Practica final: Análisis de datos geoespaciales

Ricardo Garcia Ruiz

3 de julio, 2018

Contents

1 Práctica 1: Análisis de datos geoespaciales **1**
1.1 Enunciado . . . 1

2 Ejercicio 1: Análisis geocodificado de tweets **1**
2.1 Desarrollo . . . 1
2.2 Resolución Ejercicio 1 . . . 1
2.2.1 Carga de los elementos de acceso a Twitter . . . 1
2.2.2 Acceso a Twitter desde R para carga de twitts . . . 2
2.2.3 Tratamiento de los datos descargados de Twitter . . . 2
2.2.4 Búsqueda de los datos geográficos de los twitts: latitud y longitud . . . 4
2.2.5 Generación de shapefile y csv para tratamiento de datos . . . 7
2.2.6 Presentación preliminar mediante R de un mapa de los puntos . . . 7
2.2.7 Carga del fichero shapefile: paises_generalizados . . . 8
2.2.8 Guardamos el conjunto de datos en formato SHP: exportación para uso en ArcGIS . . . 8

3 Ejercicio 2: Generación de mapas de contaminación **8**
3.1 Desarrollo . . . 9
3.2 Resolución Ejercicio 2 . . . 9
3.2.1 Descarga de la información indicada . . . 9
3.2.2 Lectura de los ficheros de datos con R . . . 9
3.2.3 Cálculo de las coordenadas geográficas de localización de las estaciones a partir de las coor-denadas sexagesimales codificadas en los datos originales . . . 10
3.2.4 Asociación de los datos de contaminación con la localización geográfica de cada estación 11
3.2.5 Carga del shapefile, creación del raster y del sistema poligonal . . . 11
3.2.6 Creación del mapa mediante IDW . . . 11
3.2.7 Creación del mapa mediante krigeaje . . . 12

1 Práctica 1: Análisis de datos geoespaciales

1.1 Enunciado

Este documento recoge la práctica de la asignatura **"Análisis de Datos Geoespaciales"**. Se plantean dos ejercicios independientes; ambos ejercicios deben resolverse de forma individual y autónoma por el estudiante. Para la resolución de los ejercicios podrán emplearse como apoyo los materiales del aula y contenidos de internet.

2 Ejercicio 1: Análisis geocodificado de tweets

El análisis geográfico de las aportaciones de usuarios de redes sociales como Twitter o Facebook ha ido ganando relevancia en los últimos años como herramienta para el estudio de fenómenos socioculturales. En el ejercicio se plantea el uso de la red social Twitter.

El objetivo del ejercicio es geocodificar, a nivel país, la procedencia de los usuarios que han utilizado un hashtag o etiqueta determinada (a elegir por el estudiante). A partir de los tweets individuales obtenidos de la red social, se propone la generación de un mapa que muestre el número de tweets por país. El mapa, tematizado convenientemente, será publicado en la aplicación GIS en la nube ArcGIS Online.

2.1 Desarrollo

Se propone al estudiante seleccionar un hashtag (por ejemplo "#Eurovision2018") y realizar una búsqueda de tweets que contengan la etiqueta seleccionada. Para ello, se propone, en el Anexo I del presente documento, un método de geocodificación de los usuarios autores de los tweets. Una vez geocodificados los usuarios (a partir de la información de localización asociada a cada perfil de usuario), se realizará un análisis geográfico que contabilizará el número de usuarios por país. Se propone utilizar la misma capa (shapefile) de países que se empleó en la PEC1 y disponible en el aula (Fichero PAISES_GENERALIZADOS.rar). Todo éste proceso será desarrollado utilizando R.

El shapefile resultante de éste análisis se subirá a la plataforma ArcGIS Online y se tematizará convenientemente. Una vez finalizado, el mapa será compartido con el profesor (jde_diegoa@uoc.edu). Se incluirá también la URL del mapa en el documento de entrega que se subirá al Registro de Evaluación Continua del Aula. Nota: Ha de tenerse en cuenta que en este ejercicio se utilizarán APIs de acceso gratuito pero limitado. Se busca que el estudiante obtenga al menos 500 resultados válidos geocodificados a nivel país (como mínimo).

2.2 Resolución Ejercicio 1

2.2.1 Carga de los elementos de acceso a Twitter

En primer lugar obtenemos las claves de acceso a Twitter según el manual de procedimiento de la práctica.

Una vez adquiridas procedemos a cargarlas en variables en R para poder aplicarlas a una función de acceso a Twitter desde el entorno de RStudio:

```
# variables de acceso a Twitter mediante R a geoTwitterRussia2018
consumer_key <- "Djv1QIHp5DCTtFZDF4P7PmKhe"
consumer_secret <- "PXwhHVALeuGnH5Myn4qLzwfM0YUHCX67Ife8tlujhzAGI0lwYe"
access_token <- "287839706-VlvixPev6CuRnMwAlNn5U7VeVEafgh3JMWCHqUvL"
access_secret <- "csorAvqaLS82yU2hhJj5pZvBHPNQtOOVd04ZV7gwJWTWH"
```

Ahora procedemos a ajustar las funciones de handshake de autenticación OAuth del paquete httr para una sesión twitteR. Esto lo hacemos mediante la siguiente función de carga:

```
# Esta función ajusta las funciones de handshake de autenticación
# OAuth del paquete httr para una sesión twitteR
setup_twitter_oauth(consumer_key,consumer_secret, access_token, access_secret)
```

```
## [1] "Using direct authentication"
```

En el momento de ejecutar esta función se nos permite optar por ajuste del OAuth permanentemente o bien hacerlo por cada sesión. Nosotros optamos por el modo permanente.

2.2.2 Acceso a Twitter desde R para carga de twitts

A continuación, procedemos a delimitar el tipo de hashtag que vamos a utilizar. Aunque hemos estado dudando entre utilizar '#Russia2018' o bien combinado con '#WorldCup', finalmente optamos solo por la versión sencilla:

```
# Hashtag definitivo
searchTerm <- "#Russia2018"
```

```
# Número máximo de twitts
searchResults <- searchTwitter(searchTerm, n = 15000)
```

Además, como se ve, se pide la carga de 15000 twitts máximo.

2.2.3 Tratamiento de los datos descargados de Twitter

A continuación convertimos la lista con los resultados obtenidos en un data.frame manejable en R:

```
# convierte la lista de resultados searchResult en un dataframe
tweetFrame <- twListToDF(searchResults)
```

Table 1: Detalle del dataframe tweetFrame

	text	favorited	favoriteCount	replyToSN	created	truncated
	Length:15000	Mode :logical	Min. : 0,000	Length:15000	Min. :2018-06-28 15:20:36	Mode :logical
	Class :character	FALSE:15000	1st Qu.: 0,000	Class :character	1st Qu.:2018-06-28 17:38:44	FALSE:11157
	Mode :character	NA	Median : 0,000	Mode :character	Median :2018-06-28 20:56:39	TRUE :3843
	NA	NA	Mean : 1,421	NA	Mean :2018-06-29 01:08:13	NA
	NA	NA	3rd Qu.: 0,000	NA	3rd Qu.:2018-06-29 09:29:42	NA
	NA	NA	Max. :2420,000	NA	Max. :2018-06-29 18:31:38	NA

Table 2: Detalle del dataframe tweetFrame

	replyToUID	statusSource	screenName	retweetCount	isRetweet	retweeted	longitu
	Length:15000	Length:15000	Length:15000	Min. : 0,0	Mode :logical	Mode :logical	Length:15
	Class :character	Class :character	Class :character	1st Qu.: 0,0	FALSE:7073	FALSE:15000	Class :charac
	Mode :character	Mode :character	Mode :character	Median : 2,0	TRUE :7927	NA	Mode :charac
	NA	NA	NA	Mean : 125,5	NA	NA	
	NA	NA	NA	3rd Qu.: 39,0	NA	NA	
	NA	NA	NA	Max. :23244,0	NA	NA	

En este momento generamos una lista de los usuarios que han emitido twits sobre los datos obtenidos en **tweetFrame**:

```
# lista (userInfo) a partir del listado de usuarios asociados a los tweets anteriores

# como calcularla completa da un error persistente se calcula por
# tramos y luego se une en una sola lista
userInfo <- lookupUsers(tweetFrame$screenName[1:5000])
userInfo2 <- lookupUsers(tweetFrame$screenName[5001:10000])

# Da error:
# Error in twInterfaceObj$doAPICall(paste("users", "lookup", sep = "/"),  :
#                                   Forbidden (HTTP 403).
#userInfo3 <- lookupUsers(tweetFrame$screenName[10001:15000])

userInfo3 <- lookupUsers(tweetFrame$screenName[10001:13900])
userInfo4 <- lookupUsers(tweetFrame$screenName[13450:15000])
```

```
# fusionamos los datos en una sola lista
userInfo = c(userInfo, userInfo2, userInfo3, userInfo4)

# borramos las variables no necesarias
rm(userInfo2, userInfo3, userInfo4)
```

```
# salvamos la lista obtenida en el sistema
save(userInfo, file = "userInfo.RData")
```

Ahora pasamos a convertir esta lista en un dataframe que podamos utilizar con mayor comodidad en los cálculos posteriores:

```
# Convertimos la lista en dataFrame
userFrame <- twListToDF(userInfo)
```

Table 3: Detalle del dataframe userFrame

	description	statusesCount	followersCount	favoritesCount	friendsCount	url	
	Length:12117	Min. : 0	Min. : 0	Min. : 0	Min. : 0	Length:12117	Length:
	Class :character	1st Qu.: 1298	1st Qu.: 113	1st Qu.: 242	1st Qu.: 170	Class :character	Class :char
	Mode :character	Median : 5965	Median : 370	Median : 1424	Median : 440	Mode :character	Mode :char
	NA	Mean : 27078	Mean : 11816	Mean : 10191	Mean : 1167	NA	
	NA	3rd Qu.: 23036	3rd Qu.: 1253	3rd Qu.: 6745	3rd Qu.: 1022	NA	
	NA	Max. :2187550	Max. :10865141	Max. :681598	Max. :245943	NA	

Table 4: Detalle del dataframe userFrame

	protected	verified	screenName	location	lang	id	listedC
	Mode :logical	Mode :logical	Length:12117	Length:12117	Length:12117	Length:12117	Min. :
	FALSE:12108	FALSE:11745	Class :character	Class :character	Class :character	Class :character	1st Qu.:
	TRUE :9	TRUE :372	Mode :character	Mode :character	Mode :character	Mode :character	Median :
	NA	NA	NA	NA	NA	NA	Mean : 6
	NA	NA	NA	NA	NA	NA	3rd Qu.: 2
	NA	NA	NA	NA	NA	NA	Max. :2150

Finalmente, filtramos el dataframe de forma que mantengamos solo los datos de los usuarios de los que podamos obtener su localización geográfica:

```
# Filtramos el dataFrame obteniendo el listado de todos aquellos usuarios
# que introdujeron información relativa a su localización
addresses <- userFrame[userFrame$location !="",]
```

Table 5: Detalle del dataframe addresses

	description	statusesCount	followersCount	favoritesCount	friendsCount	url	
	Length:8724	Min. : 0	Min. : 0	Min. : 0	Min. : 0	Length:8724	Length
	Class :character	1st Qu.: 1679	1st Qu.: 155	1st Qu.: 293	1st Qu.: 208	Class :character	Class :char
	Mode :character	Median : 7053	Median : 458	Median : 1664	Median : 516	Mode :character	Mode :char
	NA	Mean : 27130	Mean : 13672	Mean : 10321	Mean : 1300	NA	
	NA	3rd Qu.: 24709	3rd Qu.: 1459	3rd Qu.: 7296	3rd Qu.: 1164	NA	
	NA	Max. :1195425	Max. :10865141	Max. :644352	Max. :245943	NA	

Table 6: Detalle del dataframe addresses

	protected	verified	screenName	location	lang	id	listedC
	Mode :logical	Mode :logical	Length:8724	Length:8724	Length:8724	Length:8724	Min. :
	FALSE:8717	FALSE:8426	Class :character	Class :character	Class :character	Class :character	1st Qu.:
	TRUE :7	TRUE :298	Mode :character	Mode :character	Mode :character	Mode :character	Median :
	NA	NA	NA	NA	NA	NA	Mean : 6
	NA	NA	NA	NA	NA	NA	3rd Qu.: 2
	NA	NA	NA	NA	NA	NA	Max. :1172

2.2.4 Búsqueda de los datos geográficos de los twitts: latitud y longitud

Gestionamos la geo localización de longitud y latitud de los twitts que tienen informado correctamente (o al menos que se pueda utilizar en la localización) el pais/localidad en el dataframe 'addresses'.

Para ello consultamos la dirección de internet http://nominatim.openstreetmap.org ya que otras alternativas eran menos fiables para consultar todos los twitts de nuestra lista.

De nuestro dataframe utilizamos el campo 'location' y utilizando la función geocode_OSM() nos da como retorno la longitud y latitud de cada twitt:

```
# Creamos un dataframe vacio para guardar todos los datos finales obtenidos
geocoded_02 <- data.frame()

# Manejamos un loop en el que accederemos a la función geocode_OSM()
# suministrandoles uno a uno los parámetros
for( i in 1:nrow(addresses) ){

  # Creamos una dataframe estandar para los datos que necesitamos obtener
  answer <- data.frame(lat=NA, long=NA, screenName = NA)

  # llamamos a la función geocode_OSM() pero capturando los eventos de
  # warnings que se puedan producir
  result = try(
    geocode_OSM(addresses$location[i],
                projection = "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs",
                return.first.only = TRUE,
                details = FALSE,
                as.data.frame = NA,
                as.SPDF = FALSE,
                server = "http://nominatim.openstreetmap.org")
  )

  # si no se ha producido un warning de tipo "try_error", si se produce se descarta
  # el usuario
  if(class(result) != "try-error"){

    # Si el valor obtenido en resultado fuera nulo se descarta
    if(!is.null(result)){

      answer$lat  = result$coords[2]    # latitud
      answer$long = result$coords[1]    # longitud
      answer$screenName = addresses$screenName[i]   # usuario
```

```
      # añadimos los datos a nuestro dataframe de salida
      geocoded_02 <- rbind(geocoded_02, answer)

    }
  }
  # en este punto manejamos el tiempo de retardo para cada llamada
  # a http://nominatim.openstreetmap.org, ya que si las llamadas no tienen
  # un retrado de al manos un segundo, la pagina de nominatim nos expulsa
  # y corta la comunicación y las consultas. Se introduce un retraso de
  # 1.2 segundos por precaución
  Sys.sleep(1.2)
}
```

Manejamos un loop en el que accederemos a la función **rev_geocode_OSM()** suministrandoles uno a uno los parámetros par obtener el pais, ya que no tenemos en la petición original los datos de los paises.

Creamos un nuevo dataframe, geo_paises_OSM, que sera el que contenga los datos finales con latitud, longitud y país.

```
# Dataframe final con los datos completos que utilizaremos
geo_paises_OSM = data.frame(lat = double(),
                            long = double(),
                            screenName = character(),
                            pais = character(),
                            code = character(),
                            stringsAsFactors = FALSE)

for( i in 1:nrow(geocoded_02) ){

  # llamamos a la función rev_geocode_OSM() pero capturando los eventos de
  # warnings que se puedan producir

  # creamos un objeto SpatialPoints para suministrarlo a la función
  # rev_geocode_OSM() con formato long:lat
  sp01 = SpatialPoints(coords = cbind(geocoded_02$long[i], geocoded_02$lat[i]),
                       proj4string = CRS("+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs"))
  result = try(
    rev_geocode_OSM(x=sp01, server = "http://nominatim.openstreetmap.org")
  )

  # si no se ha producido un warning de tipo "try_error", si se produce se descarta
  # el usuario
  if(class(result) != "try-error"){

    # Si el valor obtenido en resultado fuera nulo se descarta
    if(!is.null(result)){

      # añadimos los datos a nuestro dataframe de salida
      geo_paises_OSM <- rbind(geo_paises_OSM,
                              data.frame(geocoded_02$lat[i],
                                         geocoded_02$long[i],
                                         geocoded_02$screenName[i],
                                         result[[1]]$country,
                                         result[[1]]$country_code))
    }
```

```
  } else{
    # añadimos los datos a nuestro dataframe de salida
    geo_paises_OSM <- rbind(geo_paises_OSM,
                            data.frame(NULL,NULL,NULL, NULL, NULL))
  }
  # en este punto manejamos el tiempo de retardo para cada llamada
  # a http://nominatim.openstreetmap.org, ya que si las llamadas no tienen
  # un retrado de al manos un segundo, la pagina de nominatim nos expulsa
  # y corta la comunicación y las consultas. Se introduce un retraso de
  # 1.2 segundos por precaución
  Sys.sleep(1.2)
}

# colocamos los nombres correctos en el dataframe
colnames(geo_paises_OSM) <- c("lat", "long", "screenName", "country", "country_code")
geo_paises_OSM$country_code <- toupper(geo_paises_OSM$country_code)
```

```
# salvamos la lista obtenida en el sistema
saveRDS(geo_paises_OSM, file = "geo_paises_OSM.rds",
        compress = "bzip2")
```

```
# salvamos la lista obtenida en el sistema
saveRDS(geocoded, file = "geocoded.rds",
     compress = "bzip2")
```

```
# leemos la lista compilada directamente (RMarkdown no lo permite de otra forma)
geocoded <- readRDS(file = "geo_paises_OSM.rds")
```

Con este procedimiento se consiguen una cantidad de 6073 registros para el procesamiento gráfico y geográfico.

2.2.5 Generación de shapefile y csv para tratamiento de datos

En este paso procedemos a guardar el conjunto de datos obtenido en 2 formatos: RDS y CVS.

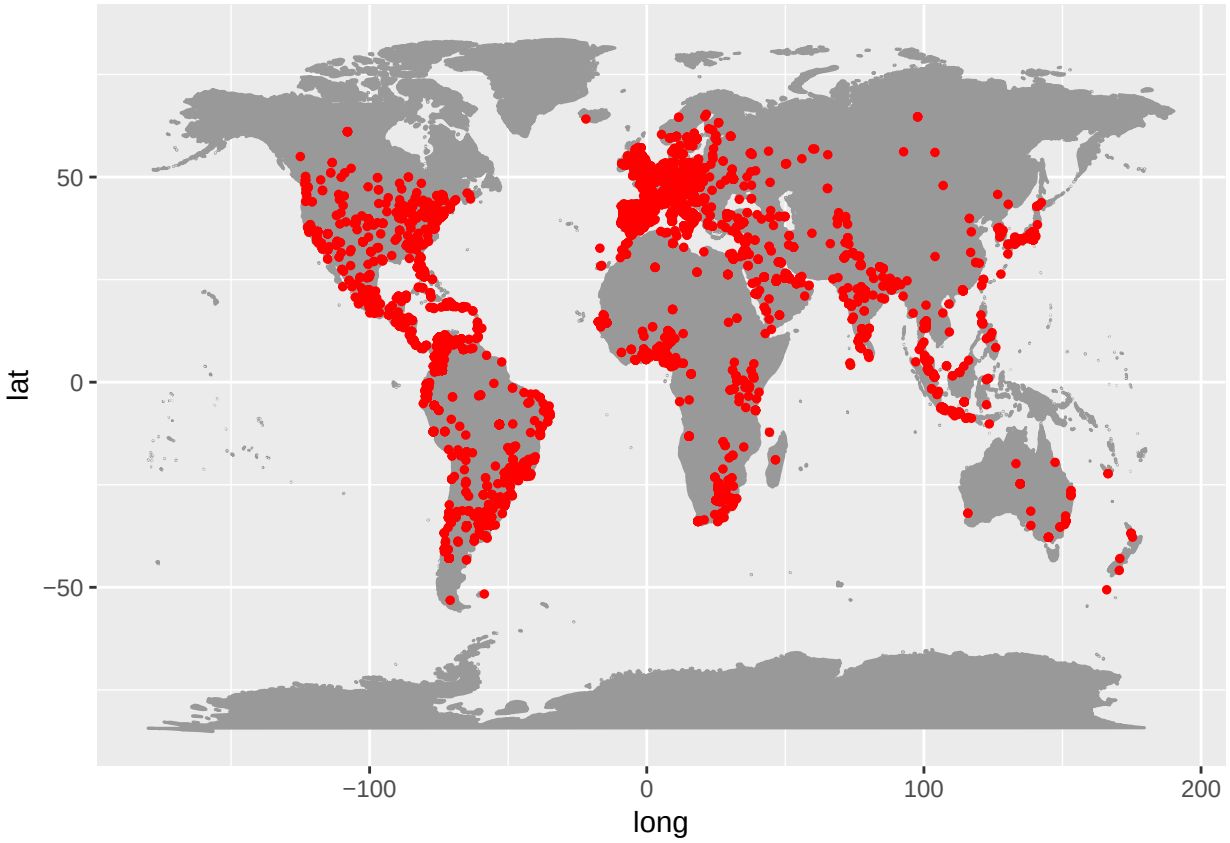
```
# En este punto pasamos a salvar en formato RDS
saveRDS(geocoded,"twitter_users_geocoded2.rds")

# En este punto pasamos a salvar en formato csv
write.table(geocoded, file="twitter_users_geocoded2.csv",
            sep=",",
            row.names=FALSE)
```

Estos formatos los utilizaremos posteriormente para la presentación del mapa en ArcGis.

2.2.6 Presentación preliminar mediante R de un mapa de los puntos

```
# De manera general creamos un mapa de visualización del conjunto de datos
# de twitts con sus coordenadas establecidas
mapaMundi <- borders("world", colour="gray60", fill="gray60")
mapa <- ggplot() + mapaMundi
mapa <- mapa + geom_point(aes(x=geocoded$long, y=geocoded$lat),
                          color="red", size=1)
mapa
```



2.2.7 Carga del fichero shapefile: paises_generalizados

En este paso procedemos a la carga de los datos del shapefile # paises generaliados para realizar los cómputos posteriores:

```
# Carga del fichero
paises_generalizados <- readOGR("./PAISES_GENERALIZADOS/paises_generalizados.shp")

# Convierte el dataframe geocoded_02 en un Spatial object
#coordinates(geocoded_02) <- 2:1
coordinates(geocoded) <- 2:1

# formato del objeto spatial
crs_geograficas ='+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs'
#geocoded_02@proj4string <-CRS(crs_geograficas)
geocoded@proj4string <-CRS(crs_geograficas)
paises_generalizados@proj4string <- CRS(crs_geograficas)

# Se crea un nuevo atributo 'contador' y lo inicializamos a 1
geocoded$contador <- 1
usuarios.pais <- aggregate(x=geocoded,
                           by=paises_generalizados,
                           FUN= length)
```

2.2.8 Guardamos el conjunto de datos en formato SHP: exportación para uso en ArcGIS

Finalmente, procedemos a exportar el objeto **'paises_generalizados'** a un shapefile para poder utilizarlo en la plataforma ArcGIS

```
writeOGR(paises_generalizados,
         "./paises_OGM/paises_twitter.shp",
         driver="ESRI Shapefile",
         layer="paises")
```

El resultado se encuentra en la dirección web: https://arcg.is/aWWvL

Tanto el código fuente como los resultados de la utilización del mismo se encuentran en un repositorio en **GitHub**, en la dirección: https://github.com/rgarciarui/geoTwitterRussia2018

3 Ejercicio 2: Generación de mapas de contaminación

En los últimos años, la contaminación atmosférica se está convirtiendo en un grave problema en los entornos urbanos. En ciudades como Madrid, los altos niveles de contaminación están provocando un grave problema de salud pública. El Ayuntamiento de la ciudad ha decidido tomar una serie de polémicas medidas, que afectan sobre todo a la circulación de vehículos privados, con el fin de reducir la emisión de estos contaminantes y en especial del dióxido de nitrógeno (NO2). El objetivo del ejercicio es la generación de dos mapas, uno utilizando el interpolador IDW y otro utilizando krigeaje, del máximo nivel de contaminación diaria para NO2 en la ciudad de Madrid. Durante el ejercicio se utilizará R como entorno para el tratamiento de los datos.

3.1 Desarrollo

Para el desarrollo del ejercicio utilizaremos los siguientes conjuntos de datos: * Datos en tiempo real de calidad del aire: https://goo.gl/BMLJ8X. Utilizaremos el archivo etiquetado como **"Calidad del aire. Tiempo real csv"**. Tened en cuenta que este fichero se actualiza cada hora. La descripción del conjunto de datos puede encontrarse en https://goo.gl/tBnK1o.

- Estaciones de control: https://goo.gl/AuCxvu. Utilizaremos el fichero etiquetado como **"Calidad del aire: estaciones de control"**, en formato xls. Las coordenadas se encuentran codificadas como coordenadas geográficas sexagesimales, con datum de referencia WGS84.
- Límites geográficos de los distritos de Madrid: https://goo.gl/SdZiH1. Utilizaremos el fichero etiquetado como **"Distritos en formato geográfico en ETRS89"**. El fichero shapefile contiene los límites municipales en sistema de coordenadas EPSG:25830.

El procedimiento propuesto incluye los siguientes pasos:

- Descarga de la información indicada.
- Lectura de los ficheros de datos con R.
- Cálculo de las coordenadas geográficas de localización de las estaciones a partir de las coor-denadas sexagesimales codificadas en los datos originales.
- Obtención del valor máximo horario de contaminación por NO2 para cada estación (los datos de dióxido de nitrógeno vienen codificados con el valor 8 en el campo magnitud).
- Asociación de los datos de contaminación con la localización geográfica de cada estación.
- Reproyección de la capa de límites municipales y obtención de un polígono único con el límite del municipio.
- Elaboración de los mapas requeridos. En ambos casos se podrán utilizar diferentes juegos de parámetros.

3.2 Resolución Ejercicio 2

3.2.1 Descarga de la información indicada

Se procede a la descarga de los 3 ficheros que se indican en la práctica:

- **"Calidad del aire. Tiempo real csv"**
- **"Calidad del aire: estaciones de control"**
- **"Distritos en formato geográfico en ETRS89"**

Estos fichero se guadan en el directorio 'contaminacion', desde donde se operará el código y los datos.

3.2.2 Lectura de los ficheros de datos con R

En primer lugar tratamos el fichero **"Calidad del aire. Tiempo real csv"**.

```
# Carga del fichero "Calidad del aire. Tiempo real csv"
medidas <- read.csv('./contaminacion/horario.csv',header=TRUE,sep=';')

# Se filtran en una nueva variable unicamente los datos de
# dioxido de nitrogeno (valor = 8)
medidas.no2 <- medidas[medidas$MAGNITUD==8,]

# Se añade una nueva variable y se inicializa sin valor
medidas.no2$maxno2 <-NA


for (i in 1:nrow(medidas.no2)){
  x<-c() # Vector c inicializado vacío

  # añade todos los valores de esas columnas para cada fila al vector (x)
  for (j in 1:24){
    # sprintf() genera los nombres de las columnas "H01","H02",...,"H24".
    x[j]<-medidas.no2[i,paste(sprintf("H%02d", j)) ]
  }

  # calcula y añade a la variable 'maxno2' el maximo de cda estación
  medidas.no2$maxno2[i]<- max(x)
}

summary(medidas.no2$maxno2)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   16,00   40,75   51,50   53,46   65,00  102,00
```

3.2.3 Cálculo de las coordenadas geográficas de localización de las estaciones a partir de las coor-denadas sexagesimales codificadas en los datos originales

En segundo lugar realizamos el tratamiento del fichero con los datos de **"Calidad del aire: estaciones de control"**.

Realizamos el cambio a coordenads decimales desde las sexagesimales en las que se encuentran en el fichero.

```
# carga de datos
estaciones <- read_excel('./contaminacion/informacion_estaciones_red_calidad_aire.xls',
```

```
                          col_names=TRUE,
                          skip=4,
                          n_max=25)

# bucle que convierte las coordenadas sexagesimales a decimales
for (i in 1:nrow(estaciones)){
  long <- unlist(strsplit(estaciones$LONGITUD[i],split=c(" ")))
  long[3]<- gsub(c("\"O"),"",long[3])
  long[3]<- gsub(c("'O"),"",long[3])
  long[3]<- gsub(c("'O"),"",long[3])
  estaciones$long[i] <- -1 *( as.integer(gsub("º","",long[1])) +
                              (as.double(gsub("'","",long[2])))/60 +
                              as.double(gsub(",",".",long[3]))/1200)
  lat <- unlist(strsplit(estaciones$LATITUD[i],split=c(" ")))
  lat[3]<- gsub(c("'N"),"",long[3])
  estaciones$lat[i] <- as.integer(gsub("º","",lat[1])) +
  (as.double(gsub("'","",lat[2])))/60 +
  as.double(gsub(",",".",lat[3]))/1200
  }
```

```
## Warning: Unknown or uninitialised column: 'long'.
```

```
## Warning: Unknown or uninitialised column: 'lat'.
```

3.2.4 Asociación de los datos de contaminación con la localización geográfica de cada estación

En este punto procedemos a unir los 2 dataframes y del dataframe final simplificamos las variables que nos van a ser útiles para cálculos posteriores y realizamos una segunda selección y simplificación:

```
# En este punto procedemos a unir los 2 dataframes y del dataframe final
# simplificamos las variables que nos van a ser útiles para cálculos
# posteriores y realizamos una segunda selección y simplificación

# Creamos un nuevo dataframe con la fusión de los conjuntos de datos:
# - medidas.no2
# - estaciones
# Procedemos a su fusión por los atributos:
# - medidas.no2$ESTACION
# - estaciones$NÚMERO
#
medidas.geo <- merge(medidas.no2,estaciones, by.x="ESTACION", by.y="NÚMERO")

# Procedemos a simplicar el dataframe, tomando solo los atributos necesarios
medidas.geo.simp <- medidas.geo[,c("ESTACION","maxno2","long","lat")]
```

3.2.5 Carga del shapefile, creación del raster y del sistema poligonal

En esta fase procedemos a la carga de los datos del shapefile y relizamos la creación del raster para poder utilizarlo en la creación de los mapas.

```
limites <-readOGR("./contaminacion/DISTRITOS_ETRS89/DISTRITOS_20151002.shp")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "F:\Users\Ricardo\Documents\Trabajo y Becas\R-projects\geoTwitterRussia2018\contaminacion\DI
```

```
## with 21 features
## It has 4 fields
```

```
limites.wgd84 <- spTransform(limites, CRS("+proj=longlat +datum=WGS84"))
limite.madrid <- gUnaryUnion(limites.wgd84)
limite.raster <- raster(limite.madrid, res=0.001)
```

Ahora procedemos a trasnformar el dataframe **'medidas.geo.simp'** en un SpatialPolygons dataframe:

```
coordinates(medidas.geo.simp) <- 3:4
medidas.geo.simp@proj4string <- limite.madrid@proj4string
```

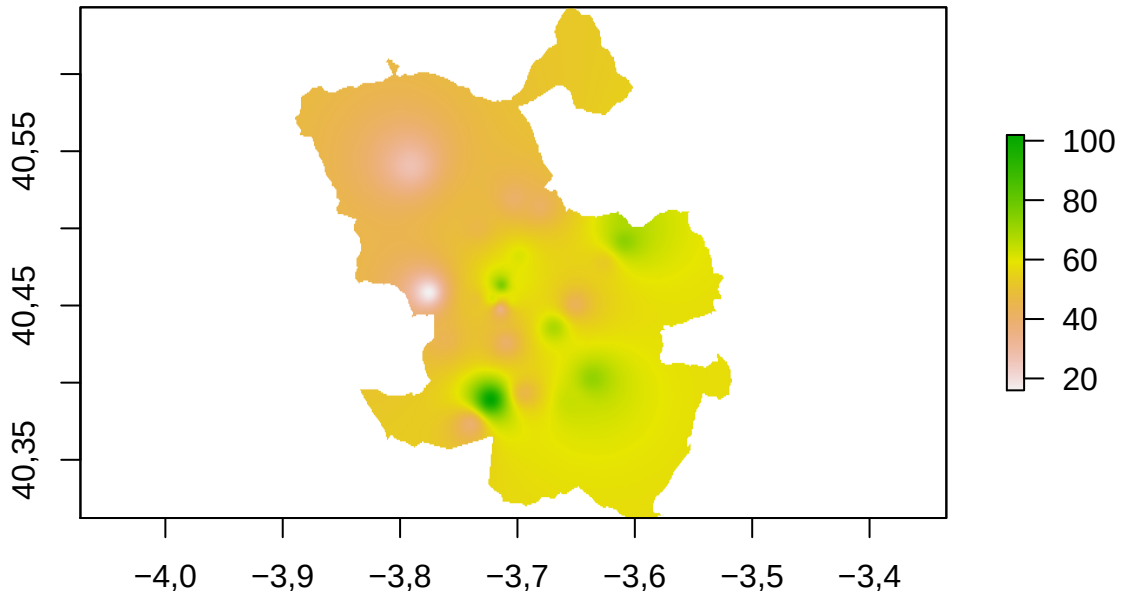
3.2.6 Creación del mapa mediante IDW

En esta siguiente fase procedemos a la construcción del objeto de visualización mediante IDW:

```
# interpolación usando IDW
gstat.parametros <- gstat(formula = maxno2~1,
                          locations=medidas.geo.simp,set = list(idp = 2))
no2.idw <- interpolate(limite.raster, gstat.parametros)
```

```
## [inverse distance weighted interpolation]
```

```
no2.idwr <- raster::mask(no2.idw,limite.madrid)
plot(no2.idwr)
```



3.2.7 Creación del mapa mediante krigeaje

Con el fin de conseguir el mapa interpolado por krigeaje, en primer lugar obtenmos el semivariograma. Aplicamos el valor 0.5 para el parámetro width y obtenemos el siguiente variograma teórico:

```
# interpolado por krigeaje
gstat.parametros <- gstat(formula=maxno2~1,
                          locations=medidas.geo.simp)
variograma <- variogram(gstat.parametros, width=0.5)
var.teorico <-fit.variogram(variograma, vgm(c("Exp", "Ste", "Sph","Mat","Gau","Spl")))
```

```
## Warning in fit.variogram(object, x, fit.sills = fit.sills, fit.ranges =
## fit.ranges, : No convergence after 200 iterations: try different initial
## values?
```

```
## Warning in fit.variogram(object, model, fit.sills = fit.sills, fit.ranges
## = fit.ranges, : No convergence after 200 iterations: try different initial
## values?
```

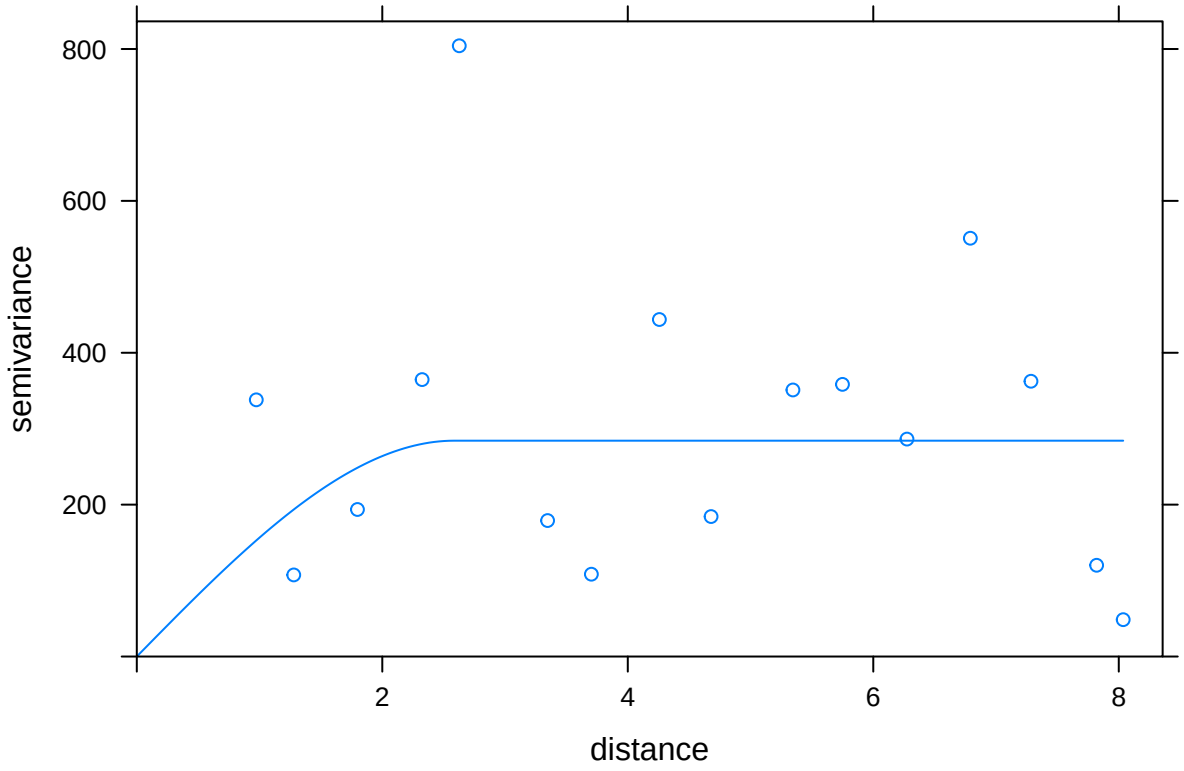
```
## Warning in fit.variogram(object, x, fit.sills = fit.sills, fit.ranges =
## fit.ranges, : singular model in variogram fit
```

```
## Warning in fit.variogram(object, x, fit.sills = fit.sills, fit.ranges =
## fit.ranges, : singular model in variogram fit
```

```
var.teorico
```

```
##   model    psill    range
## 1   Nug   0,0000 0,000000
## 2   Sph 284,2183 2,586605
```

```
plot(variograma, var.teorico)
```

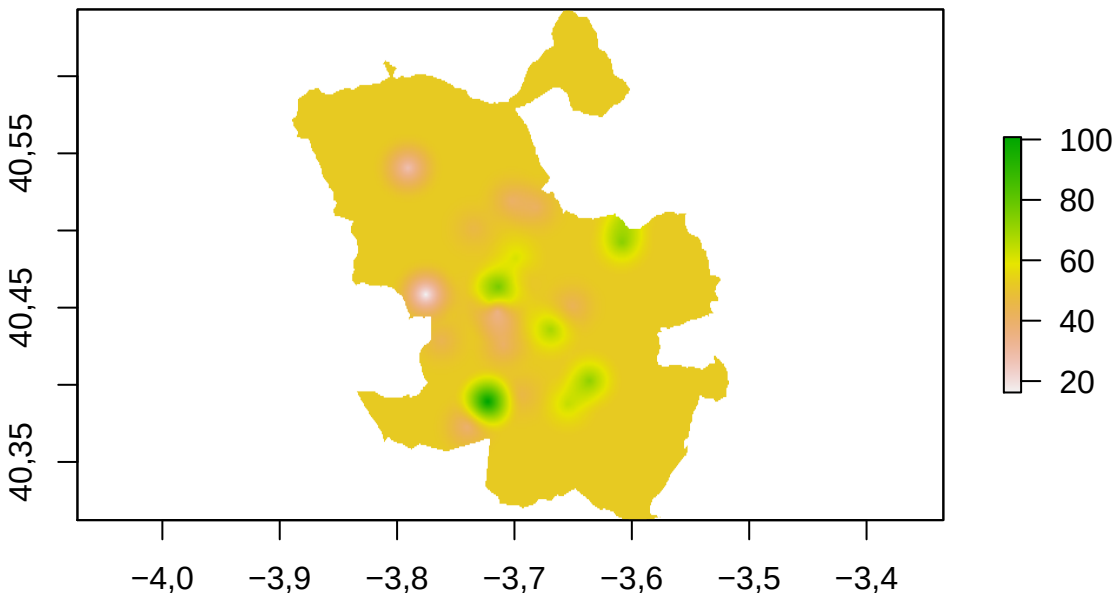


A continuación generamos el mapa de interpolación final:

```
# mapa interpolado
no2.ordkg <- krige(formula=maxno2~1,
                   locations=medidas.geo.simp,newdata<- as(limite.raster,"SpatialGrid"),
                   model=var.teorico)
```

```
## [using ordinary kriging]
```

```
no2.okgr <- raster::mask(raster(no2.ordkg),limite.madrid)
plot(no2.okgr)
```


40,55
40,45
40,35
−4,0
−3,9
−3,8
−3,7
−3,6
−3,5
−3,4
100
80
60
40
20