Chapter 4: Vector Data

Chris Holden

03/24/2015

Introduction

The **OGR** library is a companion library to **GDAL** that handles vector data capabilities, including information queryies, file conversions, rasterization of polygon features, polygonization of raster features, and much more. It handles popular formats including the *ESRI Shapefile*, *Keyhole Markup Language*, *PostGIS*, and *SpatiaLite*. For more information on how **OGR** came about and how it relates to **GDAL**, see here: http://trac.osgeo.org/gdal/wiki/FAQGeneral#WhatisthisOGRstuff.

We will be using **OGR** without necessarily knowing it, but it will be driving our input and output for vector data.

Dataset

I've digitized some polygons for our small subset study site that contain descriptions of the land cover within the polygon. These polygons will serve as training data for our future land cover classification.

To get the data, let's download it from Github:

```
if (file.exists('training_data.shp') == F) {
    download.file(url = 'https://raw.githubusercontent.com/ceholden/open-geo-tutorial/master/example/tra
                  destfile = 'training_data.zip', method = 'curl')
    unzip('training_data.zip')
}
```

To read this dataset into memory, use the `readOGR` function from `rgdal`:

```
library(rgdal)
```

```
## Loading required package: sp
## rgdal: version: 0.9-2, (SVN revision 526)
## Geospatial Data Abstraction Library extensions to R successfully loaded
## Loaded GDAL runtime: GDAL 1.11.2, released 2015/02/10
## Path to GDAL shared files: /usr/share/gdal/1.11
## Loaded PROJ.4 runtime: Rel. 4.8.0, 6 March 2012, [PJ_VERSION: 480]
## Path to PROJ.4 shared files: (autodetected)
```

```
training <- readOGR('training_data.shp', layer='training_data')
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "training_data.shp", layer: "training_data"
## with 30 features
## It has 2 fields
```

```
summary(training)
```

```
## Object of class SpatialPolygonsDataFrame
## Coordinates:
##         min       max
## x  462486.6  469601.8
## y 1734334.2 1741429.4
## Is projected: TRUE
## proj4string :
## [+proj=utm +zone=15 +datum=WGS84 +units=m +no_defs +ellps=WGS84
## +towgs84=0,0,0]
## Data attributes:
##        id               class
##  Min.   :1.000   barren    : 6
##  1st Qu.:1.000   forest    :11
##  Median :2.000   herbaceous: 6
##  Mean   :2.433   urban     : 2
##  3rd Qu.:3.750   water     : 5
##  Max.   :5.000
```

As you can see, this `training_data.shp` file contains two fields: "id", and "class". The ID is just an integer factor variable for each class label. The unique class labels within this dataset are:

```r
unique(training$class)
```

```
## [1] forest     water      urban      barren     herbaceous
## Levels: barren forest herbaceous urban water
```

The dataset read in through `readOGR` is stored as a `SpatialPolygonsDataFrame` class. There are other similar classes for `SpatialPoints*`, `SpatialLines*`, and `SpatialGrid*`. All of these classes behave like normal `DataFrame` classes, but they contain additional information about geolocation and projection.

Reprojection

A common tool needed for GIS work is reprojection which translates the coordinates of a dataset from one projection system to another. Let's check our dataset's projection:

```r
proj4string(training)
```

```
## [1] "+proj=utm +zone=15 +datum=WGS84 +units=m +no_defs +ellps=WGS84 +towgs84=0,0,0"
```

The projection is WGS84 UTM 15N already, which matches our Landsat data. The projection information that was printed out is formatted as a Proj4 string, an extremely common and simple way of defining projections. To find out projection parameters, definitions, and to translate between any of the many ways of describing a projection, visit http://spatialreference.org/.

If we wanted to reproject to latitude-longitude coordinates using WGS84, we could do so as such:

```r
training_wgs84 <- spTransform(training, CRS("+proj=longlat +datum=WGS84"))
proj4string(training_wgs84)
```

```
## [1] "+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"
```

As you can see, the coordinates are now listed in latitude/longitude.

Plotting

To visualize our training data, we can simply use the `plot` command:

```r
plot(training)
```

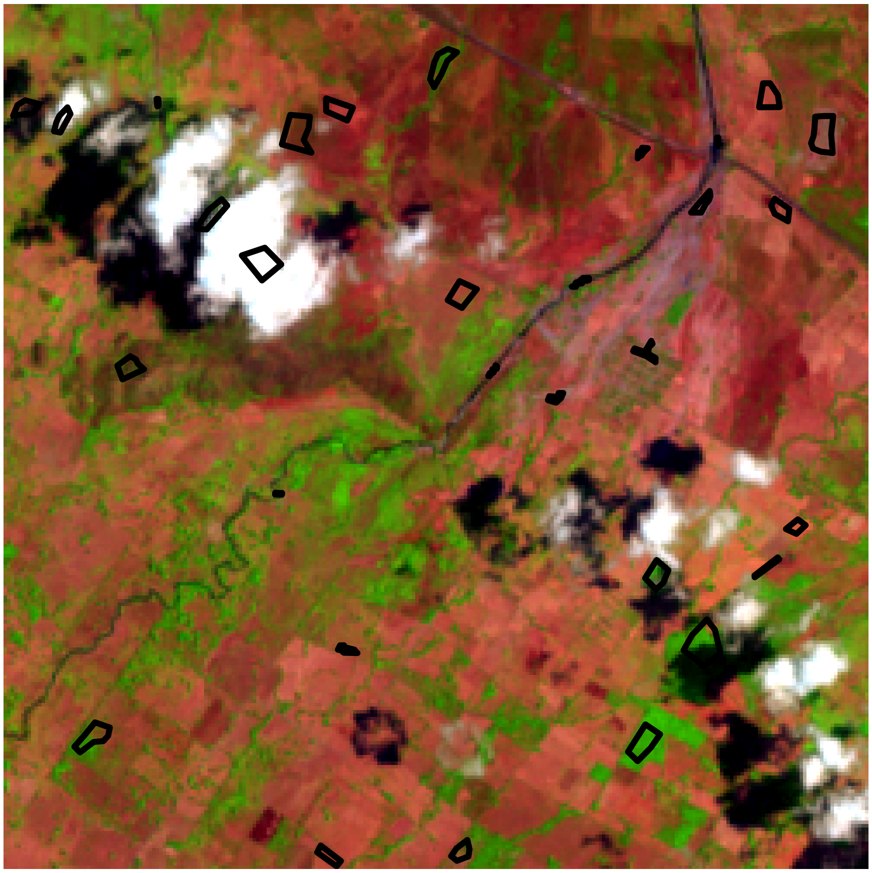
This created a boring, hard to reference plot. We can enhance it by drawing it on top of our remote sensing image. First, make sure we have it loaded in our workspace:

```r
library(raster)

if (file.exists('LE70220492002106EDC00_stack.gtif') == F) {
    download.file(url='https://raw.githubusercontent.com/ceholden/open-geo-tutorial/master/example/LE702
                  destfile='LE70220492002106EDC00_stack.gtif', method='curl')
}

le7 <- brick('LE70220492002106EDC00_stack.gtif')
proj4string(le7)
```

```
## [1] "+proj=utm +zone=15 +datum=WGS84 +units=m +no_defs +ellps=WGS84 +towgs84=0,0,0"
```

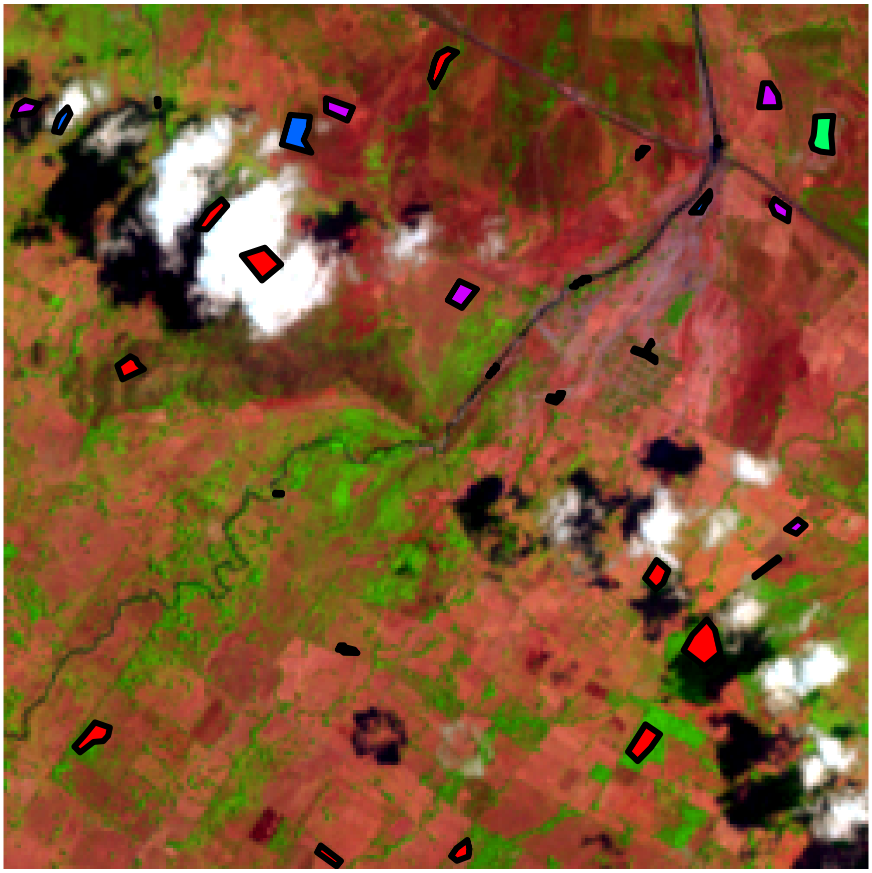
Remember that these two datasets are in the same projection already. GIS softwares like QGIS will perform "On the Fly Reprojection", but R will not do this.

```r
plotRGB(le7, r=5, g=4, b=3, stretch="lin")
plot(training, lwd=3, add=T)
```



We can add colors based on the class labels:

```
plotRGB(le7, r=5, g=4, b=3, stretch="lin")

classes <- unique(training$class)
cols <- rainbow(length(classes))
line_cols <- rep(cols[0], length(training))
for (i in 1:length(cols)) {
    line_cols[which(training$class == classes[i])] <- cols[i]
}

plot(training, lwd=3, col=line_cols, add=T)
```



Wow!

Vector and Raster

If we wanted to extract values from our Landsat 7 image within each of our training data polygons, we could use the aptly named `extract` function:

```
# Extract the region of interest (roi) data
roi_data <- extract(le7, training, df=TRUE)
names(roi_data)
```

```
## [1] "ID"                  "band.1.reflectance" "band.2.reflectance"
## [4] "band.3.reflectance" "band.4.reflectance" "band.5.reflectance"
## [7] "band.7.reflectance" "band.6.temperature" "Band.8"
```

```
dim(roi_data)
```

```
## [1] 718   9
```

```
head(roi_data)
```

```
##   ID band.1.reflectance band.2.reflectance band.3.reflectance
## 1  1                673                728                650
```

```
## 2  1               633              748              614
## 3  1               653              748              596
## 4  1               634              770              614
## 5  1               634              770              578
## 6  1               653              749              578
##   band.4.reflectance band.5.reflectance band.7.reflectance
## 1               3068               2007                986
## 2               3068               1784                799
## 3               3107               1807                775
## 4               2990               1650                705
## 5               3107               1673                728
## 6               3185               1739                752
##   band.6.temperature Band.8
## 1               2487      0
## 2               2487      0
## 3               2487      0
## 4               2537      0
## 5               2487      0
## 6               2487      0
```

We now have all of the information we need to facilitate comparison between our two datasets nicely contained within an easy to use, familiar R `DataFrame`.