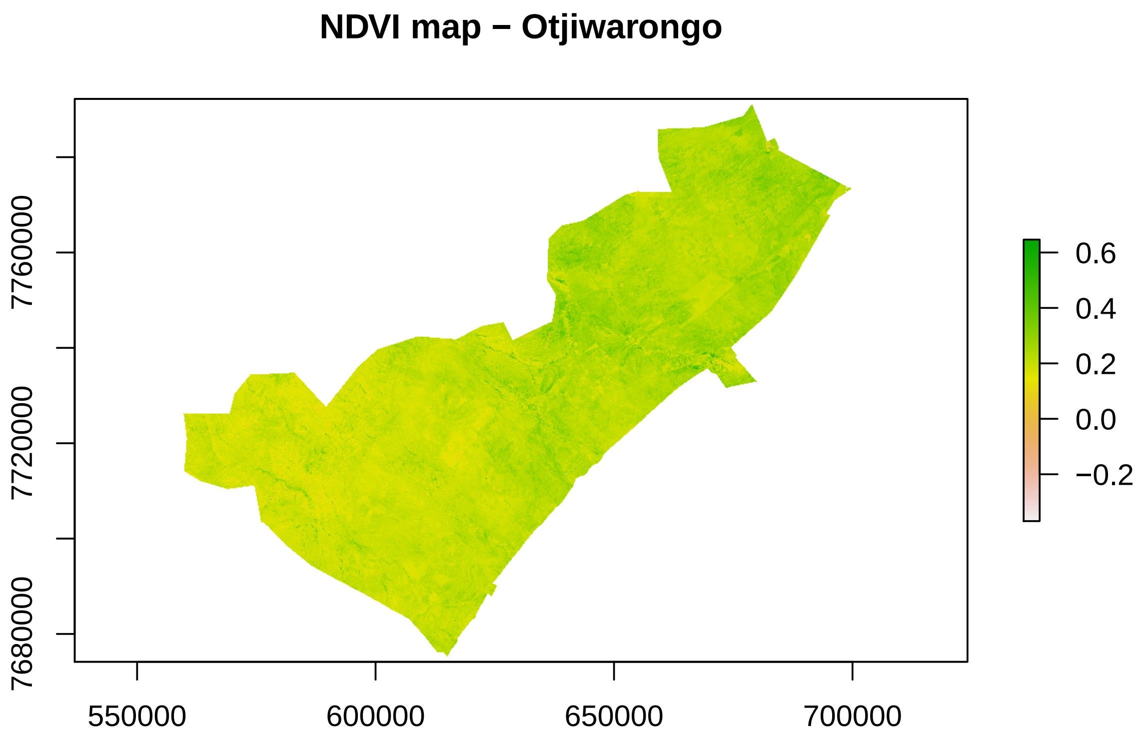
Import the image into R as a 'RasterBrick' object using the brick function from the 'raster' package. Also let's replace the original band names with shorter ones (e.g. 'B1' to 'B7').

```r
#import the image into R
img <- brick("TOA/Otji_TOA.tif")
names(img) <- c(paste0("B", 2:7, coll = ""))
img
```

```
## class       : RasterBrick
## dimensions  : 3936, 4713, 18550368, 6  (nrow, ncol, ncell, nlayers)
## resolution  : 30, 30  (x, y)
## extent      : 559815, 701205, 7674145, 7792225  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=utm +zone=33 +south +datum=WGS84 +units=m +no_defs +ellps=WGS84 +towgs84
## data source : C:\LDN_Workshop\Sample_dataset\Land_Use_Land_Cover\TOA\Otji_TOA.tif
## names       :    B2,    B3,    B4,    B5,    B6,    B7
## min values  :   696,   382,   239,     1,    44,    46
## max values  :  4751,  6510,  6164,  7215, 11800,  9882
```

Before we begin the actual land cover classification, let's calculate NDVI and save the output for use in the next session. The resulting NDVI can be plotted as a map, values range from 0.2 (bare soils) to 0.6 (dense vegetation) in the area.

```r
#calculate NDVI and save the output for use in the next session
NDVI <- (img[[4]] - img[[3]]) / (img[[4]] + img[[3]])
writeRaster(NDVI, "Otji_NDVI.tif", overwrite=TRUE)
plot(NDVI, main="NDVI map - Otjiwarongo")
```



We can make a natural colour visualization of the Landsat image in R using the 'plotRGB' command, for example, a natural colour composite 4:3:2 (Red - Green - Blue).We use the expression 'img * (img >= 0)' to convert the negative values to zero.

```r
#make a natural colour visualization of the Landsat image
plotRGB(img * (img >= 0), r = 4, g = 3, b = 2, scale = 10000)
```

Import the training data into R as an object of class 'SpatialPolygonsDataFrame' and create a variable to store the name of the 'class' column. Codes used in the training data include: 1-Forest, 2-Woodland, 31-Bushland, 32-Grassland, 42-Cultivated area, 51-Wetland, 52-Water body, 61-Artificial Surface, 71-Bareland, forest and Woodland classes later combined to Forest/woodland.

```
#import the training data into R
trainData <- shapefile("Training_data/Otji_trainingData.shp")
responseCol <- "LC_Code"
```

The training dataset ('Otji_trainingData.shp') stores the ID for each land cover type in a column in the attribute table called 'LC_Code' as shown below:

Plot the training data.

```
#plot the training data.
plot(trainData, main="Distribution of training data", axes=FALSE)
```

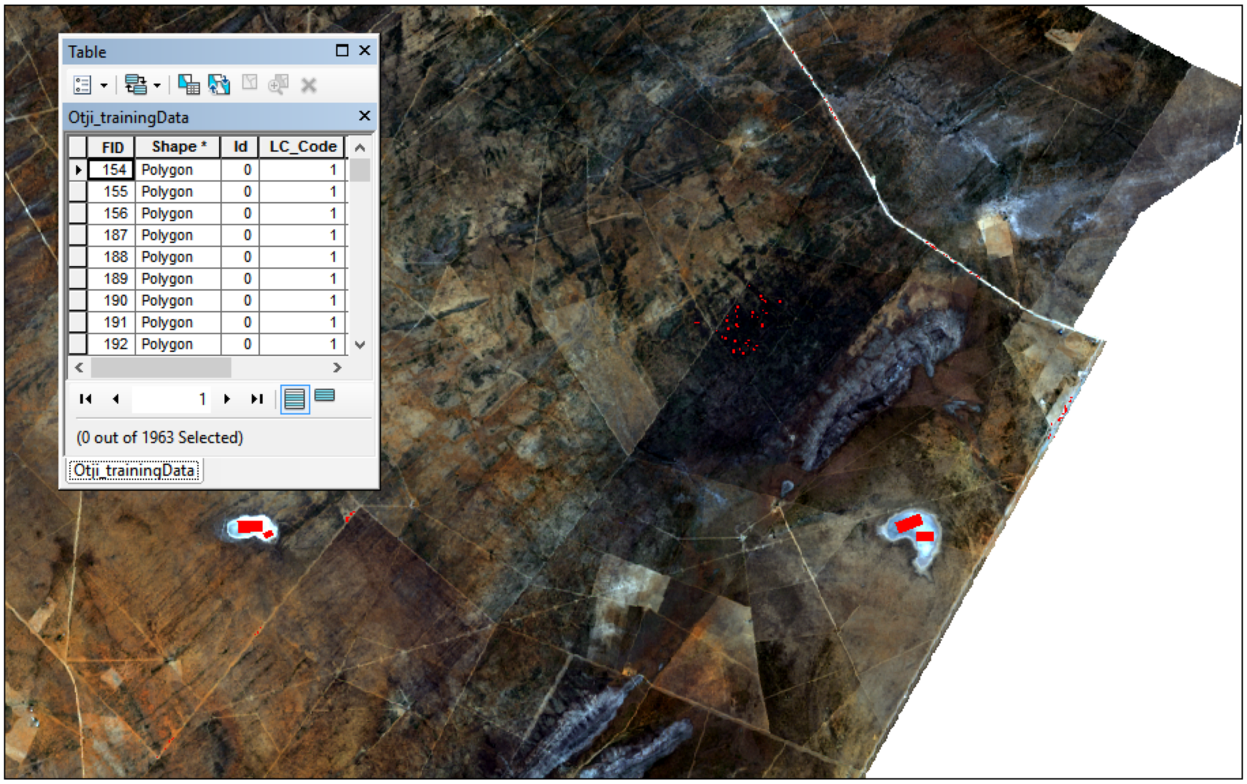


Figure 1: Training data as seen in ESRI ArcGIS

Extracting training pixels values

Extract the pixel values in the training areas for every band in the Landsat image and store them in a data frame along with the corresponding land cover class ID. The code below allows you to extract training data using both point and polygon shapefiles.

```
#extract the pixel values in the training areas
trainSet = data.frame(matrix(vector(), nrow = 0, ncol = length(names(img)) + 1))
for (i in 1:length(unique(trainData[[responseCol]]))){
  category <- unique(trainData[[responseCol]])[i]
  categorymap <- trainData[trainData[[responseCol]] == category,]
  dataSet <- extract(img, categorymap)

  if(is(trainData, "SpatialPointsDataFrame")){
    dataSet <- cbind(dataSet, class = as.numeric(category))
    trainSet <- rbind(trainSet, dataSet)
  }
  if(is(trainData, "SpatialPolygonsDataFrame")){
    dataSet <- lapply(dataSet, function(x){cbind(x, class =
                                                    as.numeric(rep(category,
                                                                   nrow(x))))})
    df <- do.call("rbind", dataSet)
    trainSet <- rbind(trainSet, df)
  }
}
```

Partition the data into training and testing, this will enable us conduct accuracy tests.

```
#partition the data into training and testing
inData <- createDataPartition(y = trainSet$class, p = 0.7, list = FALSE)
training <- trainSet[inData,]
testing <- trainSet[-inData,]
```

As you can see below, the training and testing 'data.frames' contains values for each of six 'Landsat' TOA bands plus the class attribute.

```
#how does the class look like
table(training$class)
```

```
##
##    1    2   31   32   52   71
##  225  392 2656  874    5 1855
```

```
table(testing$class)
```

```
##
##    1    2   31   32   52   71
##  111  152 1139  378    1  792
```

In our case, we will use 1000 observations which are randomly sample from the training data.frame to train the model.

```r
#randomly sample from the training data.frame
train_sample <- training[sample(1:nrow(training), 1000), ]
table(train_sample$class)
```

```
##
##   1   2  31  32  71
##  42  48 476 145 289
```

Fitting the Random Forests model

Define and fit the .RandomForests. model using the 'train' function from the 'caret' package by specifying the model as a formula with the dependent variable (i.e., the land cover types IDs) encoded as factors.

```r
#fit the trandom forest model
rf_Otji <- train(as.factor(class) ~ B3 + B4 + B5, method = "rf", data =
                   train_sample)
```

```
## Loading required package: randomForest

## randomForest 4.6-12

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin

## note: only 2 unique complexity parameters in default grid. Truncating the grid to 2 .
```

We can now use the 'predict' command to make a raster with predictions from the fitted model object (i.e., 'rf_Otji'). Speed up computations using the 'clusterR' function from the 'raster' package which supports multi-core computing for functions such as predict (NB: install 'snow' package).

```r
#cluster the predictions
beginCluster()
```

```
## 4 cores detected, using 3
```

```r
prediction <- clusterR(img, raster::predict, args = list(model = rf_Otji))
endCluster()
```

Test the accuracy using the 'testing' dataset as it is an independent set of data and let's examine the producer's accuracy (aka sensitivity in the caret package) for the model.

```r
#test accuracy of testing dataset
prediction_2 <- predict(rf_Otji, testing)
confusionMatrix(prediction_2, testing$class)$overall[1]
```

```
## Warning in levels(reference) != levels(data): longer object length is not a
```

```
## multiple of shorter object length

## Warning in confusionMatrix.default(prediction_2, testing$class): Levels are
## not in the same order for reference and data. Refactoring data to match.

##  Accuracy
## 0.9560824
```

```
confusionMatrix(prediction_2, testing$class)$byClass[, 1]
```

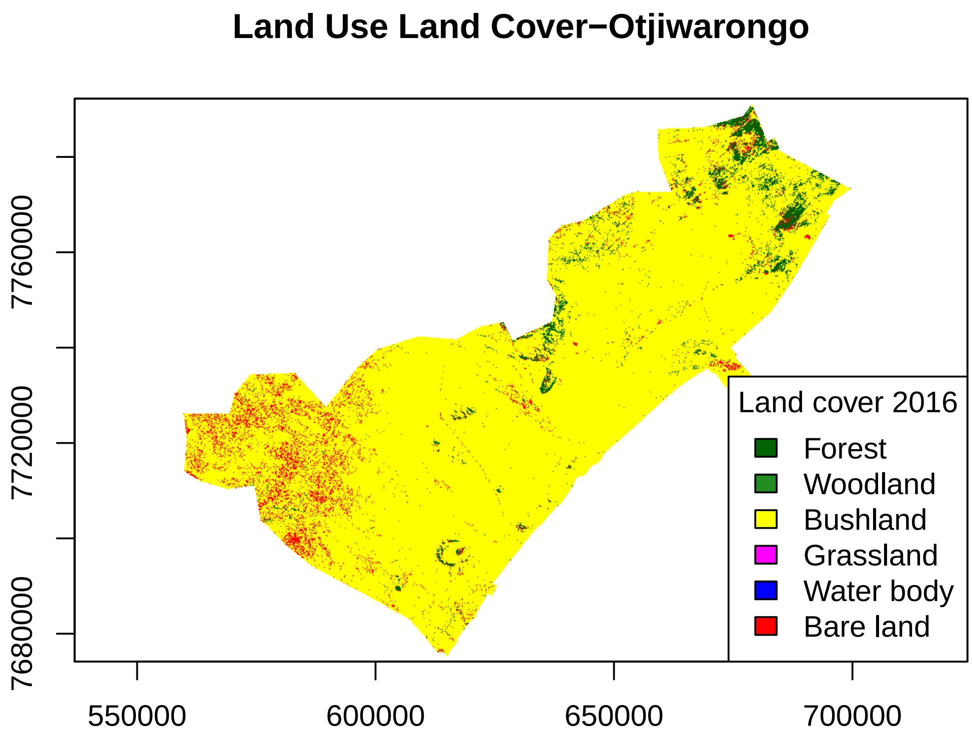
```
## Warning in levels(reference) != levels(data): longer object length is not a
## multiple of shorter object length

## Warning in levels(reference) != levels(data): Levels are not in the same
## order for reference and data. Refactoring data to match.

##  Class: 1  Class: 2 Class: 31 Class: 32 Class: 52 Class: 71
## 1.0000000 1.0000000 0.9446883 0.9206349 0.0000000 0.9760101
```

Save your classified image as a GeoTIFF.

```
#save your classified image as a GeoTIFF
writeRaster(prediction, "Otji_classified.tif", overwrite=TRUE)
```

Visualize your classified image and add a legend to the plot. Can you recall this categories 1-Forest, 2-Woodland, 31-Bushland, 32-Grassland, 42-Cultivated area, 51-Wetland, 52-Water body, 61-Artificial Surface, 71-Bareland?

```
#visualize your classified image
cols <- c("dark green", "forestgreen", "yellow", "magenta", "blue", "red")
plot(prediction, col=cols, legend=FALSE, main="Land Use Land Cover-Otjiwarongo")
legend("bottomright",
       legend=c("Forest", "Woodland", "Bushland", "Grassland", "Water body",
                "Bare land"), fill=cols, bg="white",
       title = "Land cover 2016")
```



Finally, check the amount of time you spent conducting this analysis.

```r
#check the amount of time you spent conducting this analysis
timeDiff <- Sys.time() - startTime
cat("\nProcessing time", format(timeDiff), "\n")
```

```
##
## Processing time 24.52455 mins
```