

# Análisis de patrones de puntos en R

*Rodrigo Tapia McClung*

*Agosto 11, 2017*

## PPA (Point Pattern Analysis)

Queremos estudiar el comportamiento espacial de un conjunto de puntos. En particular, explorar si presentan autocorrelación espacial global y local. Para ello, recordemos cómo se calcula la  $I$  de Moran:

$$I = \frac{\sum_i \sum_{j \neq i} w_{ij}(x_i - \bar{x})(x_j - \bar{x})}{\sum_i \sum_{j \neq i} w_{ij} \sum_i (x_i - \bar{x})^2}$$

Y la versión local se puede escribir como:

$$I_i = \frac{(x_i - \bar{x}) \sum_{j \neq i} w_{ij}(x_j - \bar{x})}{\frac{1}{n} \sum_{j \neq i} (x_j - \bar{x})^2}$$

Los valores  $x$  son nuestras observaciones y los  $w_{ij}$  representan los elementos de una matriz de pesos (renglón  $i$ , columna  $j$ ). Hay que construir esta matriz.

**Requisitos:** tener los SHPs en alguna carpeta y cambiar a ese directorio de trabajo. Por ejemplo:

```
# Cambiar directorio de trabajo
setwd("C:/Descargas/R/practica2")
```

Asegurarse de tener instaladas las librerías que vamos a usar:

```
# Lista de librerías:
list.of.packages <- c("spdep")
# Ver qué no está instalado
new.packages <- list.of.packages[!(list.of.packages %in% installed.packages() |,
  "Package")]
# Si falta algo, instalarlo
if (length(new.packages)) install.packages(new.packages)
```

## Abrir SHPs en R

Usaremos rgdal para abrir dos shapefiles en R (`mpb00.shp` y `BOUNDARY.SHP`)

```
library(rgdal)

## Warning: package 'rgdal' was built under R version 3.3.3
## Loading required package: sp
## Warning: package 'sp' was built under R version 3.3.3
## rgdal: version: 1.2-7, (SVN revision 660)
## Geospatial Data Abstraction Library extensions to R successfully loaded
## Loaded GDAL runtime: GDAL 2.0.1, released 2015/09/15
```

```

## Path to GDAL shared files: C:/Users/rodrigo.tapia/Documents/R/win-library/3.3/rgdal/gdal
## Loaded PROJ.4 runtime: Rel. 4.9.2, 08 September 2015, [PJ_VERSION: 492]
## Path to PROJ.4 shared files: C:/Users/rodrigo.tapia/Documents/R/win-library/3.3/rgdal/proj
## Linking to sp version: 1.2-4

mpb <- readOGR("data", "mpb00")

## OGR data source with driver: ESRI Shapefile
## Source: "data", layer: "mpb00"
## with 8401 features
## It has 5 fields
## Integer64 fields read as strings: POLY_ID
frontera <- readOGR("data", "BOUNDARY")

## OGR data source with driver: ESRI Shapefile
## Source: "data", layer: "BOUNDARY"
## with 1 features
## It has 5 fields
## Integer64 fields read as strings: F_FC_

```

## Analizar un patrón de puntos

Ahora cargamos la biblioteca `spdep`.

```

library(spdep)

## Warning: package 'spdep' was built under R version 3.3.3
## Loading required package: Matrix

```

## Umbral de distancia y matriz de adyacencia

Queremos encontrar el umbral de distancia que garantice que **cada** punto tenga al **menos** un vecino. Para ello, primero encontramos el primer vecino más cercano a cada punto (con `knearneigh`), luego calculamos la distancia (de cada punto a ese vecino más cercano) y por último encontramos la distancia máxima (de todas esas distancias). Uso `k1` como referencia a los *k Nearest Neighbors* (KNN) con `k=1`:

```

# Calcular los vecinos más cercanos y hacer una lista
k1 <- knn2nb(knearneigh(mpb, k = 1, longlat = TRUE))

```

Ahora encontramos la distancia entre cada punto y su vecino más cercano (con `nbdists`), hacemos un vector a partir de esa lista de distancias de los vecinos más cercanos (con `unlist`) y encontramos su máximo (con `max`).

```

# Calcular distancias de los vecinos más cercanos, hacer un vector y encontrar el máximo
distancia <- max(unlist(nbdists(k1, mpb)))

```

```

## [1] 6927.891

```

Con este umbral de distancia, garantizamos que **TODOS** los puntos tendrán por lo menos 1 vecino. Ahora encontramos los vecinos de cada punto a esa distancia.

```

# Encontrar vecinos
vecinos <- dnearneigh(mpb, 0, distancia)

```

Si quieres, puedes ver algo de información de esta lista de vecinos con

```
summary(vecinos)
```

Ahora definimos una lista de pesos para los puntos y sus vecinos. Hay de diferentes tipos: binarias (0,1), estandarizadas por renglón (cada renglón suma 1), estandarizadas por renglón y columna, etc. Claramente, la elección de cómo se construyan los pesos afectará la estadística. Recordar que la definición de la  $I$  usa  $w_{ij}$ .

Vamos a usar el caso típico, que es una matriz binaria estandarizada por renglones, caso W.

```
# Lista de pesos
mpb.lw <- nb2listw(vecinos) # style = "W" estandarizada por renglón
#mpb.lw <- nb2listw(vecinos, style = "B") # pesos binarios
#mpb.lw <- nb2listw(vecinos, style = "C") # estandarizada por renglón globalmente
```

Si quieres ver información acerca de la lista de pesos:

```
mpb.lw
```

## Hacer una *gráfica de Moran*

En una gráfica de dispersión de Moran, graficamos nuestros valores estandarizados contra una variable de retraso espacial, la cual es un promedio de las unidades vecinas (contiguas). Queremos ver una relación en términos de desviaciones estándar alrededor de la media, de modo que transformamos los valores de  $x_i$  en una variable normalizada estandarizada  $z_i = \frac{x_i - \bar{x}}{\sqrt{\frac{1}{n} \sum_i (x_i - \bar{x})^2}}$  que podemos calcular con `scale()`. Necesitamos

un par de variables nuevas:

```
# Estandarizar valores de Z y salvarlos como una nueva columna
mpb$zScaled <- scale(mpb$Z)
# Calcular la variable de retraso y salvarla
mpb$lagZ <- lag.listw(mpb.lw, mpb$Z)
```

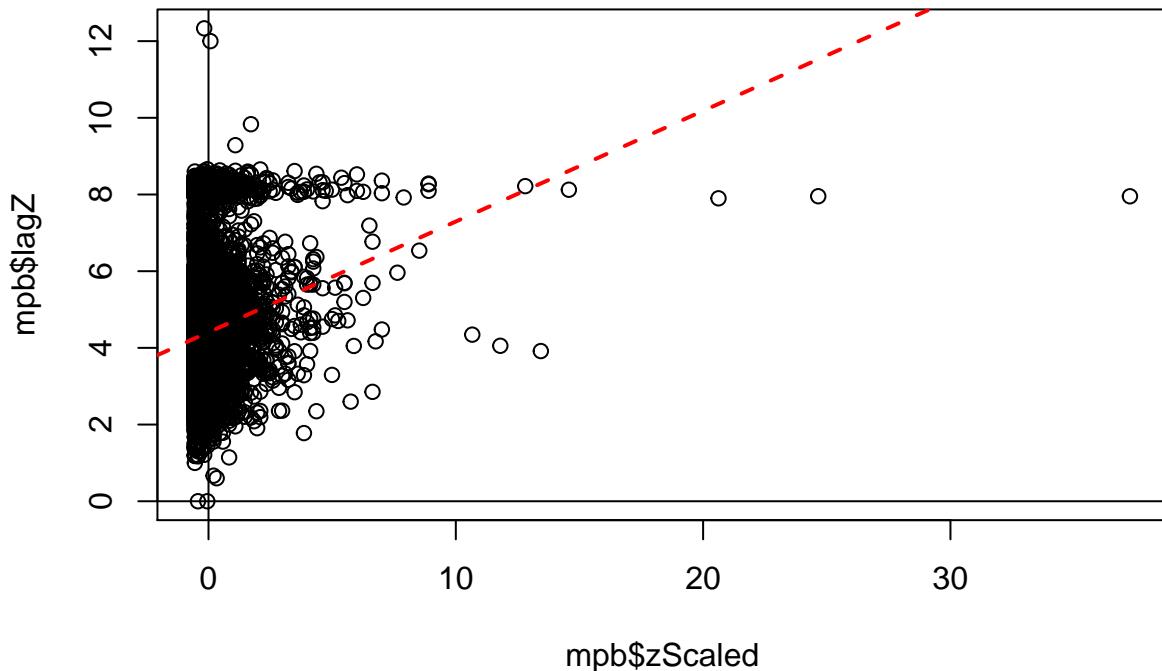
Si quieres, puedes explorar estas variables:

```
summary(mpb$zScaled)
summary(mpb$lagZ)
```

Y hacemos un diagrama de dispersión:

```
# Diagrama de dispersión
plot(mpb$zScaled, mpb$lagZ)
# Ejes que pasan por el origen
abline(h = 0, v = 0)
# Recta de ajuste lineal entre las dos variables
abline(lm(mpb$lagZ ~ mpb$zScaled), lty = 2, lwd = 2, col = "red")
title("Diagrama de dispersión de Moran")
```

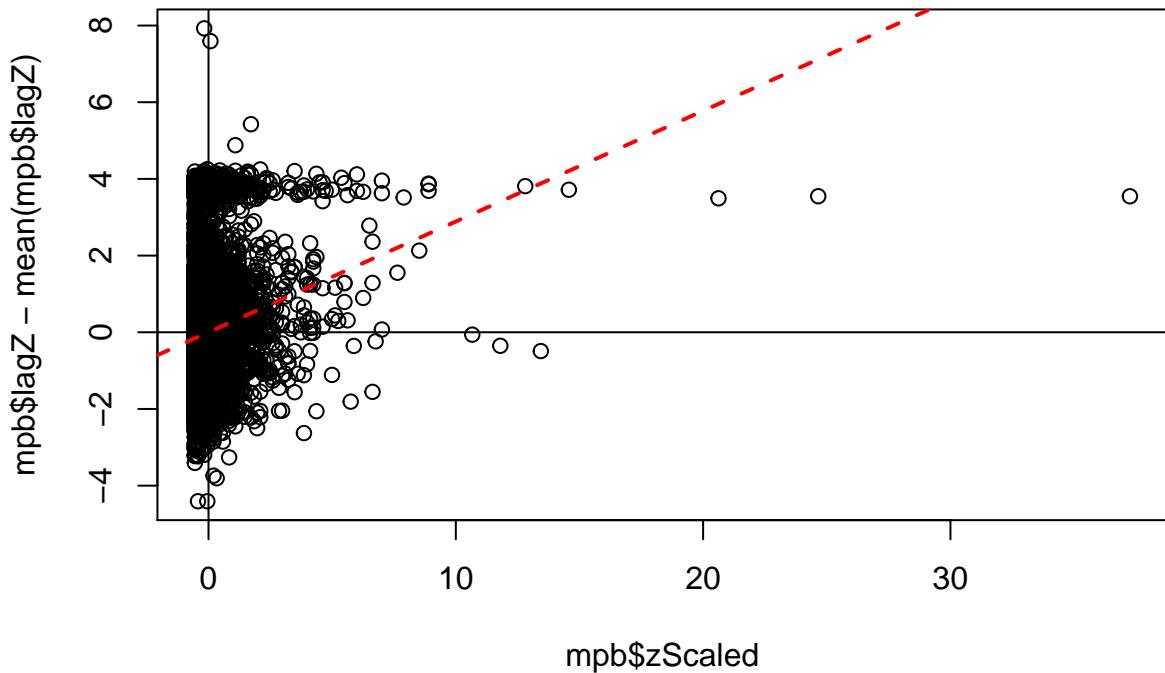
## Diagrama de dispersión de Moran



Algo raro pasa porque nuestra recta de ajuste no pasa por el origen... Hay que hacer un ligero cambio en las variables que queremos graficar. Con las variables estandarizadas, las unidades en la gráfica corresponderán a desviaciones estándar. La variable Z ya está normalizada, solo tenemos que modificar la variable de retraso espacial: a lagZ restarle su promedio.

```
# Con variables estandarizadas
plot(mpb$zScaled,mpb$lagZ-mean(mpb$lagZ))
# o bien, plot(mpb$zScaled,scale(mpb$lagZ))
abline(h = 0, v = 0)
abline(lm(mpb$lagZ-mean(mpb$lagZ) ~ mpb$zScaled), lty = 2, lwd = 2, col = "red")
title("Diagrama de dispersión de Moran")
```

## Diagrama de dispersión de Moran

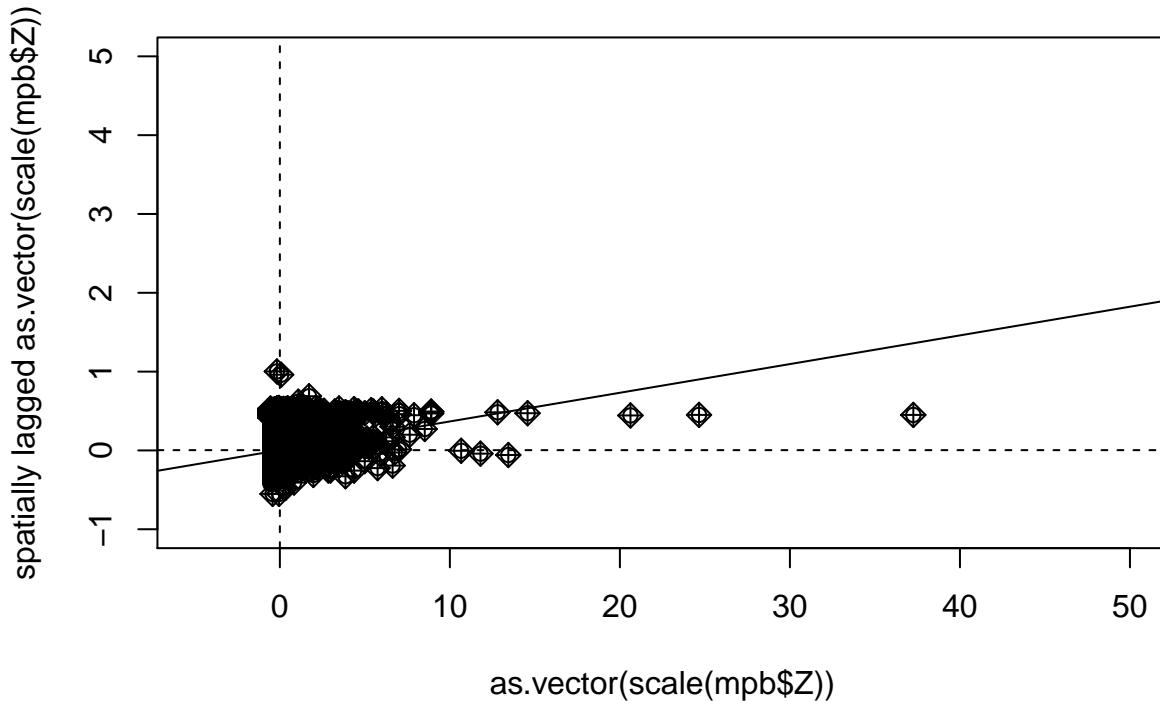


Ahora sí, los ejes están centrados en el origen y la recta de ajuste pasa por  $(0,0)$ . Hay una función más práctica que hace esto por nosotros:

```
# Moran plot

#moran.plot(mpb$Z, mpb.lw)
#moran.plot(mpb$Z, mpb.lw, xlim = c(-300, 300), ylim = c(-20, 20))
#moran.plot(as.vector(scale(mpb$Z)), mpb.lw)
#moran.plot(as.vector(scale(mpb$Z)), mpb.lw, xlim = c(-45, 45))
#moran.plot(as.vector(scale(mpb$Z)), mpb.lw, xlim = c(-45, 45), ylim = c(-5, 5))
#moran.plot(as.vector(scale(mpb$Z)), mpb.lw, xlim = c(-45, 45), ylim = c(-45, 45))
#moran.plot(as.vector(scale(mpb$Z)), mpb.lw, xlim = c(-300, 300), ylim = c(-45, 45))
moran.plot(as.vector(scale(mpb$Z)), mpb.lw, xlim = c(-5, 50), ylim = c(-1, 5), labels = F)
title("Diagrama de dispersión de Moran")
```

## Diagrama de dispersión de Moran

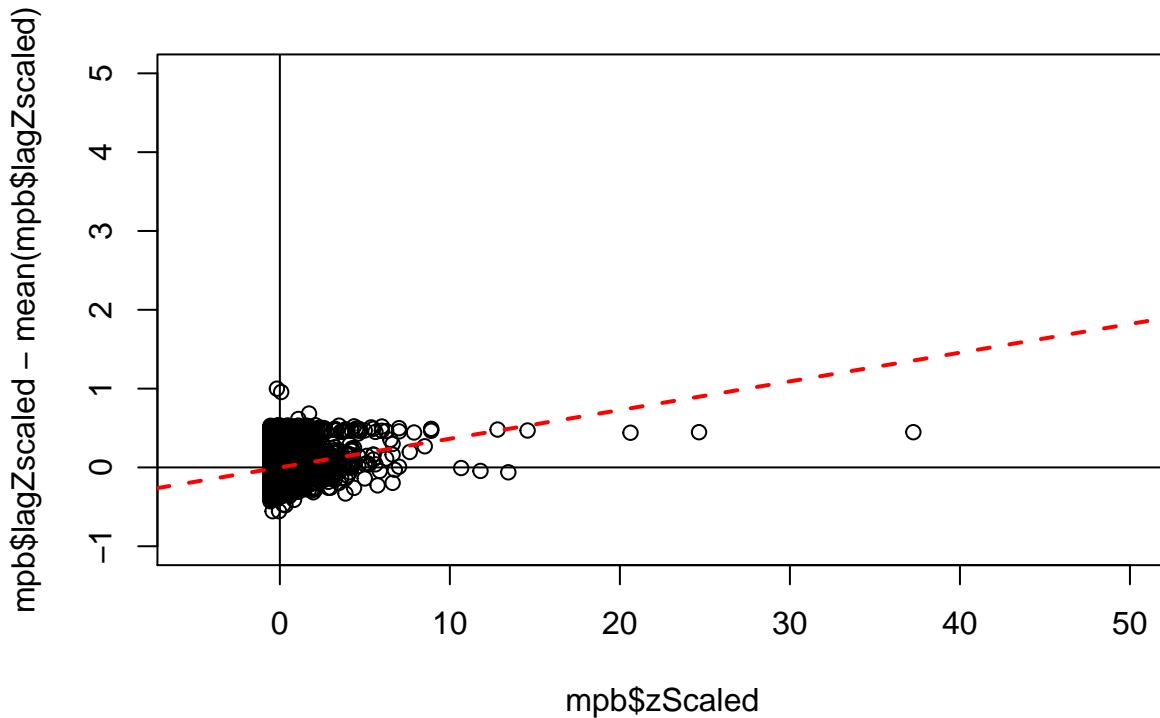


La pendiente del diagrama de dispersión parece ser positiva, lo que indicaría que hay un cierto nivel de autocorrelación espacial positiva a nivel **global**.

Una diferencia importante es que los límites de nuestra gráfica están muy diferentes de la que se obtiene con `moran.plot`. Se ven parecidas, pero no iguales. Para lograr esto, nos damos cuenta que la variable de retraso espacial en el eje horizontal también debe ser una variable estandarizada y normalizada.

```
# Variables de retraso espacial estandarizada
mpb$lagZscaled <- lag.listw(mpb.lw, scale(mpb$Z)) # moran.plot hace esto internamente
plot(mpb$zScaled,mpb$lagZscaled-mean(mpb$lagZscaled), xlim = c(-5,50), ylim = c(-1,5))
abline(h = 0, v = 0)
abline(lm(mpb$lagZscaled-mean(mpb$lagZscaled) ~ mpb$zScaled), lty = 2, lwd = 2, col = "red")
title("Diagrama de dispersión de Moran")
```

## Diagrama de dispersión de Moran



## Prueba estadística

Para una prueba estadística sobre la presunción de la autocorrelación en un patrón de puntos, podemos hacer lo siguiente:

```
# Prueba estadística
moran.test(mpb$Z, mpb.lw)

## Moran I test under randomisation

## data: mpb$Z
## weights: mpb.lw

## Moran I statistic standard deviate = 25.915, p-value < 2.2e-16
## alternative hypothesis: greater
## sample estimates:
## Moran I statistic      Expectation      Variance
##      3.639684e-02     -1.190476e-04    1.985404e-06
```

Como el valor de la estadística es mayor que el valor esperado y la varianza es muy pequeña, podemos pensar que, en efecto, hay autocorrelación espacial positiva, es decir, que el patrón de puntos que observamos tiende a *acumularse* en ciertas zonas y que no estamos ante un proceso espacial completamente aleatorio. Pero, ¿cómo saber si el valor de la  $I$  de Moran es *suficientemente* grande, o no, para que sugiera que un modelo de un proceso con autocorrelación sea una mejor alternativa a uno en el que los valores que tenemos sean observaciones independientes?

Hay dos partes que tenemos que analizar: i) el valor de  $I$  en una escala absoluta y ii) qué tan probable es que el valor observado de  $I$  se obtenga si las observaciones fueran independientes.

Para i), podemos usar la siguiente función y obtener el rango de valores de  $I$ . **NOTA:** esta ejecución puede tardar **mucho** tiempo.

```
# Rango de valores de I
moran.range <- function(lw){
  wmat <- listw2mat(lw)
  return(range(eigen((wmat+t(wmat))/2)$values)))
}
moran.range(mpb.lw)

## [1] -1.000000  1.037054
```

Como el valor que se obtiene está entre  $-1.000$  y  $1.037$ , podemos pensar que, en efecto, obtuvimos un valor que indica autocorrelación espacial global positiva, pues es de  $0.036$ , mayor que la media ( $0.018$ ).

Para ii), podemos hacer una prueba de inferencia estadística clásica, usar la hipótesis nula de que no haya autocorrelación espacial y calcular la probabilidad de obtener un valor de la  $I$  de Moran igual o mayor a la observada. Sin entrar en mayores detalles, bajo ciertos supuestos se puede mostrar que  $I$  se puede aproximar por una distribución normal. Podemos hacer una prueba de inferencia estadística con una distribución normal al usar el argumento `randomisation = FALSE` como sigue:

```
# Inferencia bajo normalidad
moran.test(mpb$Z, mpb.lw, randomisation = FALSE)

## Moran I test under normality

## data: mpb$Z
## weights: mpb.lw

## Moran I statistic standard deviate = 25.419, p-value < 2.2e-16
## alternative hypothesis: greater
## sample estimates:
## Moran I statistic      Expectation      Variance
## 3.639684e-02 -1.190476e-04  2.063765e-06
```

Volvemos a obtener valores parecidos y tenemos evidencia para rechazar la hipótesis nula de que los valores de  $Z$  están distribuidos de manera normal.

Si seguimos sin estar convencidos, podemos emplear una estrategia más: *simulaciones Monte Carlo*. En este enfoque se hace un cierto número de permutaciones sobre los datos, se les asignan a los puntos y se calcula la  $I$  de Moran para cada permutación. La  $I$  *verdadera* se calcula a partir de los datos y ubicaciones originales. Si la hipótesis nula es cierta, la probabilidad de obtener el patrón observado es igual a cualquier otra permutación de los valores de  $Z$  en los demás puntos. Si  $m$  es el número de simulaciones que obtienen una  $I$  de Moran mayor a la observada y  $M$  es el número total de simulaciones, la probabilidad de obtener la  $I$  de Moran observada o un valor mayor es:

$$p = \frac{m + 1}{M + 1}.$$

Podemos hacer este cálculo como sigue. El tercer argumento es el número de simulaciones. **Atención:** dependiendo el valor del tercer argumento, puede tardar **mucho** en terminar de correr las permutaciones...

```
# Simulaciones MC
moran.mc(mpb$Z, mpb.lw, 10000)
```

```
## Monte-Carlo simulation of Moran I
```

```

## data: mpb$Z
## weights: mpb.lw
## number of simulations + 1: 10001

## statistic = 0.036397, observed rank = 10001, p-value = 9.999e-05
## alternative hypothesis: greater

```

Observa que volvemos a obtener evidencia en favor de rechazar la hipótesis nula.

## Estadística local

Recuerda que al inicio definimos la versión local de la  $I$  de Moran,  $I_i$ . La podemos calcular como sigue:

```

# Calcular la I de Moran local
lmoran <- localmoran(mpb$Z, mpb.lw)

```

Si quieres explorar el resultado, puedes usar:

```
summary(lmoran)
```

## Mapa de cúmulos

A partir de la gráfica de dispersión de Moran, vamos a codificar los valores de cada cuadrante en un *tipo de cúmulo*: aquellos de valores altos rodeados de valores altos, los de valores bajos rodeados de bajos, los de valores bajos rodeados de altos y los de valores altos rodeados de bajos.

Primero definimos una variable en la cual vamos a almacenar esta definición de los cuadrantes. Haremos que este nuevo vector sea de tipo numérico y su tamaño sea igual al número de elementos que hay en el cálculo de la  $I$  de Moran local (igual al número de puntos en el patrón).

```

# Definir vector de cuadrante
cuadrante <- vector(mode = "numeric", length = nrow(lmoran))

```

También vamos a definir un nivel de significancia estadística con la cual vamos a trabajar. Empezamos con un valor *relajado* en vez de uno *estricto*.

```

# Definir significancia
significancia <- 0.05

```

Como lo hicimos anteriormente, definimos dos variables estandarizadas:

```

# Centrar la variable de interés alrededor de su media
centerZ <- scale(mpb$Z) #Es lo mismo que (mpb$Z-mean(mpb$Z))/sd(mpb$Z)
# Centrar el valor de la I de Moran local alrededor de su media
centerLag <- mpb$lagZscaled

```

Ahora catalogamos los valores de acuerdo al cuadrante en el que se encuentren en la gráfica de Moran:

```

# Definición de cúmulos
cuadrante[centerZ > 0 & centerLag > 0] <- 1 # High-High
cuadrante[centerZ < 0 & centerLag < 0] <- 2 # Low-Low
cuadrante[centerZ < 0 & centerLag > 0] <- 3 # Low-High
cuadrante[centerZ > 0 & centerLag < 0] <- 4 # High-Low
cuadrante[lmoran[,5] > significancia] <- 0 # Not significant

```

Definimos los colores que vamos a usar para el mapa de cúmulos:

```

# Colores para los cúmulos
cColors <- c(rgb(0.74, 0.74, 0.74), rgb(1, 0, 0), rgb(0, 0, 1), rgb(0.58, 0.58,
1), rgb(1, 0.58, 0.58))
# gris, rojo, azul, azul claro y rojo claro not significant, high-high,
# low-low, low-high, high-low

```

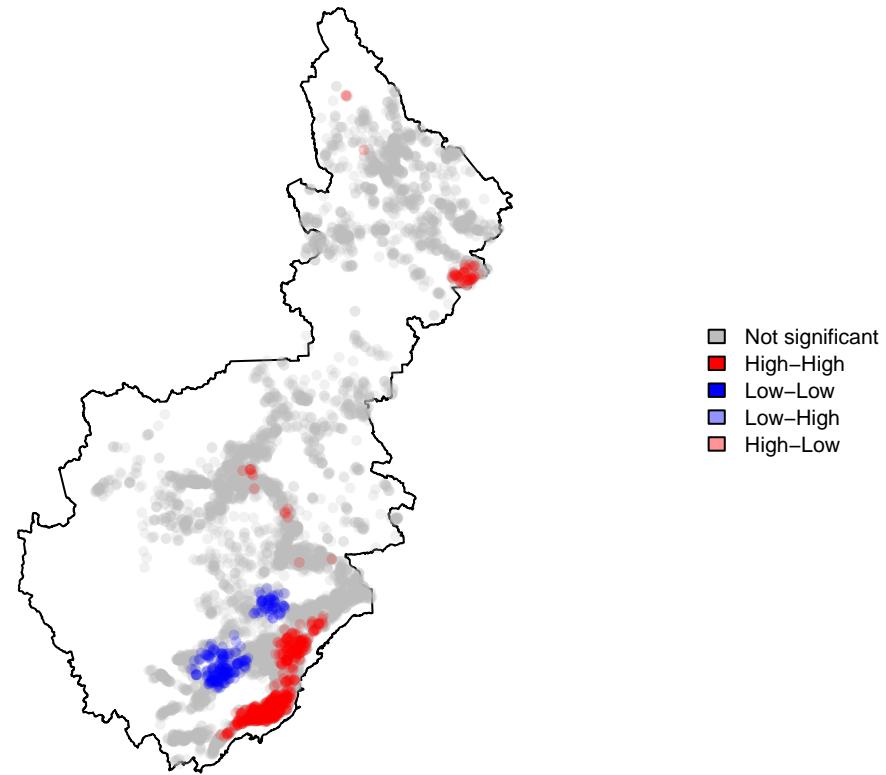
Hacemos el primer mapa:

```

# Definir márgenes para ocupar todo el espacio
par(mar = c(0, 0, 1, 0))
# Primer mapa
plot(frontera)
# Plot not significant
plot(mpb[cuadrante == 0, ], col = rgb(0.74, 0.74, 0.74, alpha = 0.2), add = T,
      pch = 16, cex = 0.75)
# Plot high-highs
plot(mpb[cuadrante == 1, ], col = rgb(1, 0, 0, alpha = 0.2), add = T, pch = 16,
      cex = 0.75)
# Plot low-lows
plot(mpb[cuadrante == 2, ], col = rgb(0, 0, 1, alpha = 0.2), add = T, pch = 16,
      cex = 0.75)
# Plot low-highs
plot(mpb[cuadrante == 3, ], col = rgb(0.58, 0.58, 1, alpha = 0.75), add = T,
      pch = 16, cex = 0.75)
# Plot high-lows
plot(mpb[cuadrante == 4, ], col = rgb(1, 0.58, 0.58, alpha = 0.75), add = T,
      pch = 16, cex = 0.75)
legend("right", legend = c("Not significant", "High-High", "Low-Low", "Low-High",
"High-Low"), fill = cColors, bty = "n", cex = 0.7, y.intersp = 1, x.intersp = 1)
title(paste("LISA Cluster Map, p = ", significancia))

```

## LISA Cluster Map, p = 0.05



### Mapa de significancia

Ahora vemos como están distribuidas las distintas significancias en nuestros datos. Primero definimos un vector para guardar los valores:

```
# Definir vector de significancias  
pValues <- vector(mode = "numeric", length = nrow(lmoran))
```

Y obtenemos cada nivel de significancia estadística a partir de los datos:

```
# Definir niveles de significancia  
pValues[(lmoran[, 5] > 0.05)] <- 0 #Not significant  
pValues[(lmoran[, 5] <= 0.05)] <- 4  
pValues[(lmoran[, 5] <= 0.01)] <- 3  
pValues[(lmoran[, 5] <= 0.001)] <- 2  
pValues[(lmoran[, 5] <= 0.0001)] <- 1
```

Definimos los colores para este mapa:

```
# Colores para las significancias  
colorsPValue <- c(rgb(0.74, 0.74, 0.74), rgb(0.22, 0.98, 0.3), rgb(0, 0.75,  
0), rgb(0, 0.44, 0), rgb(0, 0, 0))  
# gris, verde claro, verde medio, verde oscuro, negro
```

Hacemos el segundo mapa:

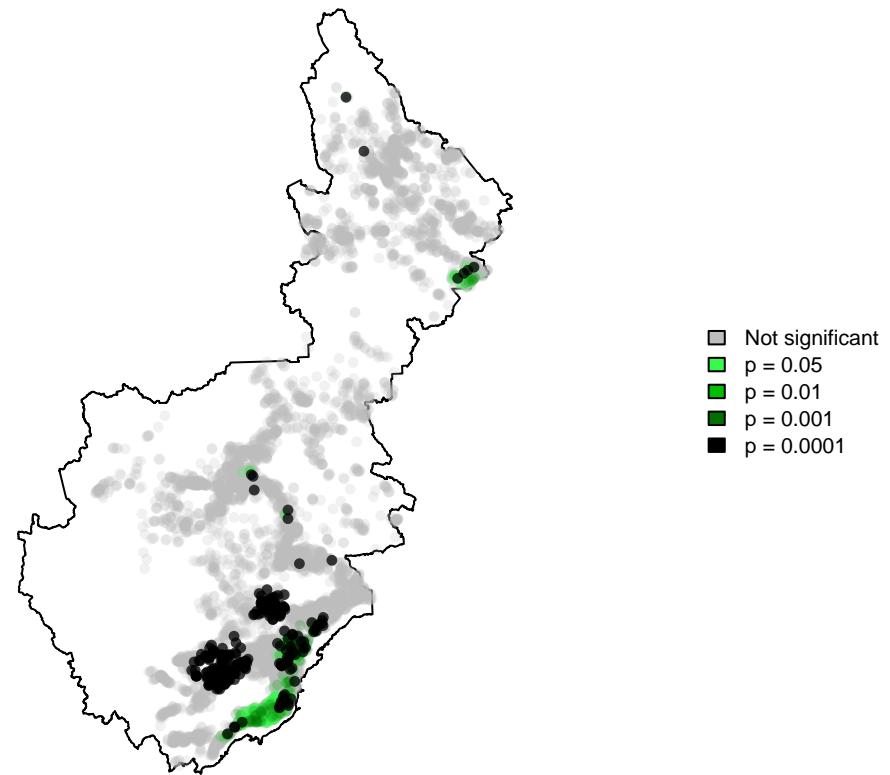
```
# Segundo mapa Definir márgenes para ocupar todo el espacio  
par(mar = c(0, 0, 1, 0))
```

```

plot(frontera)
# Plot not significant
plot(mpb[pValues == 0, ], col = rgb(0.74, 0.74, 0.74, alpha = 0.2), add = T,
      pch = 16, cex = 0.75)
# Plot 0.05
plot(mpb[pValues == 1, ], col = rgb(0.22, 0.98, 0.3, alpha = 0.2), add = T,
      pch = 16, cex = 0.75)
# Plot 0.01
plot(mpb[pValues == 2, ], col = rgb(0, 0.75, 0, alpha = 0.2), add = T, pch = 16,
      cex = 0.75)
# Plot 0.001
plot(mpb[pValues == 3, ], col = rgb(0, 0.44, 0, alpha = 0.2), add = T, pch = 16,
      cex = 0.75)
# Plot 0.0001
plot(mpb[pValues == 4, ], col = rgb(0, 0, 0, alpha = 0.75), add = T, pch = 16,
      cex = 0.75)
legend("right", legend = c("Not significant", "p = 0.05", "p = 0.01", "p = 0.001",
                           "p = 0.0001"), fill = colorsPValue, bty = "n", cex = 0.7, y.intersp = 1,
                           x.intersp = 1)
title("Local p-value")

```

**Local p-value**



Los dos mapas juntos:

```

# Definir que tendremos un gráfica de 1 renglón y dos columnas
par(mfrow = c(1, 2))

```

```

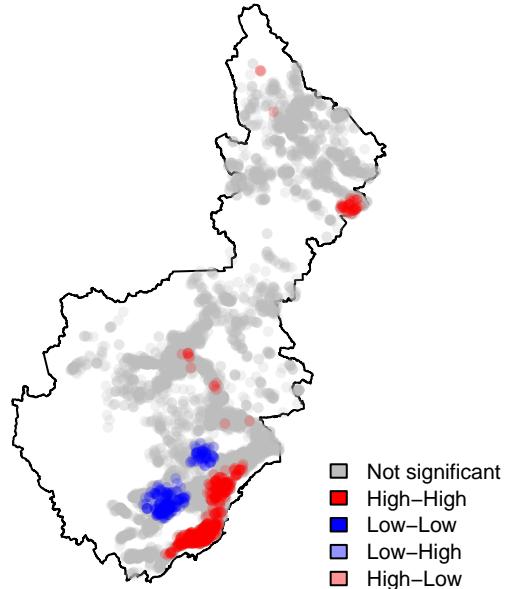
# Sin margen arriba, un poco de margen entre cada plot y definir tipo de
# región cuadrada
op <- par(oma = c(0, 0, 0, 0), mar = c(0, 0, 2, 0), pty = "s")

# Primer mapa
plot(frontera)
# Plot not significant
plot(mpb[cuadrante == 0, ], col = rgb(0.74, 0.74, 0.74, alpha = 0.2), add = T,
      pch = 16, cex = 0.75)
# Plot high-highs
plot(mpb[cuadrante == 1, ], col = rgb(1, 0, 0, alpha = 0.2), add = T, pch = 16,
      cex = 0.75)
# Plot low-lows
plot(mpb[cuadrante == 2, ], col = rgb(0, 0, 1, alpha = 0.2), add = T, pch = 16,
      cex = 0.75)
# Plot low-highs
plot(mpb[cuadrante == 3, ], col = rgb(0.58, 0.58, 1, alpha = 0.25), add = T,
      pch = 16, cex = 0.75)
# Plot high-lows
plot(mpb[cuadrante == 4, ], col = rgb(1, 0.58, 0.58, alpha = 0.75), add = T,
      pch = 16, cex = 0.75)
legend("bottomright", legend = c("Not significant", "High-High", "Low-Low",
                                 "Low-High", "High-Low"), fill = cColors, bty = "n", cex = 0.7, y.intersp = 1,
       x.intersp = 1)
title(paste("LISA Cluster Map, p = ", significancia))

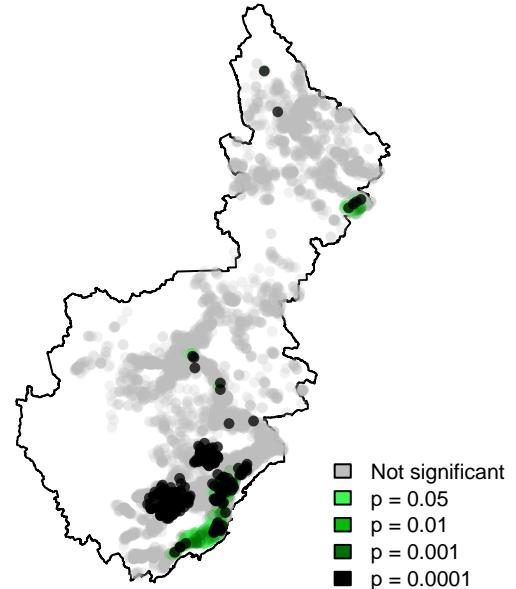
# Segundo mapa
plot(frontera)
# Plot not significant
plot(mpb[pValues == 0, ], col = rgb(0.74, 0.74, 0.74, alpha = 0.2), add = T,
      pch = 16, cex = 0.75)
# Plot 0.05
plot(mpb[pValues == 1, ], col = rgb(0.22, 0.98, 0.3, alpha = 0.2), add = T,
      pch = 16, cex = 0.75)
# Plot 0.01
plot(mpb[pValues == 2, ], col = rgb(0, 0.75, 0, alpha = 0.2), add = T, pch = 16,
      cex = 0.75)
# Plot 0.001
plot(mpb[pValues == 3, ], col = rgb(0, 0.44, 0, alpha = 0.2), add = T, pch = 16,
      cex = 0.75)
# Plot 0.0001
plot(mpb[pValues == 4, ], col = rgb(0, 0, 0, alpha = 0.75), add = T, pch = 16,
      cex = 0.75)
legend("bottomright", legend = c("Not significant", "p = 0.05", "p = 0.01",
                                 "p = 0.001", "p = 0.0001"), fill = colorsPValue, bty = "n", cex = 0.7, y.intersp = 1,
       x.intersp = 1)
title("Local p-value")

```

LISA Cluster Map,  $p = 0.05$



Local p-value



```
# Regresar a una sola gráfica en toda la página  
par(mfrow = c(1,1))
```