Week 2 Computer Lab Introduction: Land Cover Analysis in R

Scene 1: Land Cover Analysis in R

Welcome back! Today, our task is to analyze land cover data around the ponds where the Colombia spotted frogs were sampled. Specifically, most ponds are actually in the forest, and we want to find the size of the surrounding forest patch. Also, we want to know how much forest there is within 500 m from the pond.

To find the answers, we have an impressive list of things to address. At a more technical level, the main goal is to learn how spatial data are handled in R. We will also have a look at 'for' loops.

We will first talk about what land cover data are, and see how we can manipulate them in R with the 'raster' package. Then we will import the site data as spatial features with the 'sp' package. Before we can combine the land cover map and the sites, we have to tell R where on earth this is. Technically that means he have to specify the projection. Then we'll be ready to calculate patch size and percent of forest in a buffer around each pond. We'll do this separately for each pond, working through all ponds in a loop. Finally we'll take a step back and consider the relationship between R and Geographic Information Systems, GIS. Like last week, there is an interactive tutorial that you can work through to practice your R skills, and a Worked Example for future reference. This week there's also some Bonus Material that' I'll explain as we go.

Scene 2: What are Land Cover Maps?
Land cover maps tell us about the main land use or land cover at any point covered by the map. They are not created in the field but derived from satellite data, usually from Landsat imagery. Each spectral band measures reflectance in one part of the spectrum: blue, green, red, or near infrared. We could import each band into R as a raster layer, and combine them into a stack. A stack contains several raster layers with the exact same spatial information.

In essence, each raster layer shows how a single variable varies across space. For spectral bands, the variable is reflectance, and the values range from 0 to 255. We can easily plot this in grayscale. However, we can create an RGB color image by plotting the red band in red, the green band in green, and the blue band in blue. A raster dataset also contains spatial information about what part of earth the raster covers, and how big each cell is. Landsat data are typically processed with a cell size of 30 m.

Different land cover types reflect light differently, and this is used to classify grid cells into land cover types. The US National Land Cover Database maps 21 different land cover types, but in our study area, there are only eight of them present. They are coded with numbers between 11 and 95, but keep in mind that land use is a categorical variable and the codes are essentially short labels for the cover types, we can't do math on them.

Now when we import these land cover data into R as a raster, R does not realize that the data are categorical. We have to actively tell R that the numbers indicate factor levels. Even then, R will create a 'raster attribute table' but it does not really know what to do with that either. That's why we have included some Bonus Material this week that shows how to plot this map with the correct colors and labels.

Scene 3: Raster Manipulation with Package 'raster'

When you work with rasters, please don't call your rasters 'raster'. Why not? Here we have the word 'raster' four times in the same line of code. Let's pick it apart. One is the name of the function. We can see this because it is followed by the round brackets that contain any arguments to the function. That means the one inside the brackets is an argument. In this case, it defines the input object to which the function 'raster' is applied. The output object is always on the left, on the other side of the arrow. And finally, the double colon tells us that we are calling a function from the 'raster' package. This way we don't have to load the package first before calling the function.

The data in a raster layer object can be numeric, integer or logical, but not 'character'. Raster layers are meant for data that vary in space continuously. The raster package can't handle characters, and it will think that the codes of the cover types are integer data.

Here we see a summary of the raster layer with the land cover data, NLCD. It is an S4 object with several slots. These define the grid dimension, the resolution (that's the cell size), and other spatial information. At the bottom, we see that the values range between 11 and 95.

Because R treats the values as integer, we can do math on them. Here we multiply each value by 5 and subtract 17. We can also calculate the mean of all values with a dedicated function 'cellStat', or we can extract the values and take their mean manually. R will calculate it, but it does not make any sense to calculate the mean of the land cover codes. For a better summary, we use the function 'table' to tabulate how many cells of each type there are in the map.

We can also do logical tests on raster data, like on any other data. For example, if we want a map that shows only forest, we use double equation signs to test, for each cell, whether its value is 42, the code for Evergreen Forest. Here we have four cells for which this is True, for the other five it is False. The new raster 'Forest' thus contains a logical value, True or False, for each cell. The cool thing is that these can be automatically interpreted as a binary variable, where True equals 1 and False equals zero. We can now use the function 'mean' to calculate the mean of the zero's and one's, which is the same as calculating the percent of cells that were classified as Evergreen forest.

Scene 4: Site Data as Spatial Features in 'sp'

To import the site data for the ponds, we use a different package, 'sp'. This package treats the data as spatial features, where each spatial feature has spatial coordinates and attribute data, that’s all the variables measured at the location defined by the spatial coordinates. Each spatial point feature has only one pair of spatial x and y coordinates. A line feature is actually a sequence of points, or nodes, connected by the line, and therefore has multiple coordinate pairs. A polygon feature has at least an outer boundary defined by several points. It may also have an inner boundary around a hole.

In our data set, each pond was treated as a point, with a single pair of x and y coordinates. Here’s the data frame with the site data, and the first two columns are the coordinates. Each row is pond. We have additional columns for the site name, drainage, basin, substrate etc, 19 columns in total.

To create an ‘sp’ object, we use the function ‘coordinates’ and specify which columns in our data frame contain the spatial coordinates. In our case, this will turn the data frame into an object of the class ‘SpatialPointsDataFrame’. There are similar classes for lines, polygons, and even grid data. The coordinates will be extracted separately, and all the other columns form the attribute table.

So here we have a data frame with 31 ponds, and last week we had another data frame with genetic data for 181 frogs from 12 of these ponds. How would we extract site data for each frog? The technical term for this is joining data tables. Here we have an example of the frog data, where the column ‘Pop’ refers to the pond where each frog was sampled. In the site data, we have a column ‘SiteName’ that also refers to the pond. Now we can use the function ‘merge’ to join the two tables. The first two arguments, x and y, are the names of the two data frames. The arguments ‘by.x’ and ‘by.y’ specify the columns that should be used for joining. Now we can run the code, and we get a new data frame, where the attributes from the site data have been added to the frog data set. This is a safe way to combine data.

Scene 5: Where on Earth? Define Projection

Before we can plot the sites on the land use map, we need to specify the projection. GPS data are often in Longitude - Latitude format. These are measured in degrees, and one degree can be pretty far at the equator and much shorter towards the poles.

Many GIS data are available in UTM coordinates, which stands for Universal Transverse Mercator projection. UTM divides the world into zones and provides a ‘flat’ projection of the Earth’s surface within each zone. The units are in meters, and we can directly calculate distances from the UTM coordinates.

Now how do we tell R what the coordinates mean? The study area for the Colombia spotted frogs is in UTM zone 11. We can plug this into a simple function ‘get_proj4’, and it will return a rather complicated thing known as a proj4 string. Instead of loading the package ‘tmaptools’ that contains this function, we can use the double colon to call the package and function simultaneously. For lat-long data, we just change the argument inside the brackets and we’re good to go. All we need to do is add this information to our raster layers or ‘sp’ objects.

Scene 6: Patch-level Landscape Metrics

Now we can plot the sites over the land cover map! Remember that our first task was to determine the size of the forest patch around each pond. If we had a polygon map of forest patches, that would be easier, but the land cover data are in raster format with 30 m resolution. Most ponds don't show up as water in the land cover map, they are inside the forest.

To determine forest patch size, we first have to extract forest from the land cover map, then we delineate forest patches, giving a unique PatchID to each patch, and then we can calculate landscape metrics including the number of cells of each patch and its area in meters squared.

We have already learned how to use a logical test to create a binary forest map. Then we use the function ConnCompLabel from the SDMTools package to delinate patches, and 'PatchStat' from the same package to calculate patch-level landscape metrics. The result is a patch attribute table, and all we need to do is extract the values for the patches that we are interested in.

Scene 7: Class-level Metrics Within Buffer

The second task was finding the percent forest cover within a 500 m radius around each pond. We have to do this separately for each pond.

First we create a new raster that has only missing values except for all cells that lie within a 500 m buffer around the site. Then we extract the land cover data within that buffer, and calculate class-level landscape metrics. This results in a class attribute table, where each row is a land cover type, with the codes between 11 and 95, and each column is a different landscape metric. We find the percent forest within 500 m around this site in the line labelled '42', for Evergreen forest, and the column 'prop.landscape'. The pond in this table had about 40% forest cover within a radius of 500 m.

Scene 8: Looping Through Sites

Let's recall what we did for a single site: We created a raster map with the buffer around each site, then we created a land cover map within the buffer, and then we calculated class-level metrics for that map.

Now we need to repeat this for every site. For this, we create a loop with the function 'for'. Remember that the round brackets contain the arguments for the function. Here we need to define an identifier, which we call i, and we specify that i should take every value from 1 to n, the number of sites, which is 31 here. The colon symbol is used to create a series, here with all the numbers from 1 through n.

Then we have a set of curly brackets, and in between these we write the code that should be executed for each value of i. We just need to make sure that we call site 'i' where appropriate.

Now we need a place to store the results for each site without overwriting them. We create an empty list before the start of the ‘for’ loop. And at the end of the calculations for each site, we write the results into the i-th element of that list.

That’s all we need to loop through the sites. This is only pseudo-code, you can’t run it like this, but it shows you how the code in the Worked Example is constructed.

Scene 9: Final Thoughts: GIS Data in R

If you are somewhat familiar with Geographic Information Systems, you will have noticed that R can do many things that a GIS does. They actually get along quite well nowadays. If you need to exchange files, then I would recommend shapefiles for vector data, and geoTiff or asci grids for raster data. Vector and raster data are two paradigms that again remind me of angry birds. In R, we have dedicated object classes for raster data in the raster package and for vector data in the ‘sp’ package. There is also a new standard for vector data, simple features with the package ‘sf’, which is likely to replace ‘sp’ as the standard for vector data. You can learn more about it in this week’s Bonus Materials.

And that’s it for today, now it’s your turn with the tutorial.