Introduction to the analysis of spatial data using R

Ana I. Moreno-Monroy

13-16 June 2017

Chapter 4: Geolocation. Using OSM data

- Geolocation and commuting distances using ggmap and Google's API
- Geolocating a large number of addresses
- Verifying, visualizing and exporting the output
- Open Street Map Spatial Data: Basic usage

Tile mapping with ggmap

- The **qmap** function of the **ggmap** package can geolocate and plot at once. For other aesthetic options, see ?get_map

```
library(ggmap)
qmap("Campus Bellissens URV, Reus", zoom = 15, color = "bw", legend = "topleft")
```

Geolocating using ggmap

- **ggmap** uses the geolocating services available through Google's Geolocoding API
- There is free use of these services with a limit of 2,500 request per day
- To store the geolocation data, we can use the **geocode** function

```
info<-geocode("Campus Bellissens URV, Reus", output = "more")
info
```

```
##       lon      lat          type     loctype
## 1 1.12311 41.14455 establishment approximate
##                                              address   north   south
## 1 av. de la universitat, 4, 43204 reus, tarragona, spain 41.1459 41.1432
##       east     west street_number                      route locality
## 1 1.124459 1.121761             4 Avinguda de la Universitat     Reus
##   administrative_area_level_2 administrative_area_level_1 country
## 1                   Tarragona                   Catalunya   Spain
##   postal_code
## 1       43204
```

- It is also possible to find an address from a pair of coordenates (reverse geolocation), using the **revgeocode** function

Calculating distances using ggmap

- From R we can get routing information from Google's Distance Matrix API related to commuting times by different modes (and also costs, when available)
- The **mapdist** offers an easy way to get this information

```
from <- c("Gaudi Centre, Reus")
to <- c("Campus Bellissens URV, Reus")
mapdist(from, to, mode="bicycling")
```

```
##                 from                          to    m    km    miles
## 1 Gaudi Centre, Reus Campus Bellissens URV, Reus 2037 2.037 1.265792
##   seconds  minutes     hours
## 1     467 7.783333 0.1297222
```

Geolocating a large number of addresses

- Before geolocating a large number of addresses, it is advisable to check their consistency, and simplify them if necessary
- The process can start with trying to find manually in Google Maps a couple of addresses as they are given in the original dataset. If they are not found, try again removing extra information (e.g., apartment or bloc numbers, house numbers) until you find a format that works for Google Maps
- It is possible then to automatize the "cleaning" process in R using string functions
- The number of well located addresses will depend on how much information Google has on the location
- In many cases Google interpolates to return a match (read more about "Types" of result here). The result could be close to the actual location or very far away. Always verify the result by comparing the return and original addresses

Batch geolocation: an example

- Before starting, read the Google Geocoding API website, and get and activate your API key
- Load example file with addresses in Medellin (saved in Stata in tab separated format) and see first three entries

```
loc<-read.table("direcciones_MDE.TXT", header=TRUE, encoding = "utf-8")
loc[1:3,]
```

```
##          barrio                   direccion
## 1    Alejandría CL 9 B SUR 25 -161   AP 601
## 2       La Mota     CL 5  76 A -97   AP 501
## 3 La Aguacatala      CL 10  28 -70   AP 604
```

- A manual search of "CL 9 B SUR 25 -161 AP 601, Medellin" does not return any resutls on Google Maps
- **Important!** To avoid trouble with international (Spanish) accents, save your .R script with encoding UTF-8 (File/Save with Encoding)

Cleaning addresses

- We can start by removing extra information from address, for instance information in parenthesis
- We first need to find an opening parenthesis within each string and then remove content
- The base function **regexpr** retuns an integer vector giving the starting position of the first match of a given expression or -1 if there is no match

```
ind<-regexpr("(", loc$direccion, fixed=TRUE)
ind[!ind==-1]
```

```
## [1] 17
```

```
loc$direccion[17]
```

```
## [1] CR 45 A 93 -195 (INTERIOR 201 )
## 96 Levels: CL 1 SUR 34 -95   AP 201 ... TV 38  71 -146   AP 302
```

Cleaning addresses

- Use the function **substr** to remove content within parenthesis, after setting start and stop values. As we do not want to modify those without a match, we replace -1 values for the number of characters

```
ind[ind<0]<-length(loc$direccion)
loc$direccion1<-substr(loc$direccion, start=1, stop=ind-1)
loc$direccion1[17]
```

```
## [1] "CR 45 A 93 -195 "
```

- We can do the same for several words or symbols

```
matches<-c("\\(", " IN", " AP", " BL", " PS", " TORRE", " TR", " CASA", " CA")
ind<-regexpr(paste(matches, collapse = "|"), loc$direccion)
ind[ind==-1]<-length(loc$direccion)
loc$direccion1<-substr(loc$direccion, start=1, stop=ind-1)
loc$direccion1[1:3]
```

```
## [1] "CL 9 B SUR 25 -161  " "CL 5  76 A -97  "     "CL 10  28 -70  "
```

Geolocating

- After cleaning, we can build the search phrase using the **paste** base function

```
direccion <- paste("Medellin, Antioquia, Colombia,", loc$direccion1)
direccion[1]
```

```
## [1] "Medellin, Antioquia, Colombia, CL 9 B SUR 25 -161  "
```

- We can now use the **geocode** function to batch geolocate (here for the first 10 as illustration). We use the option output="more" to get the reference adresss

```
locations <- geocode(direccion[1:10], output="more")
locations[1,1:5]
```

```
##         lon      lat           type            loctype
## 1 -75.56225 6.189667 street_address range_interpolated
##                                            address
## 1 cl. 9b sur #25-161, medellín, antioquia, colombia
```

Comparing original and return addreses

- We start by add the original address to the locations data frame

```
locations$original <- paste(loc$direccion1[1:10], "Medellin, Antioquia, Colombia")
locations$original[1]
```

```
## [1] "CL 9 B SUR 25 -161   Medellin, Antioquia, Colombia"
```

```
locations$address[1]
```

```
## [1] "cl. 9b sur #25-161, medellín, antioquia, colombia"
```

- We then give the two strings the same format using the **gsub** base function (find and replace function)

```
drop<-c(" ", "\\.", "-", ",", "#")
locations$address1<-gsub(paste(drop, collapse = "|") , "", locations$address)
locations$original1<-gsub(paste(drop, collapse = "|"), "", locations$original)
```

- And convert to all capitals using the **toupper** function

```
locations$address1<-toupper(locations$address1)
locations$original1<-toupper(locations$original1)
```

Check performance

- To check whether the original and return addresses are the same, we use the **amatch** function of the **stringdist** package

```
library(stringdist)
ind_match<-amatch(locations$address1, locations$original1,  maxDist=1)
```

- The **amatch** function returns an index of the closest match. In this case it corresponds to the same line, but note that matches could be made even if the strings to compare are not in the same order (similar to VLOOKUP in Excel)

```
library(stringdist)
ind_match<-amatch(locations$address1, locations$original1,  maxDist=2)
ind_match
```

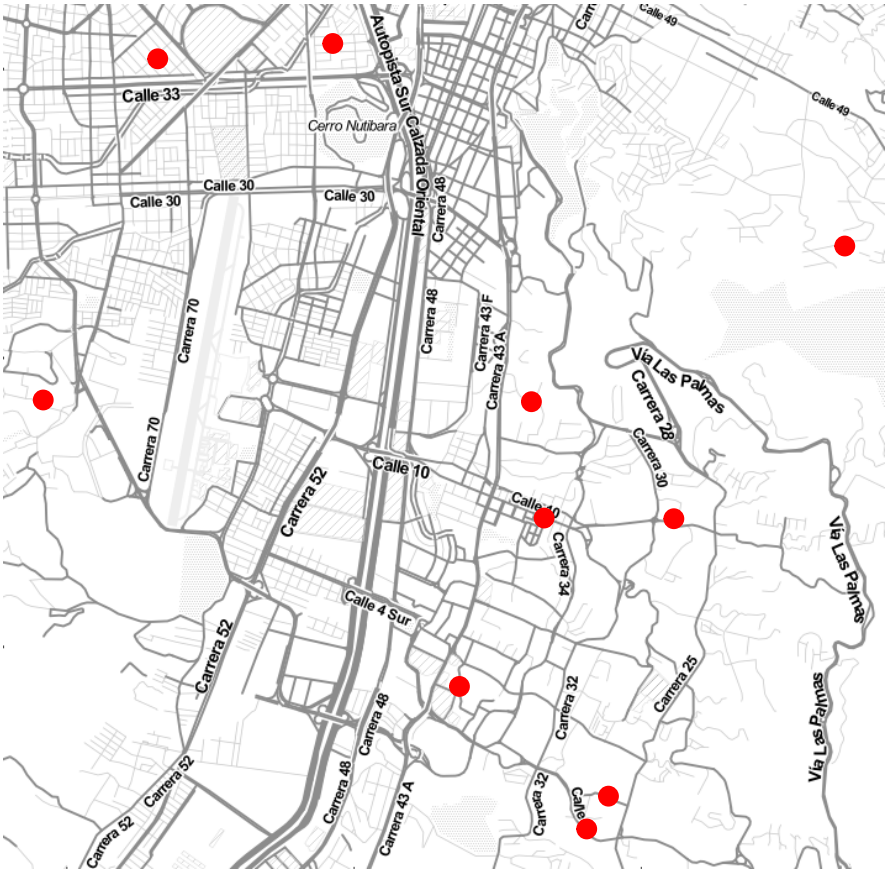
```
##  [1]  1 NA  3  4 NA  6 NA  8  9 NA
```

- According to this result, we have geolocated 5 out of 10 addresses precisely
- We can change the "maxDist" parameter to allow for small differences (e.g. due to typos). Experiment with maxDist=2 and compare the results

Visualizing and exporting the result

- We can plot the geolocated points to verify if the extent makes sense

```
qmplot(lon, lat, data = locations, colour = I("red"), size = I(3))
```



- To save the map as a .jpg image (PDF is also an option) we open the request and call the function **qmplot**. The dev.off() command shuts off the graphic device at the end

```
jpeg("plot_mde.jpg")
qmplot(lon, lat, data = locations, colour = I("red"), size = I(3))
dev.off()
```

```
## pdf
##   2
```

Exporting as data frame

- To export the locations data frame with the geographical coordenates and additional information, we can use the **write.csv** base function

```
write.csv(locations, "coordenates.csv")
```

- Another option for exporting in tab delimited format is **write.table()**. If you are having trouble with the way Excel reads points and commas from the exported data, try **write.csv2** (it uses a comma for the decimal point and a semicolon for the separator, istead of "." for the decimal point and a comma for the separator, as **write.cvs**)

Open Street Map Spatial Data: An introduction

- Open Street Map contains a wealth of spatial information for most countries in the world
- Some spatial variables available for download on shapefile format include administrative boundaries at different levels, rail, public and road networks, waterways and land use delimitations (e.g. natural, residential), as well as user fed points of interests (restaurants, churches, etc.)
- To start, check the basic availability of data at the Open Street Map for your place of interest. You can navigate through the different layers
- To download the spatial data, go to "Export" on the OSM website. Usually, the best way to download the data is through one of the external sources, such as Geofrabrik or Metro extracts

OSM: downloading and handling

- For instance, all the spatial data available for Barcelona can be downloaded in SHAPEFILE format from the Metro Extract for Barcelona
- Note that these databases are usually heavy. Subsetting and selecting the relevant data for your case can only be done after downloading the whole extract
- For easier initial visualization and browsing, you can open the shapefiles in a GIS software such as QGIS

OSM data: an example

- Suppose we wanted to identify the spot with the highest density of night entretainment in Barcelona
- After downloading and unzipping the data from Metro Extracts, we can open the shapefile corresponding to "osm_point_1" (a smaller version of osm_point):

```
library(rgdal)
bcn_poi <- readOGR(dsn = "barcelona_spain.osm", layer = "barcelona_spain_osm_point_1")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "barcelona_spain.osm", layer: "barcelona_spain_osm_point_1"
## with 11809 features
## It has 1 fields
```

OSM data: an example

- To subset information, first look at the levels of the column "amenity"

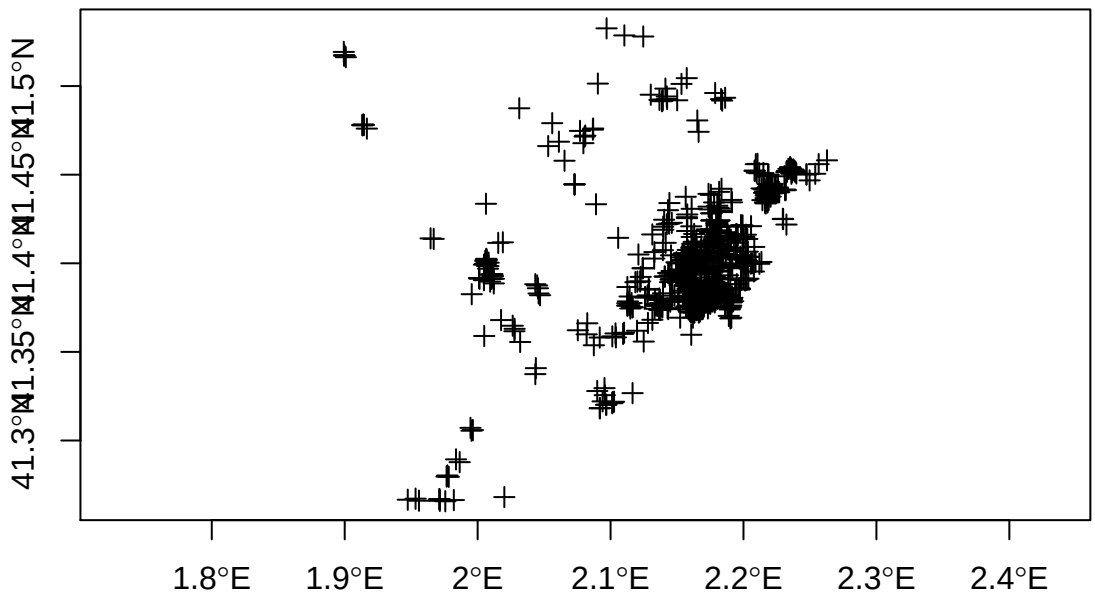
```
levels(bcn_poi@data$amenity)
```

- We subselect data corresponding to bar, pubs and nighclubs

```
bcn_party<-bcn_poi[bcn_poi@data$amenity %in% c("bar", "nightclub", "pub"),]
```

Plot the result

```
plot(bcn_party, axes=T)
```



Density

To find the spot with the highest density, we will calculate a smooth Kernel density with a bandwidth of 500 meters. First we need to project the plain coordinates so that we can work with distances in meters:

```
bcn_party_p = spTransform(x = bcn_party, CRSobj = CRS("+init=epsg:2062"))
```

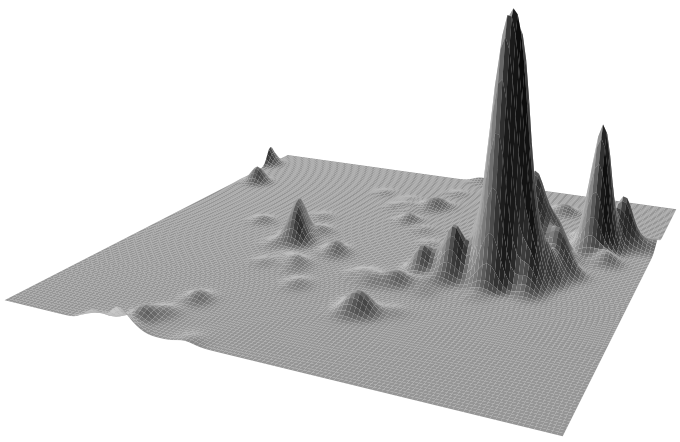
Then we can create a ppp (point pattern) object using the **as.ppp** function of the **spatstat** package. Using the **density.ppp** function on this object we can compute a kernel smooothed intensity function with a bandwidth -sigma- of 500 meters. The option "owin" defines the spatial extent.

```
library(spatstat)
p <- as.ppp(bcn_party_p@coords, owin(bcn_party_p@bbox[1,], bcn_party_p@bbox[2,] ))
dp_all <- density.ppp(p, sigma=500)
```

Density plot

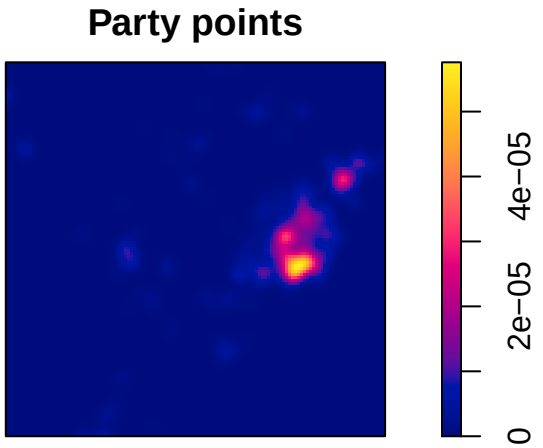
Objects created with the **density.ppp** function can be easily plotted in 3-D using the **persp** function

```
persp(dp_all, d=1, col="lightgrey",  box=FALSE, border=NA, shade=0.5, theta = 25, main="Party points")
```



Alternatively, we can obtain a heatmap using the base function **plot**

```
plot(dp_all, main="Party points")
```



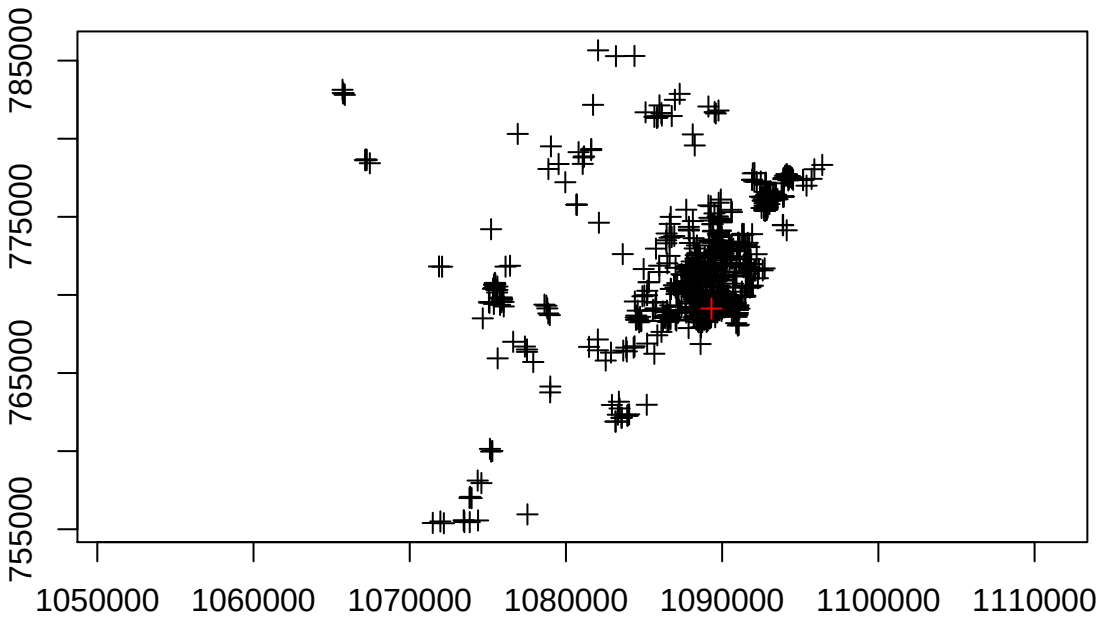
Density at each point

We have the density surface of night entretainment, but to identify the point with the highest density. The option at="points" from the **density.ppp** function allows us to do this easily

```
bcn_party_p@data$den<- density.ppp(p, sigma=500, at="points")
```

Finally we can select the point with the highest density value and plot it in a different color:

```
max<-bcn_party_p[which.max(bcn_party_p@data$den),]
plot(bcn_party_p, axes=TRUE)
plot(max, axes=TRUE, add=TRUE, col="red")
```


785000
775000
765000
755000
1050000
1060000
1070000
1080000
1090000
1100000
1110000