EDS 223: week 9 lab

Ruth Oliver

2022-11-23

Overview

Monitoring the distribution and change in land cover types can help us understand the impacts of phenomena like climate change, natural disasters, deforestation, and urbanization. Determining land cover types over large areas is a major application of remote sensing because we are able to distinguish different materials based on their spectral reflectance.

Classifying remotely sensed imagery into landcover classes enables us to understand the distribution and change in landcover types over large areas. There are many approaches for performing landcover classification – *supervised* approaches use training data labeled by the user, whereas *unsupervised* approaches use algorithms to create groups which are identified by the user afterward.

credit: this lab is based on a materials developed by Chris Kibler.

Task

In this lab, we are using a form of supervised classification, a *decision tree classifier*. Decision trees classify pixels using a series of conditions based on values in spectral bands. These conditions (or decisions) are developed based on training data. In this lab we will create a land cover classification for southern Santa Barbara County based on multi-spectral imagery and data on the location of 4 land cover types:

- green vegetation
- dry grass or soil
- urban
- water

Summary

- load and process Landsat scene
- crop and mask Landsat data to study area
- extract spectral data at training sites

- train and apply decision tree classifier
- plot results

Data

Landsat 5 Thematic Mapper

- Landsat 5
- 1 scene from September 25, 2007

- bands: 1, 2, 3, 4, 5, 7
- Collection 2 surface reflectance product

Study area and training data

- polygon representing southern Santa Barbara county
- polygons representing training sites
  - type: character string with land cover type

Workflow

Process data

Load packages and set working directory We'll be working with vector and raster data, so will need both `sf` and `terra`. To train our classification algorithm and plot the results, we'll use the `rpart` and `rpart.plot` packages. Set your working directory to the folder that holds the data for this lab.

Note: my filepaths may look different than yours!

```
library(sf)
library(terra)
library(here)
library(dplyr)
library(rpart)
library(rpart.plot)
library(tmap)

rm(list = ls())

here::i_am("labs/week9.Rmd")
setwd(here())
```

Load Landsat data Let's create a raster stack based on the 6 bands we will be working with. Each file name ends with the band number (e.g. `B1.tif`). Notice that we are missing a file for band 6. Band 6 corresponds to thermal data, which we will not be working with for this lab. To create a raster stack, we will create a list of the files that we would like to work with and read them all in at once using the `rast` function. We'll then update the names of the layers to match the spectral bands and plot a true color image to see what we're working with.

```
# list files for each band, including the full file path
filelist <- list.files("./data/week9/landsat-data/", full.names = TRUE)

# read in and store as a raster stack
landsat_20070925 <- rast(filelist)

# update layer names to match band
names(landsat_20070925) <- c("blue", "green", "red", "NIR", "SWIR1", "SWIR2")

# plot true color image
plotRGB(landsat_20070925, r = 3, g = 2, b = 1, stretch = "lin")
```

Load study area We want to contstrain our analysis to the southern portion of the county where we have training data, so we'll read in a file that defines the area we would like to study.

```
# read in shapefile for southern portion of SB county
SB_county_south <- st_read("./data/week9/SB_county_south.shp")

# project to match the Landsat data
SB_county_south <- st_transform(SB_county_south, crs = crs(landsat_20070925))
```

Crop and mask Landsat data to study area Now, we can crop and mask the Landsat data to our study area. This reduces the amount of data we'll be working with and therefore saves computational time. We can also remove any objects we're no longer working with to save space.

```
# crop Landsat scene to the extent of the SB county shapefile
landsat_cropped <- crop(landsat_20070925, SB_county_south)

# mask the raster to southern portion of SB county
landsat_masked <- mask(landsat_cropped, SB_county_south)

# remove unnecessary object from environment
rm(landsat_20070925, SB_county_south, landsat_cropped)
```

Convert Landsat values to reflectance Now we need to convert the values in our raster stack to correspond to reflectance values. To do so, we need to remove erroneous values and apply any scaling factors to convert to reflectance.

In this case, we are working with Landsat Collection 2. The valid range of pixel values for this collection 7,273-43,636, with a multiplicative scale factor of 0.0000275 and an additive scale factor of -0.2. So we reclassify any erroneous values as `NA` and update the values for each pixel based on the scaling factors. Now the pixel values should range from 0-100%.

```r
# reclassify erroneous values as NA
rcl <- matrix(c(-Inf, 7273, NA,
                43636, Inf, NA), ncol = 3, byrow = TRUE)

landsat <- classify(landsat_masked, rcl = rcl)

# adjust values based on scaling factor
landsat <- (landsat * 0.0000275 - 0.2) * 100

# plot true color image to check results
plotRGB(landsat, r = 3, g = 2, b = 1, stretch = "lin")

# check values are 0 - 100
summary(landsat)
```

Classify image

Extract reflectance values for training data We will load the shapefile identifying different locations within our study area as containing one of our 4 land cover types. We can then extract the spectral values at each site to create a data frame that relates land cover types to their spectral reflectance.

```r
# read in and transform training data
training_data <- st_read("./data/week9/trainingdata.shp") %>%
  st_transform(., crs = crs(landsat))

# extract reflectance values at training sites
training_data_values <- extract(landsat, training_data, df = TRUE)

# convert training data to data frame
training_data_attributes <- training_data %>%
  st_drop_geometry()

# join training data attributes and extracted reflectance values
SB_training_data <- left_join(training_data_values, training_data_attributes,
                              by = c("ID" = "id")) %>%
  mutate(type = as.factor(type)) # convert landcover type to factor
```

Train decision tree classifier To train our decision tree, we first need to establish our model formula (i.e. what our response and predictor variables are). The `rpart` function implements the CART algorithm. The `rpart` function needs to know the model formula and training data you would like to use. Because we are performing a classification, we set `method = "class"`. We also set `na.action = na.omit` to remove any pixels with `NA`s from the analysis.

To understand how our decision tree will classify pixels, we can plot the results. The decision tree is comprised of a hierarchy of binary decisions. Each decision rule has 2 outcomes based on a conditional statement pertaining to values in each spectral band.

```r
# establish model formula
SB_formula <- type ~ red + green + blue + NIR + SWIR1 + SWIR2

# train decision tree
```

```
SB_decision_tree <- rpart(formula = SB_formula,
                          data = SB_training_data,
                          method = "class",
                          na.action = na.omit)

# plot decision tree
prp(SB_decision_tree)
```

Apply decision tree Now that we have created our decision tree, we can apply it to our entire image. The `terra` package includes a `predict()` function that allows us to apply a model to our data. In order for this to work properly, the names of the layers need to match the column names of the predictors we used to train our decision tree. The `predict()` function will return a raster layer with integer values. These integer values correspond to the *factor levels* in the training data. To figure out what category each integer corresponds to, we can inspect the levels of our training data.

```
# classify image based on decision tree
SB_classification <- predict(landsat, SB_decision_tree, type = "class", na.rm = TRUE)

# inspect level to understand the order of classes in prediction
levels(SB_training_data$type)
```

Plot results Now we can plot the results and check out our land cover map!

```
# plot results
tm_shape(SB_classification) +
  tm_raster(style = "cat",
            palette = c("#8DB580", "#F2DDA4", "#7E8987", "#6A8EAE"),
            labels = c("green vegetation", # update labels to match clasess
                       "soil/dead grass",
                       "urban",
                       "water"),
            title = "land cover") +
  tm_layout(legend.position = c("left", "bottom"))
```