Ciencia de datos geográficos

Silvia Laceiras
Felipe Sodré M. Barros

Fabián Rechberger

Algoritmos de agrupacion (clustering)!

trabajando para una empresa del sector imobiliário de Rio de Janeiro que quiere invertir en departamentos de "alta gama" en el barrio de Copacabana.

Pero copacabana es un barrio bastante grande. Por eso nos pide para identificar el patrón espacial de los departamentos disponibles en Airbnb

Identificación de patrones

Una de las actividades más comunes en análisis de datos es la identificación de patrones.

Al trabajarmos con datos geográficos, pasamos a tener la necesidad de identificar patrones espaciales.

Aunque existan distintos algoritmos para dicha tarea, vamos a ver que al trabajar con datos espaciales quizás tengamos que considerar algún algoritmo que lleve en consideración el atributo espacial.

Agrupación: ejemplo

Cómo ustedes agruparían los elementos de cada base de datos?

DBSCAN: Density-Based Spatial Clustering and Application with Noise

DBSCAN es un algoritmo de aprendizaje de maquina (`machine learning`) diseñado para identificar agrupamientos ("clusters") espaciales.

Es decir, DBSCAN es capaz de identificar elementos próximos entre si de acuerdo a sus atributos en varias dimensiones, inclusive espacial.

DBSCAN está diseñado para trabajar con datos espaciales, lo que lo diferencíaa de algoritmos como KMeans.

Introducción al DBSCAN

Cómo ustedes agruparían los elementos de cada base de datos?

Una agrupación es un área con gran ocurrencia de datos (densidad elevada) en el espacio, separada por regiones de baja densidad de datos.

DBSCAN y Kmeans

DBSCAN en detalle

Objetivo: El objetivo es identificar regiones de alta densidad de datos, que se pueden medir por la cantidad de objetos cercanos a un punto dado.

Parámetros:

- `epsilon` ("eps"): Define el radio de vecindad alrededor de un punto `x`;
- `puntos mínimos` ("MinPts"): Define el número mínimo de vecinos dentro del radio "eps" para que exista una agrupación;

Conceptos:

Punto central: Cualquier punto `x` en el conjunto de datos, con un conteo de vecinos mayor o igual a `MinPts`, se marca como un punto central;

Punto frontera: Decimos que `x` es un punto de borde, si el número de sus vecinos es menor que `MinPts`, pero pertenece a una agrupación de algún punto central `z`.

Punto ruído: Si un punto no es ni un núcleo ni un punto de borde, entonces se llama un punto de ruído o un valor atípico.

DBSCAN: conceptos

La siguiente figura muestra los diferentes tipos de puntos, usando `MinPts` = 6:

- núcleo;
- borde y;
- puntos atípicos

`x` es un punto central porque posee 6 vecinos internos a `ϵ`;
`y` es un punto de borde porque posee menos de 6 vecinos, pero hace parte de la agrupaión de `x`;
`z` es un punto de ruído: No posee 6 vecinos ni hace parte de ningún grupo;

Manos a la obra!

Como vamos a seguir usando los datos de Rio de Janeiro, vamos a seguir con el proyecto de la clase anterior. Pero vamos a crear un *script* nuevo...

Paquete nuevo: dbscan

Vamos a ocupar el paquele `dbscan`.

Como no lo ocupamos todavía, es probable que tengamos que instalarlo:

```
install.packages("dbscan")
```

Cargando los paquetes necesarios

Tal cual como en la clase anterior, vamos a ocupar los paquetes:

```
library(sf)
library(tmap)
library(tidyverse)
library(dbscan) #nuevo
```

Carga de datos y preparación

Cargando los datos de Airbnb

Vamos a cargar el dato `csv` con la función `read_csv`, del paquete `readr`, que nos brinda `tidyverse`.

```
 datos <- read_csv("./Datos/AirbnbRJRentals_modificado.csv")
```

```
## Rows: 27507 Columns: 16
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## chr   (4): name, host_name, neighbourhood, room_type
## dbl  (11): id, host_id, latitude, longitude, price, minimum_nights, number_o.
## date  (1): last_review
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this messa
```

Dataframe a dato geográfico

Para convertir el `csv` en un dato geográfico tendremos que usar la función `st_as_sf`, informando cuales columnas teienen la información de coordenadas:

Atención: Vamos a informar el Sistema de Referência de Coordenadas (SRC) = 4326 (geográfico)

```
# transformando en geografico
(airbnb <- st_as_sf(
  datos,
  coords = c("longitude", "latitude"),
  crs=4326)
 )
```

```
## Simple feature collection with 27507 features and 14 fields
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: -43.70591 ymin: -23.07284 xmax: -43.1044 ymax: -22.74969
## Geodetic CRS:  WGS 84
## # A tibble: 27,507 × 15
##         id name   host_id host_name neighbourhood room_type price minimum_nig
##  *   <dbl> <chr>    <dbl> <chr>     <chr>         <chr>     <dbl>          <d
```

Proyectando el dato

Como nuestros datos están en formato geográfico, vamos a convertirlos al sistema proyectado (crs 32712), usando la función `st_transform`.

De esa manera, podremos informar y pensar en el parámetro `eps`, del DBSCAN, como una distáncia en metros.

```
airbnb <- st_transform(airbnb, 32723)
```

Limpieza de datos: filtrado

Vamos a ocupar la función `filter`, para limitar a los datos que sean de los barrios "Copacabana" y "Leme" y que estén ofreciendo todo el departamento (`Entire home/apt`):

```
copa <- airbnb %>%
  filter(neighbourhood %in%
           c("Copacabana", "Leme"),
         room_type == "Entire home/apt")
```

Precios del alquiler en Copacabana y Leme

Vamos a identificar la distribución de los valores de precio para definir qué sería "de alta gama":

```
 summary(copa$price)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##    58.0   453.0   700.0   899.3  1071.0  4983.0
```

Filtrando los departamentos por precio

Como queremos los departamentos más caros, podemos usar el tercer cuartil de los precios para identificar los 25% más caros...

```
copa <- copa %>% filter(price >= 1071)
```

Ya podrían hacer un mapa, no?

Primer análsis de agrupación espacial:

Vamos a ejecutar el DBSCAN buscando los grupos en un radio de 100 metros (una cuadra mas o menos) y que tengan al menos 10 otros departamentos:

```
(clusters_spacial_100_10 <-
   dbscan(
     st_coordinates(copa),
     eps = 100,
     minPts = 10))
```

```
## DBSCAN clustering for 1831 objects.
## Parameters: eps = 100, minPts = 10
## Using euclidean distances and borderpoints = TRUE
## The clustering contains 4 cluster(s) and 84 noise points.
##
##    0    1    2    3    4
##   84 1708    7   16   16
##
## Available fields: cluster, eps, minPts, dist, borderPoints
```

Incorporación de los resultados

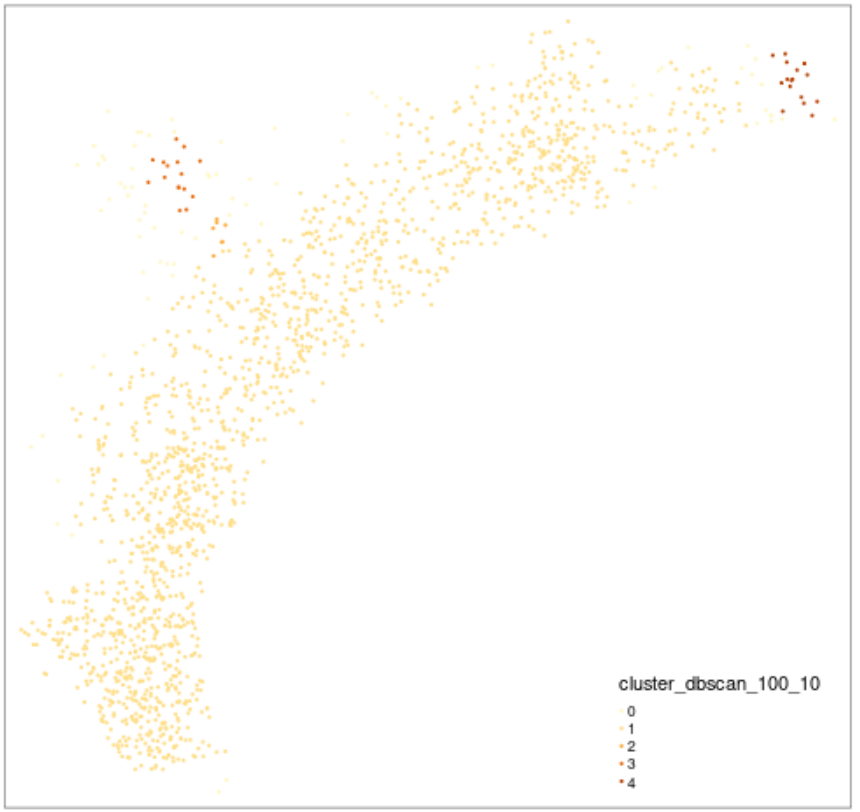
Los agrupamientos fueron creados. Falta agregar la info del grupo al cual el punto hace parta en el atributo del mismo:

Para eso vamos a usar la función `mutate`, que nos permite crear y modificar columnas en conjuntos de datos;

```
copa <- copa %>% mutate(
  cluster_dbscan_100_10 = # nombre de la columna nueva
    clusters_spacial_100_10$cluster # valor a inserir en la columna
  )
```

Mapa del resultado

```
tm_shape(copa) +
  tm_dots(col = "cluster_dbscan_100_10")
```



Tareas Extra

Hacer otro análisis de agrupación cambiando el parámetro de distáncia (*eps*) a 50 metros, agregandolo al dato espacial y representando en forma de mapa

Análsisis de agrupación con el *eps*

50

Y si reduzimos el radio de distáncia a 50 metros?

```
 (clusters_spacial_50_10 <-
    dbscan(
      st_coordinates(copa),
      eps = 50,
      minPts = 10))
```

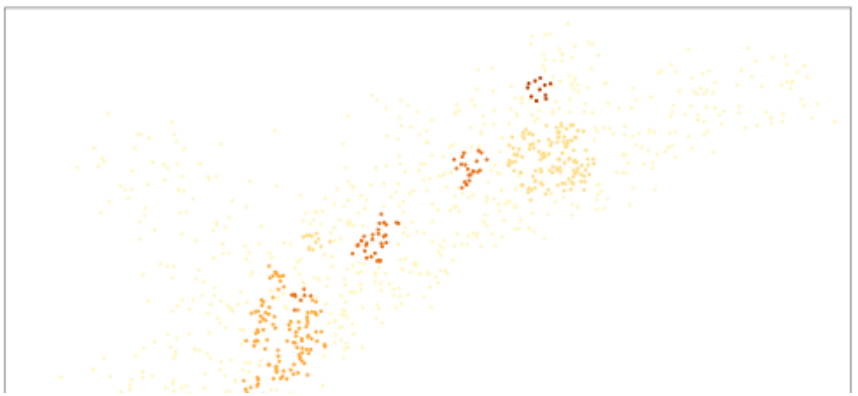
```
## DBSCAN clustering for 1831 objects.
## Parameters: eps = 50, minPts = 10
## Using euclidean distances and borderpoints = TRUE
## The clustering contains 16 cluster(s) and 857 noise points.
##
##   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16
## 857 335 257  28 128  12  12  63  15  28  13  13  19  21  11   8  11
##
## Available fields: cluster, eps, minPts, dist, borderPoints
```

Incorporación de los resultados y mapa

Agregando el dato del grupo a la tabla de atributos y mapa:

```
copa <- copa %>% mutate(
  cluster_dbscan_50_10 =
    clusters_spacial_50_10$cluster
  )

tm_shape(copa) +
  tm_dots(col = "cluster_dbscan_50_10")
```



Para reflexionar: Qué efectos tuvimos al reducir el parámetro de distáncia?

Pero nos fué solicitado hacer un análsisis de agrupación que lleve en consideración no solo el atributo espacial, así como el precio del alquiler.

En resumen, quieren saber si los departamentos más caros sueles ubicarse en una misma región

Consigna: Analizar la ayuda (*help*) de la función *dbscan* e identificar cómo podríamos hacer con que el algoritmos considere también es precio en la identificación de los grupos.

Agrupación considerando el precio

Hasta el momento, el `dbscan` estuvo identificando los grupos por su ubicación. Pero necesitamos que los grupos no sean solamente por el atributo espacial si no que, también, considerando el valor del alquiler.

Para eso debemos usar elparámetro de peso (`weights`) del `dbscan`:

```
(clusters_precio <-
   dbscan(
     st_coordinates(copa),
     eps = 100,
     minPts = 10,
     weights = copa$price))
```

```
## DBSCAN clustering for 1831 objects.
## Parameters: eps = 100, minPts = 10
## Using euclidean distances and borderpoints = TRUE
## The clustering contains 13 cluster(s) and 0 noise points.
##
##    1    2    3    4    5    6    7    8    9   10   11   12   13
## 1812    1    2    1    6    1    1    1    1    1    2    1    1
##
```

Incorporación de los resultados y mapa

```
copa <- copa %>% mutate(
  clusters_precio =
    clusters_precio$cluster
  )

tm_shape(copa) +
  tm_dots(col = "clusters_precio")
```