Voronoi diagrams: What R they for?

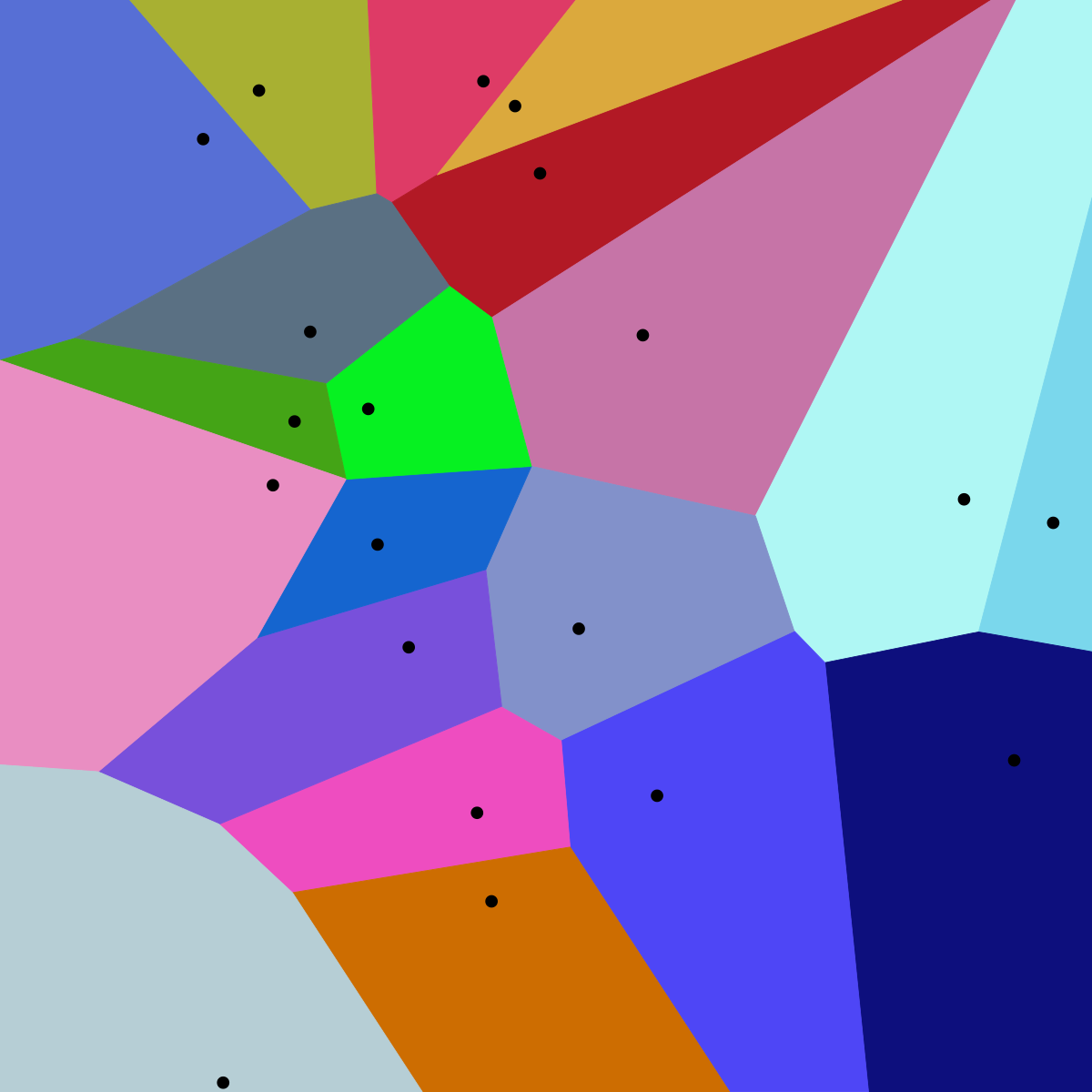
samuel gachuhi

2022-11-21

# Introduction

Imagine that you wanted to find out the source of a certain disease, and the closest locations to the source. What would you do? Of course a good approach would be to draw some concentric rings around each source point. Another similar, yet more sophisticated approach would be to you use voronoi polygons. In fact, using our allegory above, this was an actual exercise done by John Snow when mapping [cholera outbreaks in London](https://plus.maths.org/content/uncovering-cause-cholera). Surprised? London had cholera outbreaks back then, not like now where this disease is mostly confined to unsanitary conditions common in the global south, and recently, in some war zones as well.

What the heck are [voronoi polygons](https://gisgeography.com/voronoi-diagram-thiessen-polygons/)? They can also be called Thiessen polygons, after an American meteorologist called Alfred Thiessen. But lets name the feature by giving credit to the original creator–George Voronyi. Voronoi polygons are used to show how a specific location within a polygon is closer to the sample point in that polygon than any other sample point. After the polygons are created, neighbors of a sample point are defined as any other sample point whose polygon shares a border with the chosen sample point. For example, assuming that in the map below the points represent the sources of our disease, say cholera as in Snow’s story, then the polygons surrounding the points show the entire zone closest to that point compared to any other zone, including the neighbouring ones.



Voronoi Map

At first glance, voronoi polygons look like fancy maps. However, they play a useful role, such as helping telcos determine the transmitter that should serve your call. Ok. That’s an oversimplified example but you get the drift.

The following are some of the applications of voronois as extracted from this [article](https://sciendo.com/pdf/10.2478/pcr-2018-0009) of voronoi polygons:

1. Voronoi’s are used to construct maps of representiveness, for example points experiencing the same precipitation.
2. Voronoi diagrams are used in the research of socio-economic phenomena. Polygon’s in their classic form were used to determine transportation accessibility of railway stations and other transportation stops.
3. The method was used to analyze spatial patterns of distribution and accessibility of discount stores to determine the accessibility of green areas in certain large cities of Europe.
4. Voronoi diagrams have been used to delimit school zones and availability of medical services in an area called Presov, Slovakia.
5. In research conducted in the Beijing agglomeration, voronoi’s were used to determine the zone of transit transportation.
6. Voronoi polygons were used to delimit maritime zones in the Gulf of Salwa between Saudi Arabia and Qatar.

## Downloading files

We were not able to find a good dataset to create voronoi diagrams, therefore we decided to be creative and start from scratch (the norm rather than the exception). To give you a taste of what we will do, we will sample some altitude values in randomly selected locations from Kenya’s Digital Elevation Model (DEM). Thereafter, we shall create voronoi diagrams based on this *altitude* point sample locations. Sounds easy from afar, but far from easy.

Here are the usual packages we need.

# Packages for spatial analysis in R  
library(sp)

## Warning: package 'sp' was built under R version 4.2.2

library(sf)

## Warning: package 'sf' was built under R version 4.2.2

## Linking to GEOS 3.9.3, GDAL 3.5.2, PROJ 8.2.1; sf\_use\_s2() is TRUE

library(rgdal)

## Please note that rgdal will be retired by the end of 2023,  
## plan transition to sf/stars/terra functions using GDAL and PROJ  
## at your earliest convenience.  
##   
## rgdal: version: 1.5-32, (SVN revision 1176)  
## Geospatial Data Abstraction Library extensions to R successfully loaded  
## Loaded GDAL runtime: GDAL 3.4.3, released 2022/04/22  
## Path to GDAL shared files: C:/Users/gachuhi/AppData/Local/R/win-library/4.2/rgdal/gdal  
## GDAL binary built with GEOS: TRUE   
## Loaded PROJ runtime: Rel. 7.2.1, January 1st, 2021, [PJ\_VERSION: 721]  
## Path to PROJ shared files: C:/Users/gachuhi/AppData/Local/R/win-library/4.2/rgdal/proj  
## PROJ CDN enabled: FALSE  
## Linking to sp version:1.5-0  
## To mute warnings of possible GDAL/OSR exportToProj4() degradation,  
## use options("rgdal\_show\_exportToProj4\_warnings"="none") before loading sp or rgdal.

library(rgeos)

## rgeos version: 0.5-9, (SVN revision 684)  
## GEOS runtime version: 3.9.1-CAPI-1.14.2   
## Please note that rgeos will be retired by the end of 2023,  
## plan transition to sf functions using GEOS at your earliest convenience.  
## GEOS using OverlayNG  
## Linking to sp version: 1.5-0   
## Polygon checking: TRUE

library(raster)

Let’s first get the [DEM](http://datasets.wri.org/dataset/37617c74-1702-48b3-82f9-19d98b18200a/resource/6d340d80-7e95-439f-8329-2aca466cd58f/download/kesrtm.zip) we need.

# Load the raster we need  
kenya\_srtm <- raster("E:/downloads/Dukto/kesrtm/ke\_srtm/ke\_srtm/hdr.adf", band = 1, values = T)

Let’s see how the raster looks like.

# Checkout the raster  
# plot(kenya\_srtm)

For some reason, running the plot(kenya\_srtm) generates the following error:

Error in setValues(outras, m) : values must be numeric, logical or factor

Anyway, that won’t dumpen our spirits. Let’s alternatively checkout the values of our kenya\_srtm file.

# Check the values of kenya\_srtm  
kenya\_srtm

## class : RasterLayer   
## dimensions : 12130, 9603, 116484390 (nrow, ncol, ncell)  
## resolution : 0.0008333333, 0.0008333333 (x, y)  
## extent : 33.90886, 41.91136, -4.680452, 5.427881 (xmin, xmax, ymin, ymax)  
## crs : +proj=longlat +datum=WGS84 +no\_defs   
## source : hdr.adf   
## names : COUNT   
## values : -20, 4935 (min, max)

We had mentioned we wanted to generate a voronoi diagram from some random altitude points. We hinted we will start from scratch. That is, sample some altitude locations from our kenya\_srtm DEM, save this to point spatial file (not shapefile!), and thereafter create voronoi polygons from this randomly generated points.

To do the first part, creating points with altitude values, we have to do what we call point sampling.

## Point Sampling

If you are familiar with the point sampling tool in Qgis, you are in good hands. R also requires the same parameters, not unlike those of the aforementioned Qgis plugin: 1) a point shapefile, and, 2) a raster whose value fields will be feed into the point shapefile. Just note that to create a spatial points object in R, you need an extent in which the points should be delimited in. For the extent, we shall use the [administrative shapefile of Kenya](https://www.dropbox.com/scl/fo/gtmjpy54jj22qaloz0via/h?dl=0&rlkey=evchu10bwk6b6wwf2kgenb9lv).

# Load the administrative shapefile of Kenya  
kenya\_admin <- readOGR("E:/downloads/Dukto/counties\_shapefile/kenya\_counties/ken\_admbnda\_adm0\_iebc\_20191031.shp")

## OGR data source with driver: ESRI Shapefile   
## Source: "E:\downloads\Dukto\counties\_shapefile\kenya\_counties\ken\_admbnda\_adm0\_iebc\_20191031.shp", layer: "ken\_admbnda\_adm0\_iebc\_20191031"  
## with 1 features  
## It has 10 fields

Let’s do the point sampling operation.

pts <- spsample(kenya\_admin, 200, type = "random") # type "random" for completely spatially randomly located points

Let’s view our spatial points data.

# View the points  
head(pts@coords)

## x y  
## [1,] 35.68895 4.6900201  
## [2,] 40.65486 0.8470699  
## [3,] 35.53027 4.0267037  
## [4,] 33.95843 -0.4396059  
## [5,] 40.92953 -0.9834201  
## [6,] 35.13354 4.6138787

Reading the pts file with the suffix @data will not work because it does not have any data frame whatsoever. In other words, it has no attributes. It’s spatial positions can be recognised, but it’s values cannot be transferred to a readable file. That’s why the @coords, as opposed to @data suffix succeeded in displaying the contents of our spatial object.

To ameliorate this problem, lets convert pts to a dataframe.

# Convert the points `pts` to dataframe  
pts2 <- data.frame(pts@coords)  
head(pts2)

## x y  
## 1 35.68895 4.6900201  
## 2 40.65486 0.8470699  
## 3 35.53027 4.0267037  
## 4 33.95843 -0.4396059  
## 5 40.92953 -0.9834201  
## 6 35.13354 4.6138787

There you go! Like in a Cartesian plane, the x parameter represents the Longitudes, and likewise the y parameter represents the Latitudes.

Now we want to convert this points dataframe to a SpatialPointsDataframe. The SpatialPointsDataFrame function will combine the best of both worlds: a spatial object containing a dataframe. However, the SpatialPointsDataFrame function requires a dataframe of Long-Lat attributes (order matters!), and that is what we will first derive.

# Derive Longitude and Latitudes from pts2  
xy <- pts2[, c(1, 2)] # the columns follow after the comma (,) as in the convention [row, column]

# Create a SpatialPointsDataFrame object from pts2  
pts\_spdf <- SpatialPointsDataFrame(xy,   
 data = pts2,   
 proj4string = CRS("+init=epsg:4326"))  
head(pts\_spdf@data)

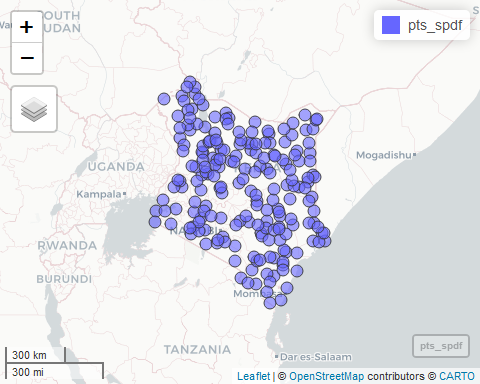
## x y  
## 1 35.68895 4.6900201  
## 2 40.65486 0.8470699  
## 3 35.53027 4.0267037  
## 4 33.95843 -0.4396059  
## 5 40.92953 -0.9834201  
## 6 35.13354 4.6138787

Now our humble legend @data suffix works! Let’s plot our spatial dataframe over a web map. You will need to install the mapview package. Installing packages in R follows this convention: install.packages("package name")

# Load the mapview package  
library(mapview)

## Warning: package 'mapview' was built under R version 4.2.2

# View the pts\_spdf variable  
mapview(pts\_spdf)



Time to put the much talked point sampling operation into action. All those aforementioned actions were just foretastes to the ultimate glory. Remember (once again), that we were to extract the altitude values from our elevation raster (kenya\_srtm), and into a spatial points object, now pts\_spdf. Here is the moment of glory.

# Extract raster value by points  
ras\_points <- extract(kenya\_srtm, pts\_spdf, method = "bilinear", fun = mean, buffer = NULL, cellnumbers = F, df = T) # 'bilinear' attribute calculates the mean of the surrounding four pixels for each raster cell  
  
# Check summary statistics of our new points file  
summary(ras\_points)

## ID COUNT   
## Min. : 1.00 Min. : 3.916   
## 1st Qu.: 50.75 1st Qu.: 289.050   
## Median :100.50 Median : 530.643   
## Mean :100.50 Mean : 701.900   
## 3rd Qu.:150.25 3rd Qu.: 915.211   
## Max. :200.00 Max. :2616.322

Running a brief overview of our new points file –the ras\_points, shows that we are missing our xy long-lat coordinates.

head(ras\_points)

## ID COUNT  
## 1 1 416.18390  
## 2 2 117.54187  
## 3 3 727.84208  
## 4 4 1138.63629  
## 5 5 86.67539  
## 6 6 527.81020

Without grumbling (by which you could be doing now), lets merge our pts\_spdf with ras\_points to get both the Long-lat (xy) and altitude (count) values. If you want to plot your points with ggplot, the xy fields are very important.

# Combine raster values with point data  
ras\_points\_combined <- cbind(pts\_spdf, ras\_points)  
  
# Check if the two variables have been merged  
head(ras\_points\_combined)

## x y ID COUNT  
## 1 35.68895 4.6900201 1 416.18390  
## 2 40.65486 0.8470699 2 117.54187  
## 3 35.53027 4.0267037 3 727.84208  
## 4 33.95843 -0.4396059 4 1138.63629  
## 5 40.92953 -0.9834201 5 86.67539  
## 6 35.13354 4.6138787 6 527.81020

Good.

Let’s change the value of the COUNT field to values to avoid any confusion.

# Change the "COUNT" field to read "values"  
names(ras\_points\_combined)[4] <- "values"  
names(ras\_points\_combined) # Read the names of the fields again

## [1] "x" "y" "ID" "values"

## Plotting the voronoi diagrams

We will plot the voronoi diagram using the ggplot tool. Awaken it into R-eality.

# Load the ggplot package  
library(ggplot2)

## Warning: package 'ggplot2' was built under R version 4.2.2

Remember ggplot only works with dataframes. We have gone through a lot to create our ras\_points\_combined object so far, but does it fit the bill for ggplot to work on it?

# Check the type of our ras\_points\_combined object  
typeof(ras\_points\_combined)

## [1] "S4"

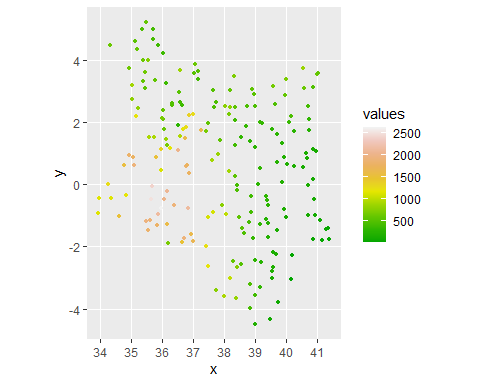
Long story short, s4 objects won’t work inside ggplot. Convert it to a dataframe. Period.

# Convert ras\_points\_combined to dataframe  
ras\_points\_combined2 <- data.frame(ras\_points\_combined)  
  
# Check the type of our resulting object  
typeof(ras\_points\_combined2)

## [1] "list"

Time to plot our ras\_points\_combined2 containing Long-Lat (xy) coordinates with altitude data.

# Plot ras\_points\_combined2  
elev\_map <- ggplot(data = ras\_points\_combined2, aes(x, y)) +   
 geom\_point(aes(color = values), size = 1) +   
 scale\_color\_gradientn(colours = terrain.colors(10)) +  
 coord\_sf(default\_crs = sf::st\_crs(4326))  
  
elev\_map



Just take a minute and digest the distribution of points in the above map. Take note of regions with denser distribution of points and those with less point density. This will help you in understanding how voronoi polygons convey information, such as denoting regions with higher point density as in this case.

Speaking of voronoi diagrams, here is the library that plots them in R.

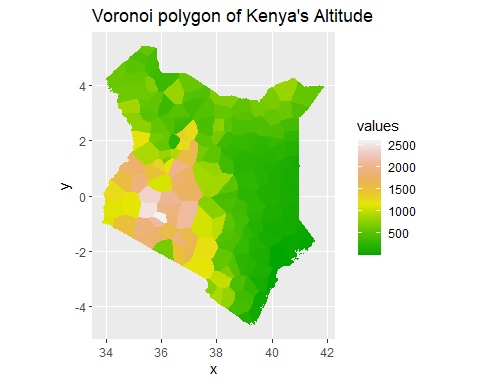
# Package for plotting voronoi diagrams in R  
library(ggvoronoi)

## Warning: package 'ggvoronoi' was built under R version 4.2.2

The voronoi diagrams are plotted using the geom\_voronoi function. Of course a ggplot object is needed on which the voronoi polygons will be overlain.

# Plot the voronoi diagram  
elev\_map +   
 geom\_voronoi(aes(fill = values),   
 outline = kenya\_admin) +  
 scale\_fill\_gradientn(colors = terrain.colors(10)) +   
 labs(title = "Voronoi polygon of Kenya's Altitude")

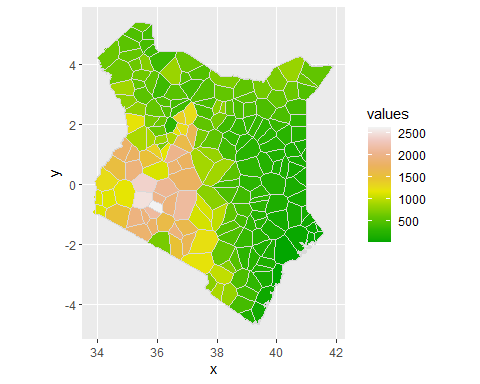
## Warning in gBuffer(outline\_spdf, byid = TRUE, width = 0): Spatial object is not  
## projected; GEOS expects planar coordinates



Do the distribution of points match to the number of polygons appearing in a certain region? Perhaps delineating borders will make this more succinct.

# Plot voronoi diagrams With polygon boundaries  
elev\_map +   
 geom\_voronoi(aes(fill = values), color = "gray85", linewidth = 0.01,  
 outline = kenya\_admin) +  
 scale\_fill\_gradientn(colors = terrain.colors(10))

## Warning in gBuffer(outline\_spdf, byid = TRUE, width = 0): Spatial object is not  
## projected; GEOS expects planar coordinates



This is clearer. The eastern region has a denser network of points than towards the west and southwest. If the spatial points object used to draw the voronoi polygons was a school’s dataset, you would say the eastern and northern regions have a denser coverage of schools than the west. This is just for explanation purposes. This, in our country, is far from the truth.

# Conclusion

Voronoi polygons are helpful in knowing which locations, within a particular region, are closest to the sampling point. In this exercise, plotting of voronoi polygons was not a click-and-run exercise, unlike in most GIS softwares such a ArcGIS. The drawing of voronoi diagrams followed a linear pattern, with each step building on preceding ones until finally the data was compatible enough for R to plot voronoi polygons. The steps involved began by delimiting the extent of operations, proceeding to point sampling and finally extraction of raster values. Finally, to close this operation, the inventor of this application, George Voronyi, was himself a Ukrainain.

George grew up in Ukraine, studied in Russia (St Petersburg), taught a lot in Warsaw (Poland) and was finally buried in Zhuravka (Ukraine). The first two countries are embroiled in a tense conflict, and the third, Poland, is acting as a safe haven for panic stricken refugees. How ironic that the countries of his birth and where he received most of his tutelage are now arch-enemies, and where he lectured serves as a host to many of his fleeing countrymen.

FINIS.