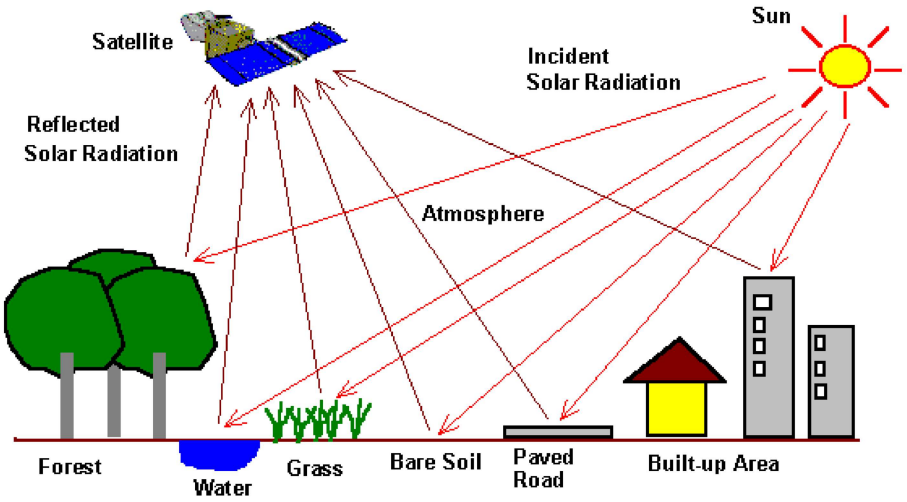
Classifying Satellite Imagery in R

Sydney Goldstein

Urban Spatial Analysis - Website (http://urbanspatialanalysis.com/) / Other Work (https://urbanspatial.github.io/ProjectsPage/)
Updated April 2021

Different materials reflect and absorb wavelengths differently. Remote sensing utilizes this phenomena to capture a picture of the Earth's surface by detecting how various forms of ground cover reflect solar radiation.

In the diagram below for example, the built-up area, bare soil, and forest will absorb the sun's radiation differently and reflect it in a way that makes them unique to the sensors on the satellite. We can then view these differences in the spectral bands of satellite imagery.



Source: Centre for Remote Imaging, Sensing & Processing (https://crisp.nus.edu.sg/~research/tutorial/optical.htm)

Raw satellite imagery, however, is not necessarily useful when performing various analyses. To get useful inputs about land cover in an area, we must transform the imagery. One way to do this is to classify the imagery into categories that we are interested in.

This tutorial introduces using rasters and classifying imagery in R. It is based on a similar tutorial (https://gfc.ucdavis.edu/events/arusha2016/_static/labs/day4/day4_lab1_remote-sensing.pdf) from UC Davis. We will cover:

- accessing the properties of a raster in R
- calculating Normalized Difference Vegetation Index (NDVI)
- supervised classification

For this tutorial, we use Landsat 8 imagery from Calgary, which can be found here (https://github.com/urbanSpatial/classifying_satellite_imagery_in_R/tree/master/data). However, the process can be repeated with any Landsat 8 imagery downloaded from either Earth Explorer (https://earthexplorer.usgs.gov/) or other sites (https://remotepixel.ca/projects/satellitesearch.html).

Loading packages and preparing the data

Let's begin by loading in the necessary libraries.

```
library(raster)
library(tidyverse)
library(sf)
library(rpart)
library(rpart.plot)
library(rasterVis)
library(mapedit)
library(mapview)
library(caret)
library(forcats)
```

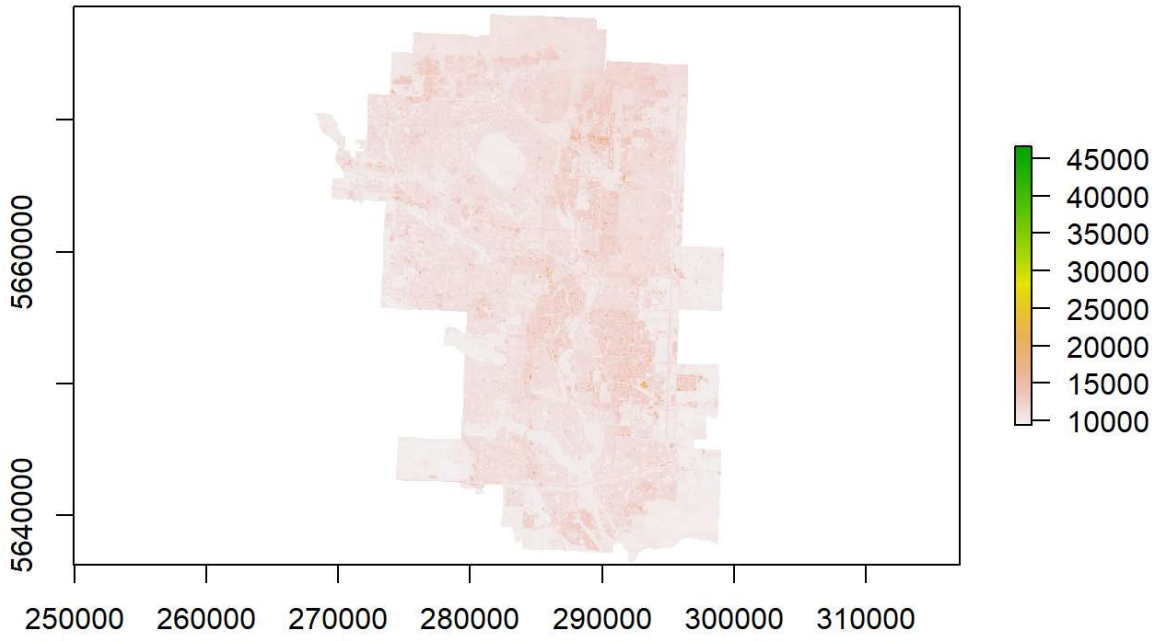
Next, we read in the different bands that comprise the satellite imagery. Each band refers to a different spectrum:

Band	Description
Band 1	Coastal aersol
Band 2	Blue
Band 3	Green
Band 4	Red
Band 5	Near Infrared (NIR)
Band 6	Shortwave Infrared (SWIR) 1
Band 7	Shortwave Infrared (SWIR) 2
Band 8	Panchromatic
Band 9	Cirrus
Band 10	Thermal Infrared (TIRS) 1
Band 11	Thermal Infrared (TIRS) 2

```
band1 <- raster("./data/band1.tif")
band2 <- raster("./data/band2.tif")
band3 <- raster("./data/band3.tif")
band4 <- raster("./data/band4.tif")
band5 <- raster("./data/band5.tif")
band6 <- raster("./data/band6.tif")
band7 <- raster("./data/band7.tif")
band8 <- raster("./data/band8.tif")
band9 <- raster("./data/band9.tif")
band10 <- raster("./data/band10.tif")
band11 <- raster("./data/band11.tif")
```

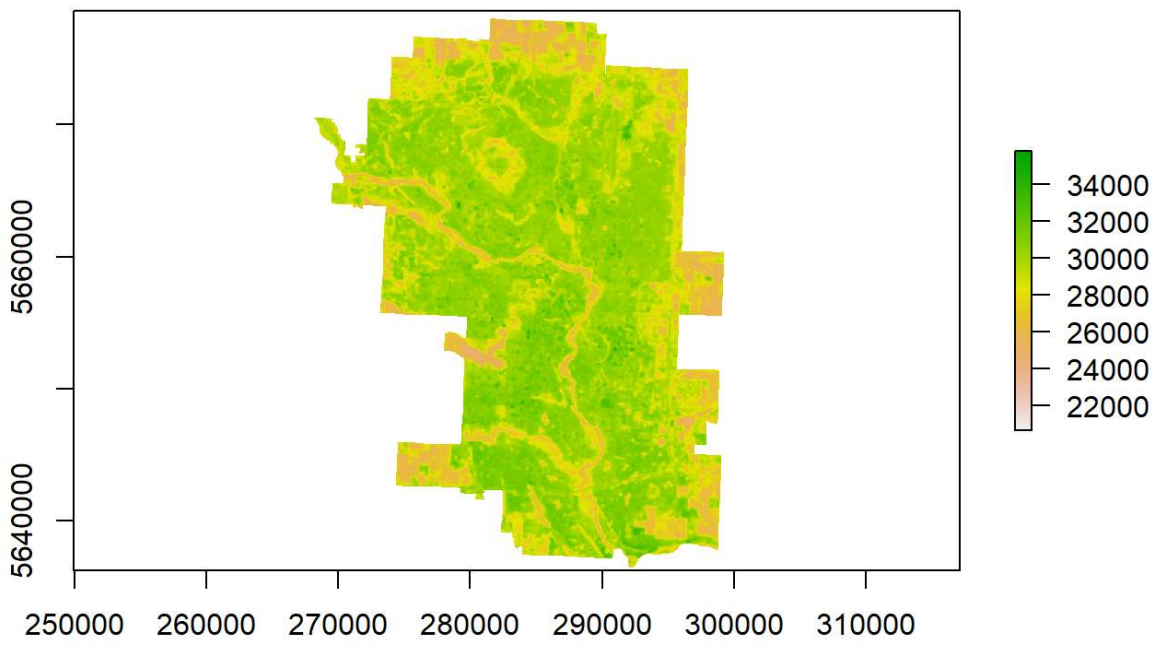
Let's take a look at what the rasters looks like. Below we plot `band1` using base R `plot` function.

```
plot(band1)
```



`band1` doesn't show much variation in the terrain. What about `band10` ?

```
plot(band10)
```



You'll notice that the range of values for each raster varies. `band1` values range from 10,000 to slightly above 40,000. Whereas values in `band10` range from a little less than 22,000 to about 34,000.

Right now, each raster is separate a object and they do not 'speak' to each other. To perform any analysis, we need to combine the individual bands into one multi-band raster. The `stack` function in the `raster` package will create one raster from the designated layers, similar to the composite bands function in ArcGIS.

Note that band 8, the Panchromatic image, is calculated at a different resolution (15 meters rather than 30 meters) than the other bands.

```
res(band8)
```

```
## [1] 15 15
```

If we try to `stack` this band with the others as is, it will return an error because the grid cells do not overlay correctly. We `aggregate` the cell size to 30 meters prior to stacking the rasters into one image. `fact` in the aggregate function is the multiplier. In this case, we are multiplying 15 meters times 2 to get 30 meters, the resolution of the other rasters in the image.

```
band8 <- aggregate(band8, fact = 2)

image <- stack(band1, band2, band3, band4, band5, band6, band7,
               band8, band9, band10, band11)
```

Exploring the imagery

There are several properties of the raster we can access in R that would normally be found in the properties window in ArcGIS. Below, we look at the number of layers (or bands) the raster has, the coordinate system the imagery is projected in, and the resolution (or grid cell size) of the raster.

```
nlayers(image)
```

```
## [1] 11
```

```
crs(image)
```

```
## CRS arguments:
##  +proj=utm +zone=12 +datum=WGS84 +units=m +no_defs
```

```
res(image)
```

```
## [1] 30 30
```

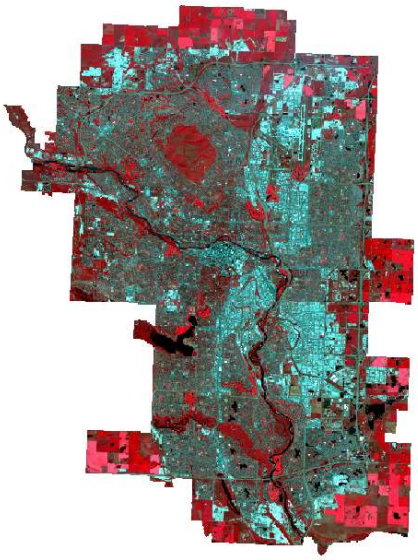
Now that we know a little more about the imagery we are using, let's plot it. Since `image` is a multi-band raster, we use the `plotRGB` function from the `raster` package, which allows us to specify what bands should be visualized.

There are two main composites that are normally used in remote sensing: the true color composite and the false color composite. As the name suggests, the true color composite plots the imagery as it would appear to the human eye. It uses the red band (4) for red, the green band (3) for green, and the blue band (2) for blue.

```
par(col.axis="white",col.lab="white",tck=0)
plotRGB(image, r = 4, g = 3, b = 2, axes = TRUE,
        stretch = "lin", main = "True Color Composite")
box(col="white")
```

The false color composite is popular in remote sensing because vegetation will appear red and areas of more developed land will appear in blue. The false color composite uses NIR (5) for red, red (4) for green, and green (3) for blue.

```
par(col.axis="white",col.lab="white",tck=0)
plotRGB(image, r = 5, g = 4, b = 3, axes = TRUE, stretch = "lin", main = "False Col
or Composite")
box(col="white")
```



Calculating Normalized Difference Vegetation Index

NDVI provides another way to identify vegetation and is calculated on a scale of -1 to 1, where values closer to 1 indicate more vegetative cover. The calculation is based on how pigments in vegetation absorb sunlight compared to other ground cover.

We calculate NDVI using the following equation `(NIR - Red)/(NIR + Red)`. The `[[ ]]` notation specifies the bands, in this case band 5 for NIR and band 4 for Red, within the multi-band raster.

```
ndvi <- (image[[5]] - image[[4]])/(image[[5]] + image[[4]])
```

Let's take a look at the range of values - they should be between -1 and 1. Pay close attention to the annotation that is used to pull out data from rasters.

```
#minimum
min(ndvi@data@values, na.rm = T)
```

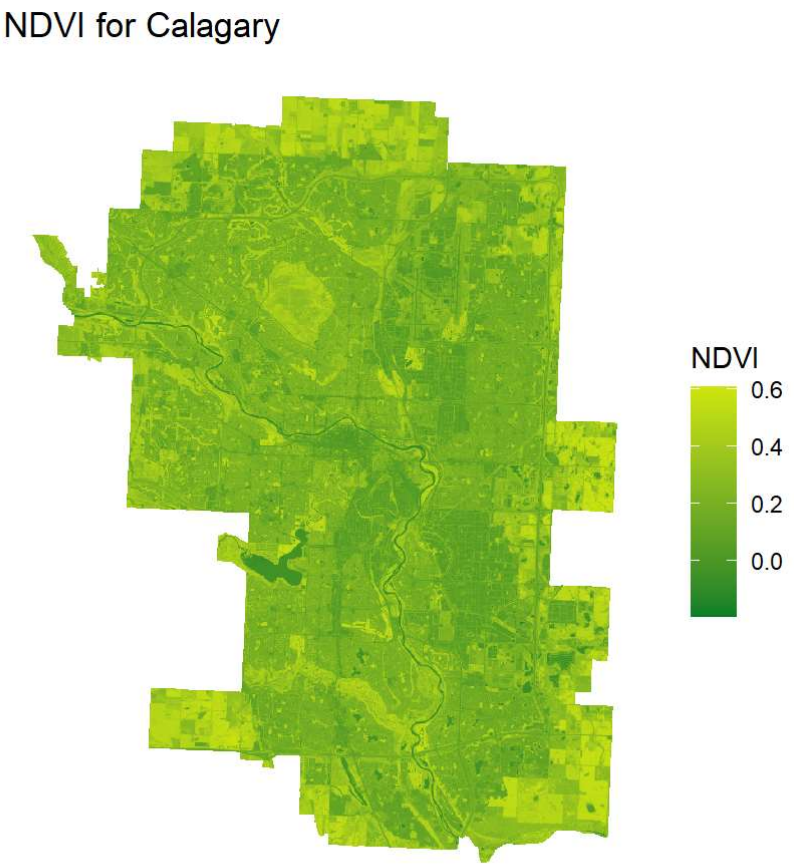
```
## [1] -0.1987892
```

```
#maximum
max(ndvi@data@values, na.rm = T)
```

```
## [1] 0.6099971
```

To plot the results with `ggplot`, we convert the raster into a data frame and use `geom_tile`.

```
as(ndvi, "SpatialPixelsDataFrame") %>%
  as.data.frame() %>%
  ggplot(data = .) +
  geom_tile(aes(x = x, y = y, fill = layer)) +
  theme(axis.text = element_blank(),
        axis.ticks = element_blank(),
        panel.background = element_blank(),
        panel.grid.minor = element_blank()) +
  labs(title = "NDVI for Calagary",
       x = " ",
       y = " ") +
  scale_fill_gradient(high = "#CEE50E",
                      low = "#087F28",
                      name = "NDVI")
```



If we compare the result to the false color composite created above, the areas with a higher NDVI correspond to the pockets of red that indicate more vegetation.

Supervised Classification

Classifying the Landsat 8 imagery will transform the satellite imagery into useable information. Rather than just having an image of a place, we will now know what grid cells represent what land cover. This will allow us to feature engineer variables, such as whether a grid cell is water or not. These features could prove helpful in various land use models.

Supervised classification is the process of training a predictive model on ‘ground-truthed’ land cover observations (grid cells we know definitively are forest or farmland) and subjecting that model onto new data to predict the land cover class.

We use `mapedit` and `mapview` to create the training dataset of ground-truthed points. `mapview` provides an interactive way to view spatial data in R. In the code below, `viewRGB` allows us to view the true composite of the satellite imagery in mapview and `editMap` will provide the tools to drop points down on the map. When we click 'done' in the mapview window the points we create will be saved as `points`.

Note that there is a 'show in new window' button at the top of the Viewer pane in R that will open mapview in an internet browser. This will make it easier to zoom in and drop points at the correct locations.

We will repeat this process **four** times, each time ground-truthing a different category we are interested in.

As we create the classification points, it is important to keep a few things in mind:

1. In the end, we should have a similar number of classification points for each category.
2. The points should be distributed relatively equally over the study area.

First, we will classify cloud cover. You must create points for one land use at a time, otherwise you will not be able to properly label the points.

```
# create training points in mapview
points <- viewRGB(image, r = 4, g = 3, b = 2) %>% editMap()
```

`points` is returned as a list. The following code extracts the final classification points for each type, converts the training points into an sf object, and adds two fields, `class` and `id`. Make sure you save the points before moving onto the next land use, or else you will overwrite the classification points you made previously.

```
# save as clouds after first iteration
clouds <- points$finished$geometry %>% st_sf() %>% mutate(class = "clouds", id = 1)
```

Repeat the above steps for the other three land classifications: developed land, undeveloped land (open space), and water. The code below saves the points and assigns a class and id for each of these land uses.

```
# save as developed land second time
developed <- points$finished$geometry %>% st_sf() %>% mutate(class = "developed", i
d = 2)
# then save as undeveloped land after third iteration
undeveloped <- points$finished$geometry %>% st_sf() %>% mutate(class = "undevelope
d", id = 3)
# finally save as water
water <- points$finished$geometry %>% st_sf() %>% mutate(class = "water", id = 4)
```

When creating the points for undeveloped land and water, it is helpful to use the false color composite to make vegetation stand out as red. To do so, use the below code:

```
# plotting the false color composite in mapview
points <- viewRGB(image, r = 5, g = 4, b = 3) %>% editMap()
```

Next, we `rbind` the different land cover categories together to create a final `training_points` sf object.

```
training_points <- rbind(clouds, developed, undeveloped, water)
```

You can write these points as a shapefile with `write_sf(training_points, "calgary_trainingPoints.shp", driver = "ESRI shapefile", getwd())` to be read back in later with `training_points <- st_read("calgary_trainingPoints.shp")`.

For this tutorial, I have pre-created between 150 and 200 training points for each category. You can read them in with the below code.

```
training_points <- st_read("./data/calgary_trainingPoints.shp", quiet = TRUE)
```

As mentioned above, when done correctly, there should be a similar number of classification points for each category and they should be distributed evenly throughout space. Let's plot the classification points I made.

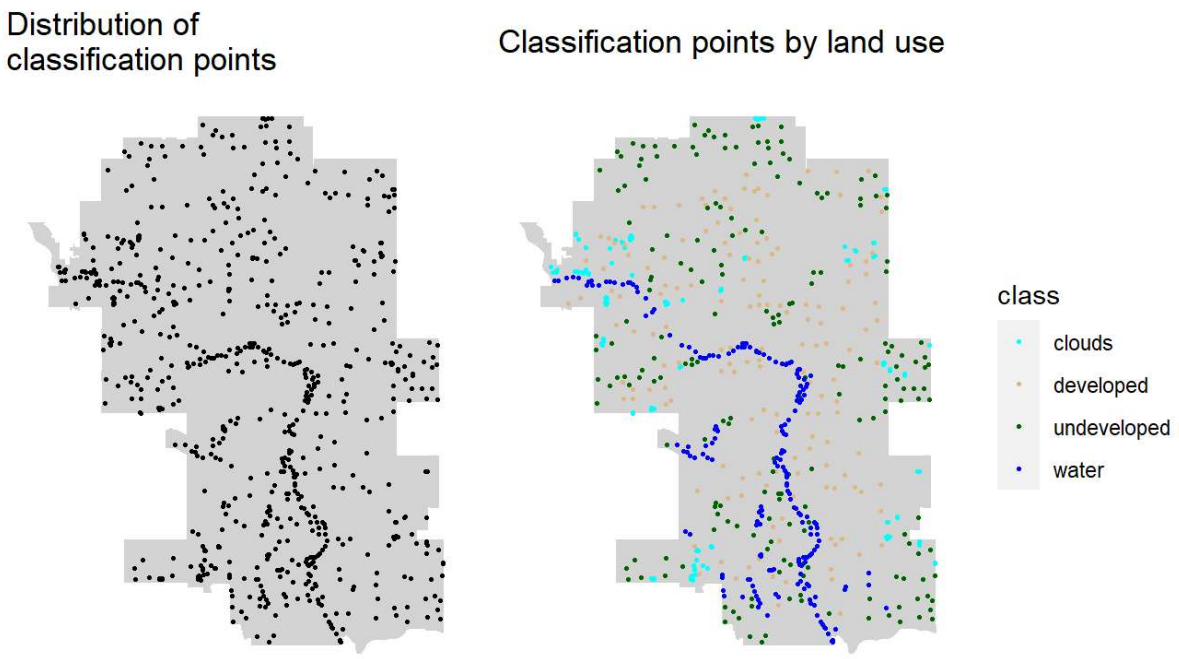
```
library(patchwork)

# read in the city boundary for calgary
cityBoundary <- st_read("./data/CityBoundary.geojson", quiet = TRUE)

# create a map looking at just the distribution of points
A <- ggplot() +
  geom_sf(data = cityBoundary, fill = "light gray", color = NA) +
  geom_sf(data = training_points, size = 0.5) +
  labs(title = "Distribution of\nclassification points") +
  theme(panel.background = element_blank(), axis.ticks = element_blank(), axis.text
= element_blank())

# create a map looking at the distribution of points by classification type
B <- ggplot() +
  geom_sf(data = cityBoundary, fill = "light gray", color = NA) +
  geom_sf(data = training_points, aes(color = class), size = 0.5) +
  scale_color_manual(values = c('cyan', 'burlywood', 'darkgreen', 'blue')) +
  labs(title = "Classification points by land use") +
  theme(panel.background = element_blank(), axis.ticks = element_blank(), axis.text
= element_blank())

# plot side by side
A + B + plot_layout(ncol = 2)
```



Extracting spectral values from the raster

Now that we have our training data, we will extract the spectral values from the imagery at the point locations. This requires that the points be formatted as a `SpatialPointsDataFrame`.

```
training_points <- as(training_points, 'Spatial')

df <- raster::extract(image, training_points) %>%
  round()
```

The result is a matrix of values for each spectral band.

```
head(df)
```

```
##      band1 band2 band3 band4 band5 band6 band7 band8 band9 band10 band11
## [1,] 23278 24686 26910 29927 34254 30755 19882 27654  5120  29911  27294
## [2,] 13592 13627 14518 15678 18062 18903 16396 15157  5138  30276  27412
## [3,] 12979 12679 12513 12704 16030 15946 14071 11222  5087  30123  27140
## [4,] 13547 13280 13815 14977 16552 15771 13154 16286  5095  31188  28323
## [5,] 15238 15537 16206 17117 18014 26436 27730 16433  5139  29918  25979
## [6,] 11807 11326 11111 11374 13239 13490 12915 11137  5082  30486  27288
```

Exploratory analysis: Plotting the spectral profile

As discussed above, different land covers will reflect and absorb the sun's radiation uniquely. A 'spectral profile' explores these differences by examining how the reflectance value for each band changes for different land cover types. Each land classification will have a unique spectral profile, with land cover like water having lower reflectance and features such as clouds having greater reflectance.

Here, we are looking at the spectral profile of the ground-truthed points. If we ground-truthed the data appropriately, we should see that each land classification has a unique spectral profile. This has important implications for our model. If the spectral profiles of our training data are not unique, it is unreasonable to expect the model to identify differences between land cover.

The spectral profiles are created by taking the mean reflectance values for each land classification for every band.

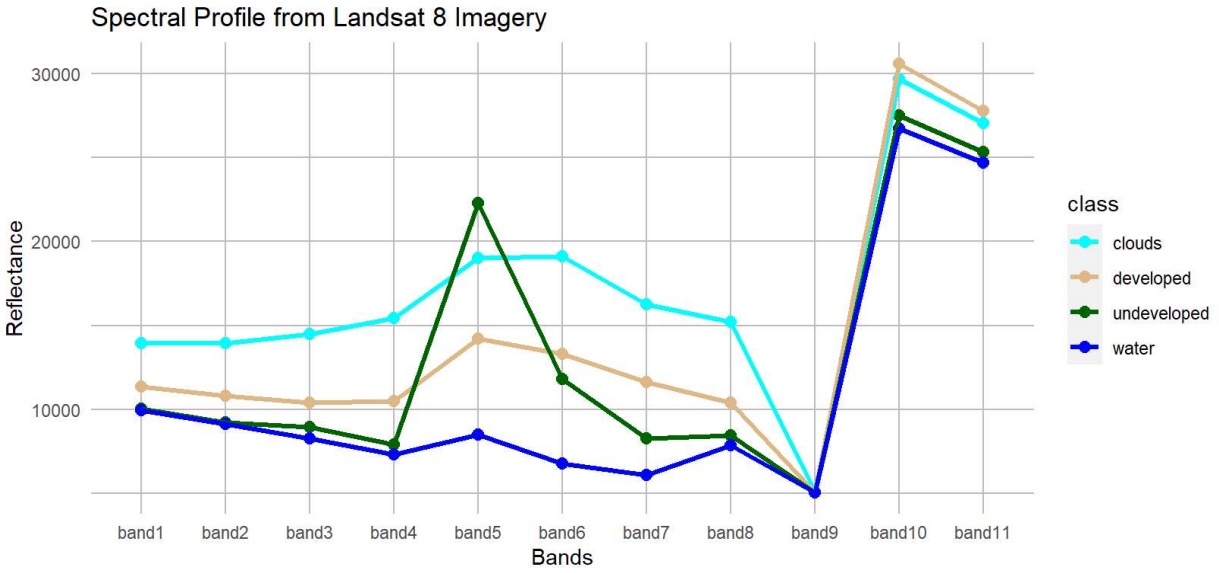
```
profiles <- df %>%
  as.data.frame() %>%
  cbind(., training_points$id) %>%
  rename(id = "training_points$id") %>%
  na.omit() %>%
  group_by(id) %>%
  summarise(band1 = mean(band1),
            band2 = mean(band2),
            band3 = mean(band3),
            band4 = mean(band4),
            band5 = mean(band5),
            band6 = mean(band6),
            band7 = mean(band7),
            band8 = mean(band8),
            band9 = mean(band9),
            band10 = mean(band10),
            band11 = mean(band11)) %>%
  mutate(id = case_when(id == 1 ~ "clouds",
                        id == 2 ~ "developed",
                        id == 3 ~ "undeveloped",
                        id == 4 ~ "water")) %>%
  as.data.frame()

head(profiles)
```

```
##            id    band1     band2     band3     band4    band5     band6
## 1      clouds 13939.56 13939.507 14474.048 15448.548 19043.53 19094.260
## 2   developed 11377.43 10799.662 10426.883 10485.799 14240.05 13323.766
## 3 undeveloped 10045.36  9216.410  8974.400  7929.635 22298.53 11825.015
## 4       water  9983.23  9139.885  8294.655  7337.490  8491.59  6799.415
##       band7     band8    band9   band10   band11
## 1 16265.342 15210.849 5099.027 29655.75 27027.03
## 2 11644.519 10393.318 5080.390 30585.69 27791.06
## 3  8280.075  8482.645 5087.400 27497.85 25338.66
## 4  6119.445  7868.505 5054.865 26732.20 24682.58
```

We then plot these values to assess if the spectral profiles of each land cover are unique. The plot shows that undeveloped land and water have similar reflectance values for bands 1 and 2, and that all four land cover classes have similar reflectance values for bands 9, 10, and 11. These spectral profiles suggest the model may have difficulty distinguishing between these land covers at those values.

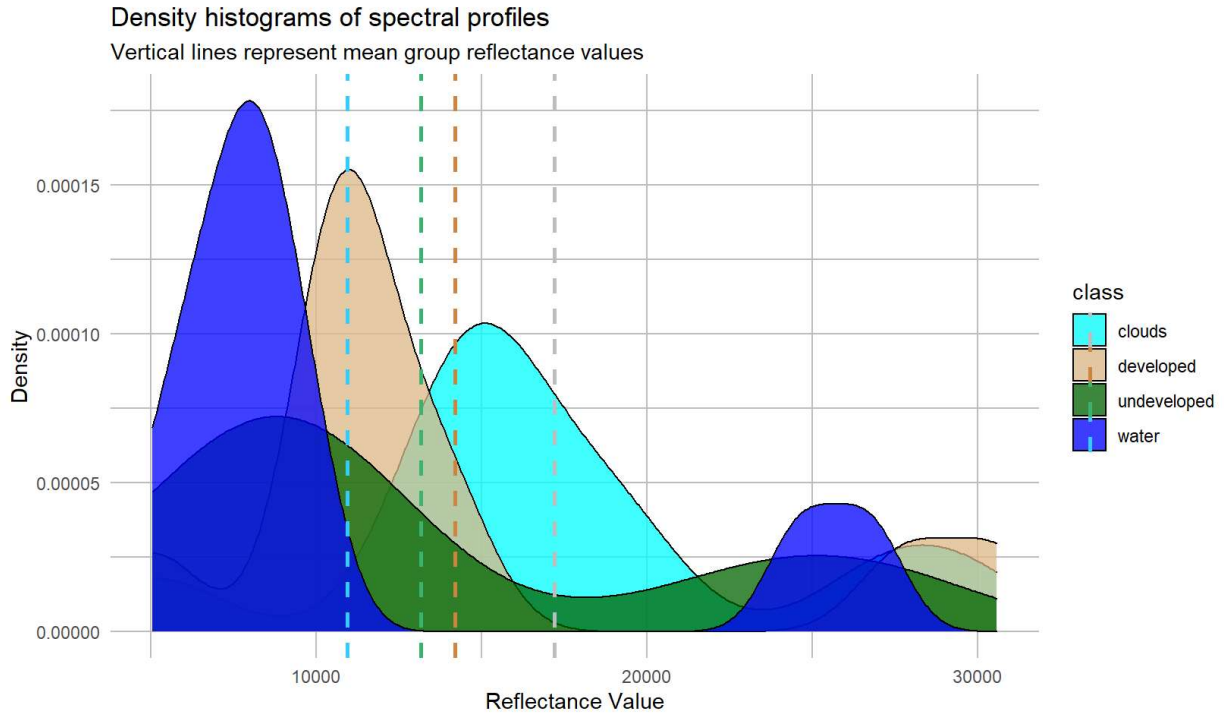
```
profiles %>%
  select(-id) %>%
  gather() %>%
  mutate(class = rep(c("clouds", "developed", "undeveloped", "water"), 11)) %>%
  ggplot(data = ., aes(x = fct_relevel(as.factor(key),
                                       levels = c("band1", "band2", "band3", "band
4","band5",
                                                  "band6", "band7", "band8", "band
9", "band10",
                                                  "band11")), y = value,
                           group=class, color = class)) +
  geom_point(size = 2.5) +
  geom_line(lwd = 1.2) +
  scale_color_manual(values=c('cyan', 'burlywood', 'darkgreen', 'blue')) +
  labs(title = "Spectral Profile from Landsat 8 Imagery",
       x = "Bands",
       y = "Reflectance") +
  #scale_y_continuous(limits=c(5000, 15000)) +
  theme(panel.background = element_blank(),
        panel.grid.major = element_line(color = "gray", size = 0.5),
        panel.grid.minor = element_line(color = "gray", size = 0.5),
        axis.ticks = element_blank())
```



Another way to assess whether the spectral profiles are unique is to plot the density of the reflectance values for each group. If the spectral profiles vary, the density histograms should have little overlap.

Looking at the mean reflectance values for each group will also illustrate if, on average, the sample points are representative of different parts of the spectrum.

```
profiles %>%
  select(-id) %>%
  gather() %>%
  mutate(class = rep(c("clouds", "developed", "undeveloped", "water"), 11)) %>%
  ggplot(., aes(x=value, group=as.factor(class), fill=as.factor(class))) +
  geom_density(alpha = 0.75) +
  geom_vline(data = . %>% group_by(class) %>% summarise(grp.mean = mean(value)),
             aes(xintercept=grp.mean, color = class), linetype="dashed", size=1) +
  scale_fill_manual(values=c('cyan', 'burlywood', 'darkgreen', 'blue'),
                    name = "class") +
  scale_color_manual(values=c("gray", "#CD853F", "#3CB371", "#33CEFF")) +
  theme(panel.background = element_blank(),
        panel.grid.major = element_line(color = "gray", size = 0.5),
        panel.grid.minor = element_line(color = "gray", size = 0.5),
        axis.ticks = element_blank()) +
  labs(x = "Reflectance Value",
       y = "Density",
       title = "Density histograms of spectral profiles",
       subtitle = "Vertical lines represent mean group reflectance values")
```



Once again, we see overlap between water and undeveloped land.

Classifying the imagery

We perform supervised classification on the imagery employing a decision tree algorithm. This type of algorithm uses a 'tree' structure to identify the best model that fits the data.

Decision trees start with the best predictor and continually subset the data. The if/else 'decisions' are represented as 'nodes' and the outcomes from the decisions become 'leaves' on the tree. The algorithm will use these rules to determine what the predicted outcome should be.

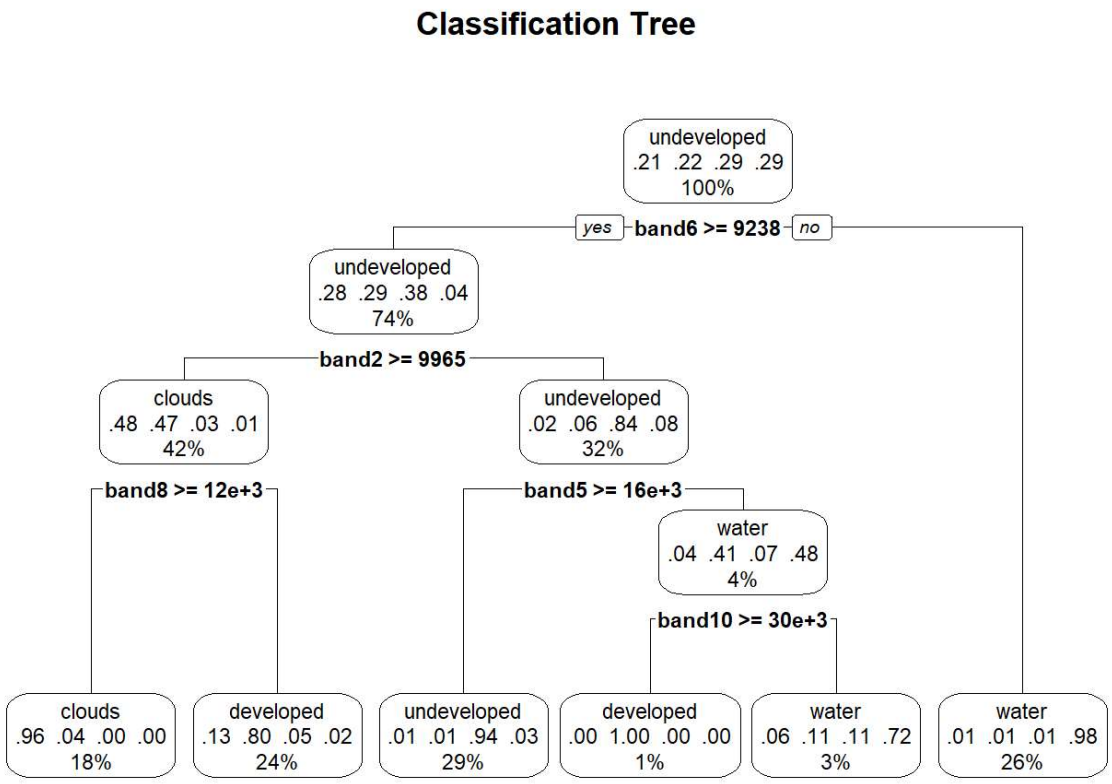
First, we combine the class and the extracted values into a dataframe. Then using `rpart`, we train the model.

```
df <- data.frame(training_points$class, df)

model.class <- rpart(as.factor(training_points.class)~., data = df, method = 'clas
s')
```

Next, we plot the decision tree.

```
rpart.plot(model.class, box.palette = 0, main = "Classification Tree")
```



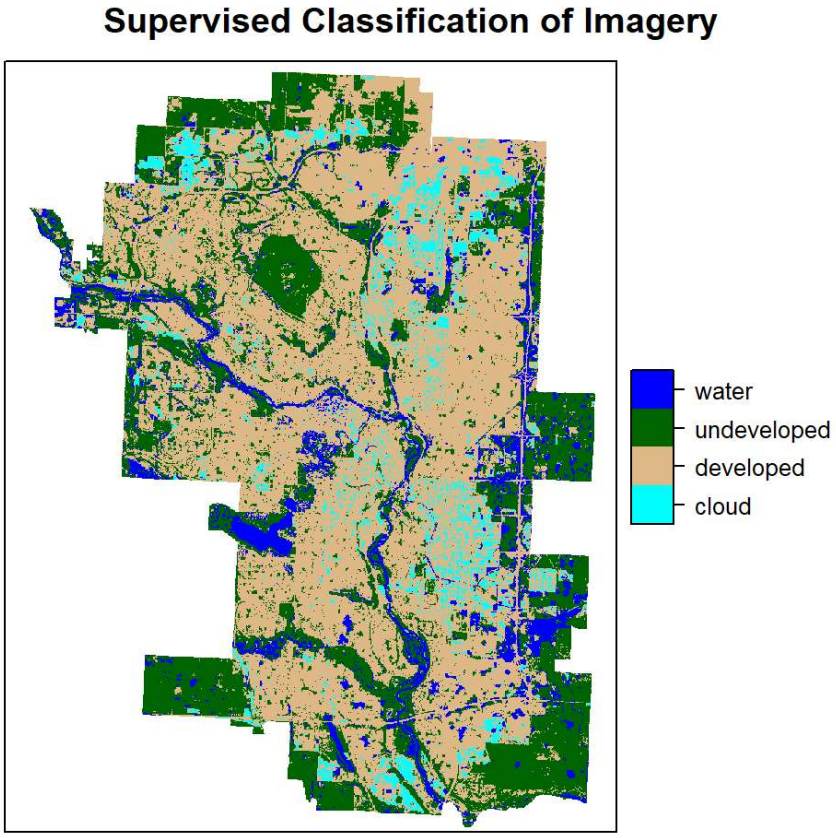
Using the model, we predict on the entire image and set the classes to the four land cover categories.

```
pr <- predict(image, model.class, type ='class', progress = 'text') %>%
  ratify()

levels(pr) <- levels(pr)[[1]] %>%
  mutate(legend = c("cloud","developed","undeveloped","water"))
```

Below are the results.

```
levelplot(pr, maxpixels = 1e6,
          col.regions = c('cyan', 'burlywood', 'darkgreen', 'blue'),
          scales=list(draw=FALSE),
          main = "Supervised Classification of Imagery")
```



Are these results reasonable?

We evaluate how well the model predicted by creating a confusion matrix that compares the predicted class to the observed ('ground-truthed') class.

```
test <- raster::extract(pr, training_points) %>%
  as.data.frame() %>%
  rename(id = ".")

testProbs <- data.frame(
  obs = as.factor(training_points$id),
  pred = as.factor(test$id)
) %>%
  mutate(correct = ifelse(obs == pred, 1, 0))

confMatrix <- confusionMatrix(testProbs$obs, testProbs$pred)
confMatrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2   3   4
##          1 121  20   3   2
##          2   5 143   3   3
##          3   0   9 188   3
##          4   0   3   6 191
##
## Overall Statistics
##
##                Accuracy : 0.9186
##                  95% CI : (0.8958, 0.9377)
##     No Information Rate : 0.2857
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                   Kappa : 0.8906
##
##  Mcnemar's Test P-Value : 0.006232
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4
## Sensitivity            0.9603   0.8171   0.9400   0.9598
## Specificity            0.9564   0.9790   0.9760   0.9820
## Pos Pred Value         0.8288   0.9286   0.9400   0.9550
## Neg Pred Value         0.9910   0.9414   0.9760   0.9840
## Prevalence             0.1800   0.2500   0.2857   0.2843
## Detection Rate         0.1729   0.2043   0.2686   0.2729
## Detection Prevalence   0.2086   0.2200   0.2857   0.2857
## Balanced Accuracy      0.9584   0.8981   0.9580   0.9709
```

The confusion matrix shows the model is more error prone in predicting developed land correctly. In fact, many of the observed developed land points were incorrectly classified as clouds. This could be a result of developed land appearing ‘brighter’ in parts of the city and explains why clouds are dispersed throughout the area in the final classification.

A key term in the confusion matrix is *Sensitivity* or the amount of grid cells predicted to be a specific land cover and were actually that land cover. Although the model struggles to predict developed land, the confusion matrix shows the model predicts 96.03%, 94%, 95.98% of the sample points correctly for clouds, undeveloped land, and water respectively.

Other tips and tricks

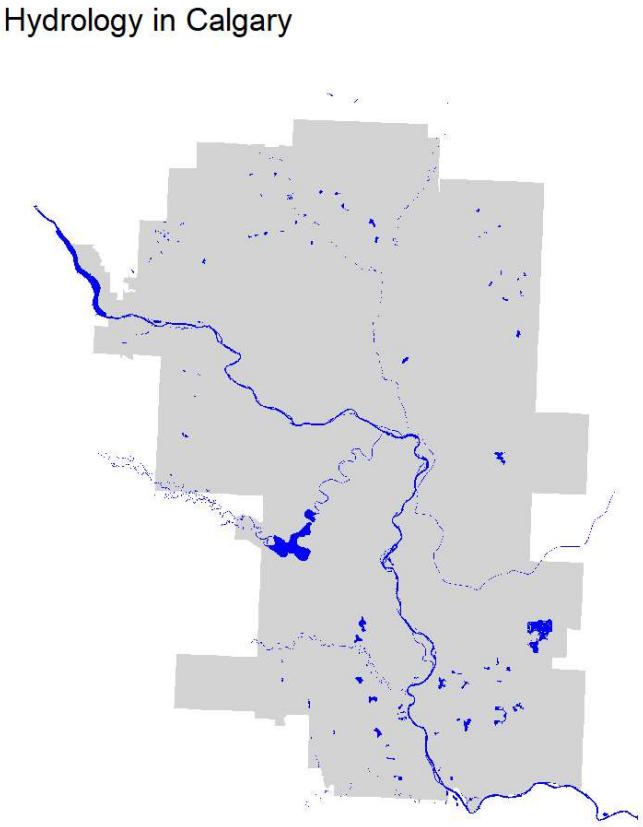
Masking rasters

When creating the classification points, `mapView` allows us to zoom into the image and using the false composite we could highlight some key areas of the image. Given that water and undeveloped land often looks quite similar (both dark in the image) it may be beneficial to mask the raster to better highlight those areas.

For example, using the hydrology from Calgary’s open data site, we can `mask` the image, so it only shows areas that fall inside hydrological features.

First read in the hydrology geojson and project it in the same coordinate system as the raster.

```
hydro <- st_read("./data/Hydrology.geojson", quiet = TRUE) %>%
  st_transform(crs = st_crs(image))
```



`mask` takes two inputs: 1) the raster and 2) the masking object.

```
image_mask <- mask(image, hydro)
```

Plot the masked raster in `mapView` to see if this would be helpful for creating water classification points.

```
points <- viewRGB(image_mask, r = 4, g = 3, b = 2) %>% editMap()
```

Subsetting predicted land use

The goal of remote sensing is to turn satellite imagery into usable information. One may want to recode the land use variables created through remote sensing or create features for a model from this data.

To get the data from the raster to a better format, we use `rasterToPoints` which creates centroid for each grid cell with the predicted land use class as an attribute of the dataframe.

```
imagePoints <- rasterToPoints(pr) %>%
  as.data.frame()
```

From here, we can filter for specific land uses or create new variables, such as a binary variable for developed land or not - mapped below.

```
imagePoints %>%
  mutate(developed_bin = as.factor(ifelse(layer == 2, 1, 0))) %>%
  ggplot() +
  geom_sf(data = cityBoundary %>% st_transform(crs = crs(image)), fill = "light gra
y", color = NA) +
  geom_point(aes(x = x, y = y, color = developed_bin), size = 0.5) +
  scale_color_manual(values = c("gray", "burlywood")) +
  labs(x = "", y = "",
       title = "Undeveloped classified land in Calgary") +
  theme(panel.background = element_blank(), axis.ticks = element_blank(), axis.text
= element_blank())
```