Importing Arne data and extracting the values

Duncan Golicher

11/29/2018

## Introduction

The R code will produce a data table that you can analyse for the assignment by overlaying the data points from the field work with the layers produced by terrain analysis.

## Load all the terrain data

library(rpostgis)
library(RPostgreSQL)
library(sp)
library(raster)
library(rgeos)
library(rgdal)
library(RColorBrewer)
library(leaflet.extras)
library(mapview)
library(sf)
library(ggplot2)

### Set some mapview options for base maps
mapviewOptions(basemaps =c("Esri.WorldImagery","OpenStreetMap","OpenTopoMap", "OpenStreetMap.BlackAndWhite", "OpenStreetMap.HOT"))
####################3

path<- "/home/rstudio/webpages/Quantitative\_and\_Spatial\_Analysis/Quantitative\_and\_Spatial\_2018/Arne\_project"

aspect<-raster(sprintf("%s/gis\_data/aspect.tif",path))
slope<-raster(sprintf("%s/gis\_data/slope.tif",path))
dtm<-raster(sprintf("%s/gis\_data/dtm.tif",path))
dsm<-raster(sprintf("%s/gis\_data/dsm.tif",path))
chm<-raster(sprintf("%s/gis\_data/chm.tif",path))
chm10<-raster(sprintf("%s/gis\_data/chm10.tif",path))
sol<-raster(sprintf("gis\_data/sol.tif",path))
twi<-raster(sprintf("%s/gis\_data/twi.tif",path))
chm[is.na(chm)]<-0 ## Need to do this to avoid errors later as very low vegetation height values had been set to missing. Change them to zeros instead.
chm10[is.na(chm10)]<-0

conn <- dbConnect("PostgreSQL", host = "postgis", dbname = "bu" ,
 user = "msc\_student", password = "msc123")
query<-"select \* from arne.study\_area"
study\_area<-st\_read(conn, query=query)

### Also bring in the OSM paths vector layer for the study area
query<-"select osm\_id, st\_intersection(geom,way) geom from
arne.study\_area a,
dorset\_line d
where st\_intersects(geom,way)"

paths<-st\_read(conn, query=query)

study\_area <- st\_transform(study\_area, 27700)
sp\_study<-as(study\_area, "Spatial")
paths <- st\_transform(paths, 27700)
dbDisconnect(conn)

## [1] TRUE

## Read Ricardo’s data points into R

I’ll also load in last year’s results so that we have more data.

library(sf)
library(mapview)
path<-"/home/ricardol/Arne data/New folder/Shapefiles"
fls<-paste(path,dir(path,pattern="shp"),sep="/")

polys<-st\_read(fls[1])

## Reading layer `Arne Site Polys' from data source `/home/ricardol/Arne data/New folder/Shapefiles/Arne Site Polys.shp' using driver `ESRI Shapefile'
## Simple feature collection with 2 features and 2 fields
## geometry type: POLYGON
## dimension: XY
## bbox: xmin: 397798.6 ymin: 87541.09 xmax: 397885.3 ymax: 87629.59
## epsg (SRID): NA
## proj4string: +proj=tmerc +lat\_0=49 +lon\_0=-2 +k=0.9996012717 +x\_0=400000 +y\_0=-100000 +datum=OSGB36 +units=m +no\_defs

library(readr)
Arne\_Data\_2 <- read\_csv("/home/ricardol/Arne data/Arne data 2/Arne Data 2.csv")
d<- st\_as\_sf(Arne\_Data\_2, coords = c("X","Y"),crs=27700)
names(d)<-c("id","site","npines","geometry")

ly<-read.csv("/home/msc\_sarah/gis\_data/study\_points.csv")
last\_year<-data.frame(id=1,site="last\_year",npines=as.numeric(ly$no.t),X=ly$X,Y=ly$Y)
d2<- st\_as\_sf(last\_year, coords = c("X","Y"),crs=27700)

d2$pine\_density<-d2$npines/(pi\*4)
d$pine\_density<-d$npines/(pi\*1)

d<-rbind(d,d2)

mapview(d) +mapview(polys)



#st\_write(polys,"gis\_data/study\_polys", driver="ESRI Shapefile")
#polys<-st\_read("gis\_data/study\_polys")
#mapview(polys)

## Extract the terrain variables.

study\_points<-as(d, "Spatial")

study\_points$aspect<-raster::extract(aspect,study\_points)
study\_points$sol<-raster::extract(sol,study\_points)
study\_points$slope<-raster::extract(slope,study\_points)
study\_points$twi<-raster::extract(twi,study\_points)
study\_points$dtm<-raster::extract(dtm,study\_points)
study\_points$chm<-raster::extract(chm,study\_points)
study\_points$chm10<-raster::extract(chm10,study\_points)

sf\_study\_points<-st\_as\_sf(study\_points) # remake the sf object

## Extracting values around a point.

Another way of looking at the issue of finding values for a point is to take an area around the point and calculate some value based on **all** the pixels that fall into this buffer region. If there are a lot of points and very large raster layers this can take much longer. However it can make more sense to look at a neighbourhood around a point rather than the point itself.

If you recall, the chm10 layer is the canopy height layer averaged over around 10m rather than 2m (i.e. thinned by a factor of 5). So if we take a 5m buffer around the points and calculate the mean canopy height we will get a similar (but not identical) value.

We can do this in R using a rather longer code line (don’t worry about the syntax, just notice the inputs and the definition of the buffer radius in meters)

study\_points$chm5b<-unlist(lapply(raster::extract(chm,study\_points,buffer=5),mean))
sf\_study\_points<-st\_as\_sf(study\_points) # remake the sf object

Look at the comparison between the extracted values for chm10. If you want to calculate elements such as the slope **around** a point rather than the slope precisely **at** a point this is the way to do it. You might want to carefully copy the lines above and change the input variable and maybe the value of the buffer.

## Clipping the canopy height model raster

We are now going to use the canopy height model (chm) to automatically derive a layer holding the position of trees. However this layer extends far beyond the study site so some of the calculations would take time to run on the whole raster. So we will first crop the raster to the bounds of the study area.

chm\_study<-crop(chm, sp\_study) ## Use the sp class object

## Vectorising the trees

Distance calculations are much easier to run when both layers incolved are vectors. We will define “trees” as areas with a canopy height over 2 meters and vectorise (draw around) all the pixels that are selected by this definition.

chm\_study[chm\_study<2]<-NA ## Set below 2m as blanks (NA)
chm\_study[chm\_study>=2]<-1 ## Set above 2m to a single value
chm\_vect<-rasterToPolygons(chm\_study, dissolve=TRUE) ## Vectorise

The chm\_vect layer is a multipolygon of the boundaries around the clusters of pixels. Let’s take a quick look at it using a default plot.

plot(chm\_vect,axes=T)



As this layer has a multipolygon geometry if we calculate distances between the study points and the layer we will only get one value (the closest distance). Think in terms of distances to the coastline of British Isles. So if we want the distances between the points and all the polygons we need to split the layer up into single polygons and have multiple rows.

sf\_chm<-st\_as\_sf(chm\_vect) ## Change to an sf object
sf\_chm<-st\_transform(sf\_chm,27700) ## Just to ensure the exact same CRS string --- it was in 27700 but needed the epsg code

sf\_chm<-st\_cast(sf\_chm,"POLYGON")

## Warning in st\_cast.sf(sf\_chm, "POLYGON"): repeating attributes for all sub-
## geometries for which they may not be constant

Now look at the sf\_chm object. It is now in the form of single polygons. You will see a warning about the attributes being repeated. This is OK in this case, but think what the warning implies. If you split up a multipolygon of the British isles the area of each polygon would still be the area of the British isles, which would not necesssarily make sense. It is worth being aware of these sorts of issues, as they commonly arise in GIS work. They can be easily fixed, as you would just recalculate the areas, providing that you realised what was happening. There could be errors if you don’t, hence the warning. A GIS can’t guess new values when a multipolygon is split, without being told. The same applies to Arc and QGIS.

## Forming a distance matrix

We can now ask R to calculate **all** the distances that are possible between every single point and every single polygon. In some cases this could lead to a huge matrix, so this approach won’t always be practical for very large amounts of data. However it will work well here.

distances<-st\_distance(sf\_study\_points,sf\_chm)

OK so that’s done, but we don’t need to look at this object as it consists of 166500 units. It is just held in memory and can now be used for other calculations.

Now, if we want to find the distance between our study points and the nearest tree we will need to find the **minimum** distance in each rown of our matrix. We can do that in R using the following code.

sf\_study\_points$min\_dist<-apply(distances,1,min)

If we want some other type of calculation we can write any sort of function and *apply* it to the matrix. So, this function counts the number of polygons (trees) within a distance of 50 meters.

f50<-function(x)length(x[x<50])
sf\_study\_points$n50m<-apply(distances,1,f50)

Now look again at the sf\_study\_points object. We have the two extra columns added to it. We can write this all out again and we end up with a csv data file with all these derived variables added in. You might like to carefully alter the codeline to derive counts of trees within different distances. For example 100 m

f100<-function(x)length(x[x<100])
sf\_study\_points$n100m<-apply(distances,1,f100)

## Distance to paths

The same analysis can add in the distance to the nearest path.

paths<-st\_transform(paths,27700)
distances\_p<-st\_distance(sf\_study\_points,paths)
sf\_study\_points$paths\_dist<-apply(distances\_p,1,min)

## Conclusions

Although all this R code may look rather complicated, think about how much time this has all saved. If you wanted to measure these properties in the field it would have involved a lot of additional field work. Calculating the results in a point and click GIS would also take some time. With a little care and attention to the input data all the R code can now be reused to carry out exactly the same calculations at other sites for which Lidar data is available.

## Write out the data file

We can now form a CSV file and a shapefile with all the variable columns added as attributes.

st\_write(sf\_study\_points, "gis\_data/study\_points.csv", layer\_options = "GEOMETRY=AS\_XY", delete\_dsn=TRUE)

## Deleting source `/home/rstudio/webpages/Quantitative\_and\_Spatial\_Analysis/Quantitative\_and\_Spatial\_2018/Arne\_project/gis\_data/study\_points.csv' using driver `CSV'
## Writing layer `study\_points' to data source `/home/rstudio/webpages/Quantitative\_and\_Spatial\_Analysis/Quantitative\_and\_Spatial\_2018/Arne\_project/gis\_data/study\_points.csv' using driver `CSV'
## options: GEOMETRY=AS\_XY
## features: 163
## fields: 16
## geometry type: Point

## Write out study points as shapefile for direct use in another GIS

study\_points<-as(sf\_study\_points, "Spatial")
writeOGR(study\_points,"gis\_data","study\_point\_shapefile","ESRI Shapefile",overwrite\_layer =TRUE)

## Warning in writeOGR(study\_points, "gis\_data", "study\_point\_shapefile",
## "ESRI Shapefile", : Field names abbreviated for ESRI Shapefile driver

## Write chm\_vect as shapefile
chm\_vect\_sp<-as(chm\_vect, "Spatial")
writeOGR(chm\_vect,"gis\_data","chm\_vect","ESRI Shapefile",overwrite\_layer=TRUE)

system("zip -r gis\_data.zip gis\_data")

## Mapview of the results

Because the derived vector layer for the trees was imperfect there will be some errors in the results. This is not necessarily particularly important for a study of this type. All studies have some measurement errors, and statistical analysis is designed in part to take this into account. So, for example, the count of the number of trees within 50m might not actually represent the precise number of trees, but it would be closely correlated with this quantity. It would represent an index, rather than a precise measurement. This can be acceptable in many situations, providing that you fully document your methodology to avoid any confusion.

It is however a good idea to always check your results for general “sanity” by looking at the data in context. Making a mapview web map helps with this process.

If the vectorised layer for the trees is very different from the visible pattern shown in the satellite image an alternative approach would be to digitise each visible tree as a point and calculate the distance matrix using this layer instead.

mp<-mapview(sf\_study\_points,color="red") + mapview(chm\_vect) + mapview(polys)
mp@map <-mp@map %>% addFullscreenControl() %>% addMeasure(
position = "bottomleft", primaryLengthUnit = "meters", primaryAreaUnit = "sqmeters")
mp



## Further work

Try downloading all the layers in the **gis\_data\*** folder and form a desktop project in either Arc or QGIS. Because a desktopp GIS has a point and click interfaces it is easier to explore further processing options on your own using the options in the menus. However you will find that step by step data processing in a desktop GIS tends to lead to forming many intermediate files and becomes rather tedious. Using a pre-built R script is much quicker.

### Statistical analysis

The result of this data processing is a data frame with columns representing variables of interest. This is already loaded into memory in R, so the statistical analysis and the GIS data processing are linked together seamlessly when using R. We will start looking at statistics next week, but just to illustrate the idea, the following code will form a scatterplot of the number of pines against the distance to the nearest tree and fit a regresion line through the points. As I made up the numbers at random I would not expect to find any relationship.

g0<-ggplot(sf\_study\_points,aes(y=pine\_density, x=min\_dist))
g0<-g0+geom\_point()+geom\_smooth(method="lm")
g0



g0<-ggplot(sf\_study\_points,aes(y=pine\_density, x=site))
g0<-g0+geom\_boxplot()
g0



Just as expected, there is no relationship between the two variables. You will see how to formally test for relationships such as this through fitting statistical models in the next few weeks.