## CONTENTS

1 Introduction  

2 Scale and distance  
   2.1 Introduction  
   2.2 Scale and resolution  
   2.3 Zonation  
   2.4 Distance  
      2.4.1 Distance matrix  
      2.4.2 Distance for longitude/latitude coordinates  
   2.5 Spatial influence  
      2.5.1 Adjacency  
      2.5.2 Two nearest neighbours  
      2.5.3 Weights matrix  
      2.5.4 Spatial influence for polygons  
   2.6 Raster based distance metrics  
      2.6.1 distance  
      2.6.2 cost distance  
      2.6.3 resistance distance  

3 Spatial autocorrelation  
   3.1 Introduction  
      3.1.1 Temporal autocorrelation  
      3.1.2 Spatial autocorrelation  
   3.2 Example data  
   3.3 Adjacent polygons  
   3.4 Compute Moran’s I  

4 Interpolation  
   4.1 Introduction  
   4.2 Temperature in California  
      4.2.1 9.2 NULL model  
      4.2.2 proximity polygons  
      4.2.3 Nearest neighbour interpolation  
      4.2.4 Inverse distance weighted  
   4.3 California Air Pollution data  
      4.3.1 Data preparation  
      4.3.2 Fit a variogram  
      4.3.3 Ordinary kriging  
      4.3.4 Compare with other methods  
      4.3.5 Cross-validate  

1  

3  

3  

4  

8  

10  

10  

11  

12  

13  

18  

18  

18  

19  

19  

22  

23  

24  

27  

33  

33  

36  

36  

40  

42  

44  

44  

45  

48  

50  

53
## 5 Spatial distribution models

5.1 Data
  5.1.1 Observations
  5.1.2 Predictors
  5.1.3 Background data
  5.1.4 Combine presence and background

5.2 Fit a model
  5.2.1 CART
  5.2.2 Random Forest

5.3 Predict
  5.3.1 Regression
  5.3.2 Classification

5.4 Extrapolation

5.5 Further reading

## 6 Local regression

6.1 California precipitation
6.2 California House Price Data
6.3 Summarize
6.4 Regression
6.5 Geographically Weighted Regression
  6.5.1 By county
  6.5.2 By grid cell

6.6 spgwr package

## 7 Spatial regression models

7.1 Introduction
7.2 Reading & aggregating data
  7.2.1 Get the data
7.3 Basic OLS model
7.4 Spatial lag model
7.5 Spatial error model
7.6 Questions

## 8 Point pattern analysis

8.1 Introduction
8.2 Basic statistics
8.3 Density
8.4 Distance based measures
8.5 Spatstat package
In this section we introduce a number of approaches and techniques that are commonly used in spatial data analysis and modelling.

Spatial data are mostly like other data. The same general principles apply. But there are few things that are rather important to consider when using spatial data that are not common with other data types. These are discussed in Chapters 2 and 3 and include issues of scale and zonation (the modifiable areal unit problem), distance and spatial autocorrelation.

The other chapters, introduce methods in different areas of spatial data analysis. These include the three classical area of spatial statistics (point pattern analysis, regression and inference with spatial data, geostatistics (interpolation using Kriging), as well some other methods (local and global regression and classification with spatial data).

Some of the material presented here is based on examples in the book “Geographic Information Analysis” by David O’Sullivan and David J. Unwin. This book provides an excellent and very accessible introduction to spatial data analysis. It has much more depth than what we present here. But the book does not show how to practically implement the approaches that are discussed — which is the main purpose of this website.

The spatial statistical methods are treated in much more detail in “Applied Spatial Data Analysis with R” by Bivand, Pebesma and Gómez-Rubio.

This section builds on our Introduction to Spatial Data Manipulation R, that you should read first.
CHAPTER TWO

SCALE AND DISTANCE

2.1 Introduction

Scale, aggregations, and distance are two key concepts in spatial data analysis that can be tricky to come to grips with. This chapter first discusses scale and related concepts resolution, aggregation and zonation. The second part of the chapter discusses distance and adjacency.

2.2 Scale and resolution

The term “scale” is tricky. In its narrow geographic sense, it is the the ratio of a distance on a (paper) map to the actual distance. So if a distance of 1 cm on map “A” represents 100 m in the real world, the map scale is 1/10,000 (1:10,000 or 10-4). If 1 cm on map “B” represents 10 km in the real world, the scale of that map is 1/1,000,000. The first map “A” would have relatively large scale (and high resolution) as compared to the second map “B”, that would have a small scale (and low resolution). It follows that if the size maps “A” and “B” were the same, map “B” would represent a much larger area (would have a much larger “spatial extent”). For that reason, most people would refer to map “B” having a “larger scale”. That is technically wrong, but there is not much point in fighting that, and it is simply best to avoid the term “scale”, and certainly “small scale” and “large scale”, because that technically means the opposite of what most people think. If you want to use these terms, you should probably use them how they are commonly understood; unless you are among cartographers, of course.

Now that mapping has become a computer based activity, scale is even more treacherous. You can use the same data to make maps of different sizes. These would all have a different scale. With digital data, we are more interested in the “inherent” or “measurement” scale of the data. This is sometimes referred to as “grain” but I use “(spatial) resolution”. In the case of raster data the notion of resolution is straightforward: it is the size of the cells. For vector data resolution is not as well defined, and it can vary largely within a data set, but you can think of it as the average distance between the nodes (coordinate pairs) of the lines or polygons. Point data do not have a resolution, unless cases that are within a certain distance of each other are merged into a single point (the actual geographic objects represented by points, actually do cover some area; so the actual average size of those areas could also be a measure of interest, but it typically is not).

In the digital world it is easy to create a “false resolution”, either by dividing raster cells into 4 or more smaller cells, or by adding nodes in-between nodes of polygons. Imagine having polygons with soils data for a country. Let’s say that these polygons cover, on average, an area of 100 * 100 = 10,000 km². You can transfer the soil properties associated with each polygon, e.g. pH, to a raster with 1 km² spatial resolution; and now might (incorrectly) say that you have a 1 km² spatial resolution soils map. So we need to distinguish the resolution of the representation (data) and the resolution of the measurements or estimates. The lowest of the two is the one that matters.

Why does scale/resolution matter?

First of all, different processes have different spatial and temporal scales at which they operate Levin, 1992 — in this context, scale refers both to “extent” and “resolution”. Processes that operate over a larger extent (e.g., a forest) can be
studied at a larger resolution (trees) whereas processes that operate over a smaller extent (e.g., a tree) may need to be studied at the level of leaves.

From a practical perspective: it affects our estimates of length and size. For example, if you wanted to know the length of the coastline of Britain, you could use the length of a spatial dataset representing that coastline. You could get rather different numbers depending on the dataset used. The higher the resolution of the spatial data, the longer the coastline would appear to be. This is not just a problem of the representation (the data), also at a theoretical level, one can argue that the length of the coastline is not defined, as it becomes infinite if your resolution approaches zero. This is illustrated here.

Resolution also affects our understanding of relationships between variables of interest. In terms of data collection, this means that we want data to be at the highest spatial (and temporal) resolution possible (affordable). We can aggregate our data to lower resolutions, but it is not nearly as easy, or even impossible to correctly disaggregate (“downscale”) data to a higher resolution.

### 2.3 Zonation

Geographic data are often aggregated by zones. While we would like to have data at the most granular level that is possible or meaningful (individuals, households, plots, sites), reality is that we often can only get data that is aggregated. Rather than having data for individuals, we may have mean values for all inhabitants of a census district. Data on population, disease, income, or crop yield, is typically available for entire countries, for a number of sub-national units (e.g., provinces), or a set of raster cells.

The areas used to aggregate data are arbitrary (at least relative to the data of interest). The way the borders of the areas are drawn (how large, what shape, where) can strongly affect the patterns we see and the outcome of data analysis. This is sometimes referred to as the “Modifiable Areal Unit Problem” (MAUP). The problem of analyzing aggregated data is referred to as “Ecological Inference”.

To illustrate the effect of zonation and aggregation, I create a region with 1000 households. For each household we know where they live and what their annual income is. I then aggregate the data to a set of zones.

The income distribution data

```r
set.seed(0)
xy <- cbind(x=runif(1000, 0, 100), y=runif(1000, 0, 100))
income <- (runif(1000) * abs((xy[,1] - 50) * (xy[,2] - 50))) / 500
```

Inspect the data, both spatially and non-spatially. The first two plots show that there are many poor people and a few rich people. The third that there is a clear spatial pattern in where the rich and the poor live.

```r
par(mfrow=c(1,3), las=1)
plot(sort(income), col=rev(terrain.colors(1000)), pch=20, cex=.75, ylab='income')
hist(income, main='', col=rev(terrain.colors(1000)), pch=20, cex=.75, xlab='income')
plot(xy, xlim=c(0,100), ylim=c(0,100), cex=income, col=rev(terrain.colors(1000)))[10*(income+1)]
```
Income inequality is often expressed with the Gini coefficient.

\[
n \leftarrow \text{length}(\text{income}) \\
G \leftarrow \frac{2 \sum (\text{sort}(\text{income}) \times 1:n)}{\sum \text{income}} - (n + 1) / n \\
\]

G

## [1] 0.5814548

For our data set the Gini coefficient is 0.581.

Now assume that the household data was grouped by some kind of census districts. I create different districts, in our case rectangular raster cells, and compute mean income for each district.

```r
library(raster)
## Loading required package: sp
rl <- raster(ncol=1, nrow=4, xmin=0, xmax=100, ymin=0, ymax=100, crs=NA)
rl <- rasterize(xy, rl, income, mean)
```

(continues on next page)
Spatial Data Analysis with R

r2 <- raster(ncol=4, nrow=1, xmn=0, xmx=100, ymn=0, ymx=100, crs=NA)
r2 <- rasterize(xy, r2, income, mean)

r3 <- raster(ncol=2, nrow=2, xmn=0, xmx=100, ymn=0, ymx=100, crs=NA)
r3 <- rasterize(xy, r3, income, mean)

r4 <- raster(ncol=3, nrow=3, xmn=0, xmx=100, ymn=0, ymx=100, crs=NA)
r4 <- rasterize(xy, r4, income, mean)

r5 <- raster(ncol=5, nrow=5, xmn=0, xmx=100, ymn=0, ymx=100, crs=NA)
r5 <- rasterize(xy, r5, income, mean)

r6 <- raster(ncol=10, nrow=10, xmn=0, xmx=100, ymn=0, ymx=100, crs=NA)
r6 <- rasterize(xy, r6, income, mean)

Have a look at the plots of the income distribution and the sub-regional averages.

par(mfrow=c(2,3), las=1)
plot(r1); plot(r2); plot(r3); plot(r4); plot(r5); plot(r6)

It is not surprising to see that the smaller the regions get, the better the real pattern is captured. But in all cases, the histograms show that we do not capture the full income distribution (compare to the histogram with the data for

Chapter 2. Scale and distance
individuals).

```r
define `par(mfrow=c(1,3), las=1)`
hist(r4, main='', col=rev(terrain.colors(10)), xlim=c(0,5), breaks=seq(0, 5, 0.5))
hist(r5, main='', col=rev(terrain.colors(10)), xlim=c(0,5), breaks=seq(0, 5, 0.5))
hist(r6, main='', col=rev(terrain.colors(10)), xlim=c(0,5), breaks=seq(0, 5, 0.5))
```

2.3. Zonation
2.4 Distance

Distance is a numerical description of how far apart things are. It is the most fundamental concept in geography. After all, Waldo Tobler’s First Law of Geography states that “everything is related to everything else, but near things are more related than distant things”. But how far away are things? That is not always as easy a question as it seems. Of course we can compute distance “as the crow flies” but that is often not relevant. Perhaps you need to also consider national borders, mountains, or other barriers. The distance between A and B may even by asymmetric, meaning that it the distance from A to B is not the same as from B to A (for example, the President of the United States can call me, but I cannot call him (or her)); or because you go faster when walking downhill than when waling uphill.

2.4.1 Distance matrix

Distances are often described in a “distance matrix”. In a distance matrix we have a number for the distance between all objects of interest. If the distance is symmetric, we only need to fill half the matrix.

Let’s create a distance matrix from a set of points. We start with a set of points:

```r
A <- c(40, 43)
B <- c(101, 1)
C <- c(111, 54)
D <- c(104, 65)
E <- c(60, 22)
F <- c(20, 2)
pts <- rbind(A, B, C, D, E, F)
```

```
##   [,1] [,2]
## A   40  43
## B 101   1
## C 111  54
## D 104  65
## E  60  22
## F  20   2
```

Plot the points and labels:

```r
plot(pts, xlim=c(0,120), ylim=c(0,120), pch=20, cex=2, col='red', xlab='X', ylab='Y', las=1)
text(pts+5, LETTERS[1:6])
```
You can use the `dist` function to make a distance matrix with a data set of any dimension.

```r
dis <- dist(pts)
print(dis)
```

```
## A B C D E
## B 74.06079
## C 71.84706 53.93515
## D 67.67570 64.07027 13.03840
## E 29.00000 46.06517 60.20797 61.52235
## F 45.61798 81.00617 104.80935 105.00000 44.72136
```

We can check that for the first point using Pythagoras' theorem.

```r
sqrt((40 - 101)^2 + (43 - 1)^2)
```

```
## [1] 74.06079
```

We can transform a distance matrix into a normal matrix.

2.4. Distance
Distance matrices are used in all kinds of non-geographical applications. For example, they are often used to create cluster diagrams (dendograms).

**Question 4:** Show R code to make a cluster dendogram summarizing the distances between these six sites, and plot it. See `?hclust`.

### 2.4.2 Distance for longitude/latitude coordinates

Now consider that the values in `pts` were coordinates in degrees (longitude/latitude). Then the cartesian distance as computed by the `dist` function would be incorrect. In that case we can use the `pointDistance` function from the `raster` package.

```r
library(raster)
gdis <- pointDistance(pts, lonlat=TRUE)
gdis
```

**Question 5:** What is the unit of the values in `gdis`?

### 2.5 Spatial influence

An important step in spatial statistics and modelling is to get a measure of the spatial influence between geographic objects. This can be expressed as a function of adjacency or (inverse) distance, and is often expressed as a spatial weights matrix. Influence is of course very complex and cannot really be measured and it can be estimated in many ways. For example the influence between a set of polygons (countries) can be expressed as having a shared border or not (being adjacent); as the “crow-fly” distance between their centroids; or as the lengths of a shared border, and in other ways.
2.5.1 Adjacency

Adjacency is an important concept in some spatial analysis. In some cases objects are considered adjacent when they “touch”, e.g. neighboring countries. It can also be based on distance. This is the most common approach when analyzing point data.

We create an adjacency matrix for the point data analysed above. We define points as “adjacent” if they are within a distance of 50 from each other. Given that we have the distance matrix $D$, this is easy to do.

```r
a <- D < 50
a
## A B C D E F
## A TRUE FALSE FALSE FALSE TRUE TRUE
## B FALSE TRUE FALSE FALSE TRUE FALSE
## C FALSE FALSE TRUE TRUE FALSE FALSE
## D FALSE FALSE TRUE TRUE FALSE FALSE
## E TRUE TRUE FALSE FALSE TRUE TRUE
## F TRUE FALSE FALSE FALSE TRUE TRUE
```

In adjacency matrices the diagonal values are often set to \texttt{NA} (we do not consider a point to be adjacent to itself). And \texttt{TRUE/FALSE} values are commonly stored as \texttt{1/0} (this is equivalent, and we can make this change with a simple trick: multiplication with 1).

```r
diag(a) <- NA
Adj50 <- a * 1
Adj50
## A B C D E F
## A NA 0 0 0 1 1
## B 0 NA 0 0 1 0
## C 0 0 NA 1 0 0
## D 0 0 1 NA 0 0
## E 1 1 0 0 NA 1
## F 1 0 0 0 1 NA
```

2.5.2 Two nearest neighbours

What if you wanted to compute the “two nearest neighbours” (or three, or four) adjacency-matrix? Here is how you can do that. For each row, we first get the column numbers in order of the values in that row (that is, the numbers indicate how the values are ordered).

```r
cols <- apply(D, 1, order)
# we need to transpose the result
cols <- t(cols)
```

And then get columns 2 to 3 (why not column 1?)

```r
cols <- cols[, 2:3]
cols
## [,1] [,2]
## A 5 6
## B 5 3
## C 4 2
## D 3 5
```

(continues on next page)
As we now have the column numbers, we can make the row-column pairs that we want (rowcols).

```
rowcols <- cbind(rep(1:6, each=2), as.vector(t(cols))
head(rowcols)
## [,1] [,2]
## [1,] 1 5
## [2,] 1 6
## [3,] 2 5
## [4,] 2 3
## [5,] 3 4
## [6,] 3 2
```

We use these pairs as indices to change the values in matrix \( A_k3 \).

```
Ak3 <- Adj50 * 0
Ak3[rowcols] <- 1
Ak3
## A B C D E F
## A NA 0 0 0 1 1
## B 0 NA 1 0 1 0
## C 0 1 NA 1 0 0
## D 0 0 1 NA 1 0
## E 1 0 0 0 NA 1
## F 1 0 0 0 1 NA
```

### 2.5.3 Weights matrix

Rather than expressing spatial influence as a binary value (adjacent or not), it is often expressed as a continuous value. The simplest approach is to use inverse distance (the further away, the lower the value).

```
W <- 1 / D
round(W, 4)
## A B C D E F
## A Inf 0.0135 0.0139 0.0148 0.0345 0.0219
## B 0.0135 Inf 0.0185 0.0156 0.0217 0.0123
## C 0.0139 0.0185 Inf 0.0767 0.0166 0.0095
## D 0.0148 0.0156 0.0767 Inf 0.0163 0.0095
## E 0.0345 0.0217 0.0166 0.0163 Inf 0.0224
## F 0.0219 0.0123 0.0095 0.0095 0.0224 Inf
```

Such as “spatial weights” matrix is often “row-normalized”, such that the sum of weights for each row in the matrix is the same. First we get rid if the Inf values by changing them to NA. (Where did the Inf values come from?)

```
W[!is.finite(W)] <- NA
```

Then compute the row sums.

```
rtot <- rowSums(W, na.rm=TRUE)
# this is equivalent to
```

(continues on next page)
And divide the rows by their totals and check if they row sums add up to 1.

\[ W \leftarrow W / \text{rtot} \]
\[ \text{rowSums}(W, \text{na.rm=TRUE}) \]
\[ \# A B C D E F \]  
\[ \# 1 1 1 1 1 1 \]

The values in the columns do not add up to 1.

\[ \text{colSums}(W, \text{na.rm=TRUE}) \]
\[ \# A B C D E F \]  
\[ \# 0.9784548 0.7493803 1.2204900 1.1794393 1.1559273 0.7163082 \]

### 2.5.4 Spatial influence for polygons

Above we looked at adjacency for a set of points. Here we look at it for polygons. The difference is that

\[ \text{library}(raster) \]
\[ p \leftarrow \text{shapefile}(\text{system.file}("external/lux.shp", \text{package}="raster")) \]

To find adjacent polygons, we can use the spdep package.

\[ \text{library}(\text{spdep}) \]

We use poly2nb to create a “rook’s case” neighbors-list. And from that a neighbors matrix.

\[ \text{wr} \leftarrow \text{poly2nb}(p, \text{row.names=}{p$ID_2, \text{queen=FALSE}}) \]
\[ \text{wr} \]
\[ \# Neighbour list object: \]
\[ \# Number of regions: 12 \]
\[ \# Number of nonzero links: 46 \]
\[ \# Percentage nonzero weights: 31.94444 \]
\[ \# Average number of links: 3.833333 \]
\[ \text{wm} \leftarrow \text{nb2mat}(\text{wr}, \text{style='B', zero.policy = TRUE}) \]
\[ \text{dim}(\text{wm}) \]
\[ \# [1] 12 12 \]

Inspect the content or \text{wr} and \text{wm}

\[ \text{wr}[1:6] \]
\[ \# [[1]] \]
\[ \# [1] 2 4 5 \]
\[ \# [[2]] \]
\[ \# [1] 1 3 4 5 6 12 \]
\[ \# [[3]] \]
Compute the number of neighbors for each area.

```r
i <- rowSums(wm)
i
## 1 2 3 4 5 6 7 12 8 9 10 11
## 3 6 4 2 3 3 4 4 3 5 6
```

Expresses as percentage

```r
round(100 * table(i) / length(i), 1)
i
## 2 3 4 5 6
## 8.3 41.7 25.0 8.3 16.7
```

Plot the links between the polygons.

```r
par(mai=c(0,0,0,0))
plot(p, col='gray', border='blue')
xy <- coordinates(p)
plot(wr, xy, col='red', lwd=2, add=TRUE)
```
Now some alternative approaches to compute “spatial influence”.

Distance based:

```r
wd10 <- dnearneigh(xy, 0, 10)
wd25 <- dnearneigh(xy, 0, 25, longlat=TRUE)
```

Nearest neighbors:

```r
k3 <- knn2nb(knearneigh(xy, k=3))
k6 <- knn2nb(knearneigh(xy, k=6))
## Warning in knearneigh(xy, k = 6): k greater than one-third of the number of data points
```

Lag-two Rook:

```r
wr2 <- wr
for (i in 1:length(wr)) {
(continues on next page)
lag1 <- wr[[i]]
lag2 <- wr[lag1]
lag2 <- sort(unique(unlist(lag2)))
lag2 <- lag2[(lag2 %in% c(wr[[i]], i))]
wr2[[i]] <- lag2

And now we plot them all using the plotit function.

plotit <- function(nb, lab='') {
  plot(p, col='gray', border='white')
  plot(nb, xy, add=TRUE, pch=20)
  text(6.3, 50.1, paste0('(' , lab, ')'), cex=1.25)
}

par(mfrow=c(2, 3), mai=c(0,0,0,0))
plotit(wr, 'adjacency')
plotit(wr2, 'lag-2 adj.')
plotit(wd10, '10 km')
plotit(wd25, '25 km')
plotit(k3, 'k=3')
plotit(k6, 'k=6')
2.5. Spatial influence
2.6 Raster based distance metrics

2.6.1 distance

2.6.2 cost distance

2.6.3 resistance distance
3.1 Introduction

Spatial autocorrelation is an important concept in spatial statistics. It is both a nuisance, as it complicates statistical tests, and a feature, as it allows for spatial interpolation. Its computation and properties are often misunderstood. This chapter discusses what it is, and how statistics describing it can be computed.

Autocorrelation (whether spatial or not) is a measure of similarity (correlation) between nearby observations. To understand spatial autocorrelation, it helps to first consider temporal autocorrelation.

3.1.1 Temporal autocorrelation

If you measure something about the same object over time, for example a person’s weight or wealth, it is likely that two observations that are close to each other in time are also similar in measurement. Say that over a couple of years your weight went from 50 to 80 kg. It is unlikely that it was 60 kg one day, 50 kg the next and 80 the day after that. Rather it probably went up gradually, with the occasional tapering off, or even reverse in direction. The same may be true with your bank account, but that may also have a marked monthly trend. To measure the degree of association over time, we can compute the correlation of each observation with the next observation.

Let \( d \) be a vector of daily observations.

```r
set.seed(0)
d <- sample(100, 10)
d
```

Compute auto-correlation.

```r
a <- d[-length(d)]
b <- d[-1]
plot(a, b, xlab='t', ylab='t-1')
```
The autocorrelation computed above is very small. Even though this is a random sample, you (almost) never get a value of zero. We computed the “one-lag” autocorrelation, that is, we compare each value to its immediate neighbour, and not to other nearby values.

After sorting the numbers in d autocorrelation becomes very strong (unsurprisingly).

```
cor(a, b)
## [1] 0.1227634

d <- sort(d)
d
## [1]  1  14  34  39  43  59  68  82  87 100
a <- d[-length(d)]
b <- d[-1]
plot(a, b, xlab='t', ylab='t-1')
```
The `acf` function shows autocorrelation computed in a slightly different way for several lags (it is 1 to each point it self, very high when comparing with the nearest neighbour, and than tapering off).
3.1.2 Spatial autocorrelation

The concept of *spatial* autocorrelation is an extension of temporal autocorrelation. It is a bit more complicated though. Time is one-dimensional, and only goes in one direction, ever forward. Spatial objects have (at least) two dimensions and complex shapes, and it may not be obvious how to determine what is “near”.

Measures of spatial autocorrelation describe the degree to which observations (values) at spatial locations (whether they are points, areas, or raster cells), are similar to each other. So we need two things: observations and locations.

Spatial autocorrelation in a variable can be exogenous (it is caused by another spatially autocorrelated variable, e.g. rainfall) or endogenous (it is caused by the process at play, e.g. the spread of a disease).

A commonly used statistic that describes spatial autocorrelation is Moran’s $I$, and we’ll discuss that here in detail. Other indices include Geary’s $C$ and, for binary data, the join-count index. The semi-variogram also expresses the amount of spatial autocorrelation in a data set (see the chapter on interpolation).
3.2 Example data

Read the example data

```r
library(raster)
p <- shapefile(system.file("external/lux.shp", package="raster"))
p <- p[p$NAME_1="Diekirch", ]
p$value <- c(10, 6, 4, 11, 6)
data.frame(p)
## ID_1 NAME_1 ID_2 NAME_2 AREA value
## 0 1 Diekirch 1 Clervaux 312 10
## 1 1 Diekirch 2 Diekirch 218 6
## 2 1 Diekirch 3 Redange 259 4
## 3 1 Diekirch 4 Vianden 76 11
## 4 1 Diekirch 5 Wiltz 263 6
```

Let’s say we are interested in spatial autocorrelation in variable “AREA”. If there were spatial autocorrelation, regions of a similar size would be spatially clustered.

Here is a plot of the polygons. I use the `coordinates` function to get the centroids of the polygons to place the labels.

```r
par(mai=c(0,0,0,0))
plot(p, col=2:7)
xy <- coordinates(p)
points(xy, cex=6, pch=20, col='white')
text(p, 'ID_2', cex=1.5)
```
3.3 Adjacent polygons

Now we need to determine which polygons are “near”, and how to quantify that. Here we’ll use adjacency as criterion. To find adjacent polygons, we can use package ‘spdep’.

```r
library(spdep)
w <- poly2nb(p, row.names=p$Id)
class(w)
## [1] "nb"
summary(w)
## Neighbour list object:
## Number of regions: 5
## Number of nonzero links: 14
## Percentage nonzero weights: 56
## Average number of links: 2.8
## Link number distribution:
##
## 2 3 4
## 2 2 1
## 2 least connected regions:
```

(continues on next page)
summary(w) tells us something about the neighborhood. The average number of neighbors (adjacent polygons) is 2.8, 3 polygons have 2 neighbors and 1 has 4 neighbors (which one is that?).

For more details we can look at the structure of w.

```
str(w)
## List of 5
## $ : int [1:3] 2 4 5
## $ : int [1:4] 1 3 4 5
## $ : int [1:2] 2 5
## $ : int [1:2] 1 2
## $ : int [1:3] 1 2 3
## - attr(*, "class")= chr "nb"
## - attr(*, "region.id")= chr [1:5] "0" "1" "2" "3" ...
## - attr(*, "call")= language poly2nb(pl = p, row.names = p$Id)
## - attr(*, "type")= chr "queen"
## - attr(*, "sym")= logi TRUE
```

**Question 1:** Explain the meaning of the first 5 lines returned by str(w)

Plot the links between the polygons.

```
plot(p, col='gray', border='blue', lwd=2)
plot(w, xy, col='red', lwd=2, add=TRUE)
```
We can transform $w$ into a spatial weights matrix. A spatial weights matrix reflects the intensity of the geographic relationship between observations (see previous chapter).

```r
wm <- nb2mat(w, style='B')
wm
##  [1,]  0  0  1  0  1  1
##  [2,]  1  1  0  1  1  1
##  [3,]  0  1  0  0  0  1
##  [4,]  1  1  0  0  0  0
##  [5,]  1  1  1  0  0  0
## attr(, "call")
## nb2mat(neighbours = w, style = "B")
```
3.4 Compute Moran’s I

Now let’s compute Moran’s index of spatial autocorrelation

$$I = \frac{n}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij}(y_i - \bar{y})(y_j - \bar{y})}{\sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij}}$$

Yes, that looks impressive. But it is not much more than an expanded version of the formula to compute the correlation coefficient. The main thing that was added is the spatial weights matrix.

The number of observations

```r
n <- length(p)
```

Get ‘y’ and ‘ybar’ (the mean value of y)

```r
y <- p$value
ybar <- mean(y)
```

Now we need

$$(y_i - \bar{y})(y_j - \bar{y})$$

That is, $(y_i - \bar{y})(y_j - \bar{y})$ for all pairs. I show two methods to get that.

Method 1:

```r
dy <- y - ybar
g <- expand.grid(dy, dy)
yiyj <- g[,1] * g[,2]
```

Method 2:

```r
yi <- rep(dy, each=n)
yj <- rep(dy)
yiyj <- yi * yj
```

Make a matrix of the multiplied pairs

```r
pm <- matrix(yiyj, ncol=n)
```

And multiply this matrix with the weights to set to zero the value for the pairs that are not adjacent.

```r
pmw <- pm * wm
pmw
```

## 0    0.00  -3.64  0.00   9.36  -3.64  
## 1   -3.64  0.00  4.76  -5.04  1.96   
## 2    0.00  4.76  0.00   0.00  4.76   
## 3    9.36 -5.04  0.00   0.00  0.00   
## 4   -3.64  1.96  4.76  0.00  0.00   
## attr(,"call")
## nb2mat(neighbours = w, style = "B")
We now sum the values, to get this bit of Moran’s $I$:

$$
\sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} (y_i - \bar{y})(y_j - \bar{y})
$$

```r
spmw <- sum(pmw)
spmw
## [1] 17.04
```

The next step is to divide this value by the sum of weights. That is easy.

```r
smw <- sum(wm)
sw <- spmw / smw
```

And compute the inverse variance of $y$

```r
vr <- n / sum(dy^2)
```

The final step to compute Moran’s $I$

```r
MI <- vr * sw
MI
## [1] 0.1728896
```

This is a simple (but crude) way to estimate the expected value of Moran’s $I$. That is, the value you would get in the absence of spatial autocorrelation (if the data were spatially random). Of course you never really expect that, but that is how we do it in statistics. Note that the expected value approaches zero if $n$ becomes large, but that it is not quite zero for small values of $n$.

```r
EI <- -1/(n-1)
EI
## [1] -0.25
```

After doing this ‘by hand’, now let’s use the spdep package to compute Moran’s $I$ and do a significance test. To do this we need to create a ‘listw’ type spatial weights object (instead of the matrix we used above). To get the same value as above we use ‘style=’B’’ to use binary (TRUE/FALSE) distance weights.

```r
ww <- nb2listw(w, style='B')
```

Now we can use the `moran` function. Have a look at `?moran`. The function is defined as ‘moran(y, ww, n, Szero(ww))’. Note the odd arguments `n` and `S0`. I think they are odd, because “ww” has that information. Anyway, we supply them and it works. There probably are cases where it makes sense to use other values.
moran(p$value, ww, n=length(ww$neighbours), S0=Szero(ww))

## $I
## [1] 0.1728896
##
## $K
## [1] 1.432464

#Note that
Szero(ww)
## [1] 14
# is the same as
pmw
## 0 0.00 -3.64 0.00 9.36 -3.64
## 1 -3.64 0.00 4.76 -5.04 1.96
## 2 0.00 4.76 0.00 0.00 4.76
## 3 9.36 -5.04 0.00 0.00 0.00
## 4 -3.64 1.96 4.76 0.00 0.00
## attr(,”call”)
## nb2mat(neighbours = w, style = “B”)
sum(pmw==0)
## [1] 11

Now we can test for significance. First analytically, using linear regression based logic and assumptions.

moran.test(p$value, ww, randomisation=FALSE)

## Moran I test under normality
##
## data: p$value
## weights: ww
##
## Moran I statistic standard deviate = 2.3372, p-value = 0.009714
## alternative hypothesis: greater
## sample estimates:
## Moran I statistic Expectation Variance
## 0.1728896 -0.2500000 0.0327381

Instead of the approach above you should use Monte Carlo simulation. That is the preferred method (in fact, the only good method). The way it works is that the values are randomly assigned to the polygons, and the Moran’s $I$ is computed. This is repeated several times to establish a distribution of expected values. The observed value of Moran’s $I$ is then compared with the simulated distribution to see how likely it is that the observed values could be considered a random draw.

moran.mc(p$value, ww, nsim=99)

## Monte-Carlo simulation of Moran I
##
## data: p$value
## weights: ww
## number of simulations + 1: 100
##
## statistic = 0.17289, observed rank = 98, p-value = 0.02
Question 2: How do you interpret these results (the significance tests)?

Also try this code, it gives an error: `moran.mc(p$value, ww, nsim=999)`

Question 3: What is the maximum value we can use for `nsim`?

We can make a “Moran scatter plot” to visualize spatial autocorrelation. We first get the neighbouring values for each value.

```r
n <- length(p)
ms <- cbind(id=rep(1:n, each=n), y=rep(y, each=n), value=as.vector(wm * y))
```

Remove the zeros

```r
ms <- ms[ms[,3] > 0, ]
```

And compute the average neighbour value

```r
ams <- aggregate(ms[,2:3], list(ms[,1]), FUN=mean)
ams <- ams[-1]
colnames(ams) <- c('y', 'spatially lagged y')
head(ams)
## y spatially lagged y
## 1 10 7.666667
## 2 6 7.750000
## 3 4 6.000000
## 4 11 8.000000
## 5 6 6.666667
```

Finally, the plot.

```r
plot(ams)
reg <- lm(ams[,2] ~ ams[,1])
abline(reg, lwd=2)
abline(h=mean(ams[,2]), lt=2)
abline(v=ybar, lt=2)
```
Note that the slope of the regression line:

```r
coefficients(reg)[2]
## ams[, 1]
## 0.2315341
```

is almost the same as Moran’s $I$.

Here is a more direct approach to accomplish the same thing (but hopefully the above makes it clearer how this is actually computed). Note the row standardisation of the weights matrix:

```r
rwm <- mat2listw(wm, style='W')
# Checking if rows add up to 1
mat <- list2mat(rwm)
apply(mat, 1, sum)[1:15]
## 0 1 2 3 4 <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA>
## 1 1 1 1 1 NA NA NA NA NA NA NA NA NA
```

Now we can plot

```r
moran.plot(y, rwm)
```

3.4. Compute Moran’s $I$
Question 4: Show how to use the ‘geary’ function to compute Geary’s C

Question 5: Write your own Monte Carlo simulation test to compute p-values for Moran’s I, replicating the results we obtained with the function from spdep. Show a histogram of the simulated values.

Question 6: Write your own Geary C function, by completing the function below

```
gearyC <- ((n-1)/sum( "-----"\^2)) * sum(wm * (" --- ")\^2) / (2 * sum(wm))
```
CHAPTER
FOUR

INTERPOLATION

4.1 Introduction

Almost any variable of interest has spatial autocorrelation. That can be a problem in statistical tests, but it is a very useful feature when we want to predict values at locations where no measurements have been made; as we can generally safely assume that values at nearby locations will be similar. There are several spatial interpolation techniques. We show some of them in this chapter.

4.2 Temperature in California

We will be working with temperature data for California. If have not yet done so, first install the rspatial package to get the data. You may need to install the devtools package first.

```r
if (!require("rspatial")) devtools::install_github('rspatial/rspatial')
```

## Loading required package: rspatial

Now get the data

```r
library(rspatial)
d <- sp_data('precipitation')
head(d)
```

```
# ID   NAME          LAT    LONG     ALT  JAN  FEB  MAR  APR  MAY  JUN  JUL  AUG  SEP  OCT  NOV  DEC
# 1 ID741 DEATH VALLEY 36.47 -116.87 -59 7.4  9.5  7.5  3.4  1.7  1.0  3.7  2.8  4.3  2.2  4.7  3.9
# 2 ID743 THERMAL/FAA AIRPORT 33.63 -116.17 -34 9.2  6.9  7.9  1.8  1.6  0.4  1.9  3.4  5.3  2.0  6.3  5.5
# 3 ID744 BRAKLEY 2SW 32.96 -115.55 -31 11.3  8.3  7.6  2.0  0.8  0.1  1.9  9.2  6.5  5.0  4.8  9.7
# 4 ID753 IMPERIAL/FAA AIRPORT 32.83 -115.57 -18 10.6  7.0  6.1  2.5  0.2  0.0  2.4  2.6  8.3  5.4  7.7  7.3
# 5 ID754 NILAND 33.28 -115.51 -18 9.0  8.0  9.0  3.0  0.0  1.0  8.0  9.0  7.0  8.0  7.0  9.0
# 6 ID758 EL CENTRO/NAF 32.82 -115.67 -13 9.8  1.6  3.7  3.0  0.4  0.0  3.0  10.8  0.2  0.0  3.3  1.4
```

Compute annual precipitation

```r
d$prec <- rowSums(d[, c(6:17)])
plot(sort(d$prec), ylab='Annual precipitation (mm)', las=1, xlab='Stations')
```
Now make a quick map.

```r
library(sp)
dsp <- SpatialPoints(d[,4:3], proj4string=CRS("+proj=longlat +datum=NAD83"))
dsp <- SpatialPointsDataFrame(dsp, d)
CA <- sp_data("counties")

# define groups for mapping
cuts <- c(0,200,300,500,1000,3000)
# set up a palette of interpolated colors
blues <- colorRampPalette(c('yellow', 'orange', 'blue', 'dark blue'))
pols <- list("sp.polygons", CA, fill = "lightgray")
spplot(dsp, 'prec', cuts=cuts, col.regions=blues(5), sp.layout=pols, pch=20, cex=2)
```
Transform longitude/latitude to planar coordinates, using the commonly used coordinate reference system for California ("Teale Albers") to assure that our interpolation results will align with other data sets we have.

```r
TA <- CRS("+proj=aea +lat_1=34 +lat_2=40.5 +lat_0=0 +lon_0=-120 +x_0=0 +y_0=-4000000 +datum=WGS84 +units=m")
library(rgdal)
dta <- spTransform(dsp, TA)
cata <- spTransform(CA, TA)
```
4.2.1 9.2 NULL model

We are going to interpolate (estimate for unsampled locations) the precipitation values. The simplest way would be to take the mean of all observations. We can consider that a “Null-model” that we can compare other approaches to. We’ll use the Root Mean Square Error (RMSE) as evaluation statistic.

```r
RMSE <- function(observed, predicted) {
  sqrt(mean((predicted - observed)^2, na.rm=TRUE))
}
```

Get the RMSE for the Null-model

```r
null <- RMSE(mean(dsp$prec), dsp$prec)
null
# [1] 435.3217
```

4.2.2 proximity polygons

Proximity polygons can be used to interpolate categorical variables. Another term for this is “nearest neighbour” interpolation.

```r
library(dismo)
v <- voronoi(dta)
plot(v)
```
Looks weird. Let's confine this to California

```r
ca <- aggregate(cata)
vca <- intersect(v, ca)
spplot(vca, 'prec', col.regions=rev(get_col_regions()))
```
Much better. These are polygons. We can ‘rasterize’ the results like this.

```r
r <- raster(cata, res=10000)
vr <- rasterize(vca, r, 'prec')
plot(vr)
```
Now evaluate with 5-fold cross validation.

```r
set.seed(5132015)
kf <- kfold(nrow(dta))

rmse <- rep(NA, 5)
for (k in 1:5) {
  test <- dta[kf == k, ]
  train <- dta[kf != k, ]
  v <- voronoi(train)
  p <- extract(v, test)
  rmse[k] <- RMSE(test$prec, p$prec)
}
rmse
## [1] 199.0686 187.8069 166.9153 197.8713 238.9696
mean(rmse)
## [1] 198.1263
```

(continues on next page)

### 4.2. Temperature in California
Question 1: Describe what each step in the code chunk above does

Question 2: How does the proximity-polygon approach compare to the NULL model?

Question 3: You would not typically use proximity polygons for rainfall data. For what kind of data would you use them?

4.2.3 Nearest neighbour interpolation

Here we do nearest neighbour interpolation considering multiple (5) neighbours.

We can use the `gstat` package for this. First we fit a model. ~1 means “intercept only”. In the case of spatial data, that would be only ‘x’ and ‘y’ coordinates are used. We set the maximum number of points to 5, and the “inverse distance power” `idp` to zero, such that all five neighbors are equally weighted.

```r
library(gstat)
gs <- gstat(formula=prec~1, locations=dta, nmax=5, set=list(idp = 0))
nn <- interpolate(r, gs)
## [inverse distance weighted interpolation]
nnmsk <- mask(nn, vr)
plot(nnmsk)
```
Cross validate the result. Note that we can use the predict method to get predictions for the locations of the test points.

```r
rmsenn <- rep(NA, 5)
for (k in 1:5) {
  test <- dta[kf == k, ]
  train <- dta[kf != k, ]
  gscv <- gstat(formula=prec~1, locations=train, nmax=5, set=list(idp = 0))
  p <- predict(gscv, test)$var1.pred
  rmsenn[k] <- RMSE(test$prec, p)
}
```

4.2. Temperature in California
## 

\[
\begin{align*}
\text{mean(rmsenn)} & = 195.9423 \\
1 - \left(\frac{\text{mean(rmsenn)}}{\text{null}}\right) & = 0.5498908
\end{align*}
\]

### 4.2.4 Inverse distance weighted

A more commonly used method is “inverse distance weighted” interpolation. The only difference with the nearest neighbour approach is that points that are further away get less weight in predicting a value a location.

```r
library(gstat)
gs <- gstat(formula=prec~1, locations=dta)
idw <- interpolate(r, gs)
## [inverse distance weighted interpolation]
idwr <- mask(idw, vr)
plot(idwr)
```
Question 4: IDW generated rasters tend to have a noticeable artefact. What is that?

Cross validate. We can predict to the locations of the test points

```r
rmse <- rep(NA, 5)
for (k in 1:5) {
  test <- dta[kf == k, ]
  train <- dta[kf != k, ]
  gs <- gstat(formula=prec~1, locations=train)
  p <- predict(gs, test)
  rmse[k] <- RMSE(test$prec, p$var1.pred)
}

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

(continues on next page)
```
rmse
## [1] 215.3319 211.9383 190.0231 211.8308 230.1893
mean(rmse)
## [1] 211.8627
1 - (mean(rmse) / null)
## [1] 0.5133192
```

**Question 5:** Inspect the arguments used for and make a map of the IDW model below. What other name could you give to this method (IDW with these parameters)? Why?

```
gs2 <- gstat(formula=prec~1, locations=dta, nmax=1, set=list(idp=1))
```

### 4.3 California Air Pollution data

We use California Air Pollution data to illustrate geostatistical (Kriging) interpolation.

#### 4.3.1 Data preparation

We use the airqual dataset to interpolate ozone levels for California (averages for 1980-2009). Use the variable OZDLYAV (unit is parts per billion). Original data source.

Read the data file. To get easier numbers to read, I multiply OZDLYAV with 1000

```
x <- sp_data("airqual")
x$OZDLYAV <- x$OZDLYAV * 1000
```

Create a SpatialPointsDataFrame and transform to Teale Albers. Note the units=km, which was needed to fit the variogram.

```
coordinates(x) <- ~LONGITUDE + LATITUDE
proj4string(x) <- CRS("+proj=longlat +datum=NAD83")
TA <- CRS("+proj=aea +lat_1=34 +lat_2=40.5 +lat_0=0 +lon_0=-120 +x_0=0 +y_0=-4000000␣...
˓
library(rgdal)
aq <- spTransform(x, TA)
```

Create an template raster to interpolate to. E.g., given a SpatialPolygonsDataFrame of California, `ca`. Coerce that to a `SpatialGrid` object (a different representation of the same idea)

```
cageo <- sp_data('counties.rds')
ca <- spTransform(cageo, TA)
r <- raster(ca)
res(r) <- 10  # 10 km if your CRS's units are in km
g <- as(r, 'SpatialGrid')
```
4.3.2 Fit a variogram

Use gstat to create an empirical variogram \( v \)

```r
library(gstat)
gs <- gstat(formula=OZLYAV~1, locations=aq)
v <- variogram(gs, width=20)
head(v)
```

```
  np  dist gamma  dir.hor  dir.ver id
1  1010 11.35040 34.80579    0      0  var1
2  1806 30.63737 47.52591    0      0  var1
3  2355 50.58656 67.26548    0      0  var1
4  2619 70.10411 80.92707    0      0  var1
5  2967 90.13917 88.93653    0      0  var1
6  3437 110.42302 84.13589    0      0  var1
```

```r
plot(v)
```

4.3. California Air Pollution data 45
Now, fit a model variogram

```r
fve <- fit.variogram(v, vgm(85, "Exp", 75, 20))
fve
## model    psill     range
## 1  Nug  21.96600 0.00000
## 2   Exp  85.52957 72.31404
plot(variogramLine(fve, 400), type='l', ylim=c(0,120))
points(v[,2:3], pch=20, col='red')
```

Try a different type (spherical instead of exponential)

```r
fvs <- fit.variogram(v, vgm(85, "Sph", 75, 20))
fvs
## model    psill     range
## 1  Nug  25.57019 0.00000
## 2   Sph  72.65881 135.7744
```

(continues on next page)
plot(variogramLine(fvs, 400), type='l', ylim=c(0,120) ,col='blue', lwd=2)
points(v[,2:3], pch=20, col='red')

Both look pretty good in this case.

Another way to plot the variogram and the model

plot(v, fve)

4.3. California Air Pollution data
### 4.3.3 Ordinary kriging

Use variogram `fve` in a kriging interpolation

```r
k <- gstat(formula=OZDLYAV~1, locations=aq, model=fve)
# predicted values
kp <- predict(k, g)
## [using ordinary kriging]
spplot(kp)
```
# variance
ok <- brick(kp)
ok <- mask(ok, ca)
names(ok) <- c('prediction', 'variance')
plot(ok)
4.3.4 Compare with other methods

Let’s use `gstat` again to do IDW interpolation. The basic approach first.

```r
library(gstat)
idm <- gstat(formula=OZLYAV~1, locations=aq)
idp <- interpolate(r, idm)
## [inverse distance weighted interpolation]
idp <- mask(idp, ca)
plot(idp)
```

We can find good values for the idw parameters (distance decay and number of neighbours) through optimization. For simplicity’s sake I do not do that \( k \) times here. The `optim` function may be a bit hard to grasp at first. But the essence is simple. You provide a function that returns a value that you want to minimize (or maximize) given a number of unknown parameters. Your provide initial values for these parameters, and `optim` then searches for the optimal values (for which the function returns the lowest number).
RMSE <- function(observed, predicted) {
  sqrt(mean((predicted - observed)^2, na.rm=TRUE))
}

f1 <- function(x, test, train) {
  nmx <- x[1]
  idp <- x[2]
  if (nmx < 1) return(Inf)
  if (idp < .001) return(Inf)
  m <- gstat(formula=OZDLYAV~1, locations=train, nmax=nmx, set=list(idp=idp))
  p <- predict(m, newdata=test, debug.level=0)$var1.pred
  RMSE(test$OZDLYAV, p)
}

set.seed(20150518)
i <- sample(nrow(aq), 0.2 * nrow(aq))
tst <- aq[i,]
trn <- aq[-i,]
opt <- optim(c(8, .5), f1, test=tst, train=trn)
opt
## $par
## [1] 9.2594442 0.6817524
##
## $value
## [1] 7.861426
##
## $counts
## function gradient
## 35 NA
##
## $convergence
## [1] 0
##
## $message
## NULL

Our optimal IDW model

m <- gstat(formula=OZDLYAV~1, locations=aq, nmax=opt$par[1], set=list(idp=opt$par[2]))
idw <- interpolate(r, m)
## [inverse distance weighted interpolation]
idw <- mask(idw, ca)
plot(idw)

4.3. California Air Pollution data
A thin plate spline model

```r
library(fields)
m <- Tps(coordinates(aq), aq$OZDLYAV)
tps <- interpolate(r, m)
tps <- mask(tps, idw)
plot(tps)
```
4.3.5 Cross-validate

Cross-validate the three methods (IDW, Ordinary kriging, TPS) and add RMSE weighted ensemble model.

```r
library(dismo)

nfolds <- 5
k <- kfold(aq, nfolds)

ensrmse <- tpsrmse <- krigmse <- idwrmse <- rep(NA, 5)

for (i in 1:nfolds) {
  test <- aq[k!=i,]
  train <- aq[k==i,]
  m <- gstat(formula=OZLYAV~1, locations=train, nmax=opt$par[1], set=list(idp=opt$par[2]))

  # Add code here for cross-validation
  # Example:
  # test_pred <- predict(m, locations=test)
  # ensmse[i] <- mean((test_pred - test$OZLYAV)^2)

  # Add code here for TPS and Ordinary kriging
}
```

(continues on next page)
### Question 6: Which method performed best?

We can use the RMSE values to make a weighted ensemble. I use the inverse of the difference between a model’s RMSE and a NULL model.

```r
nullrmse <- RMSE(test$OZDLYAV, mean(test$OZDLYAV))
w <- 1 / (nullrmse - rms)
weights <- (w / sum(w))
# check
sum(weights)
## [1] 1
s <- stack(idw, ok[[1]], tps)
ensemble <- sum(s * weights)
```

And compare maps.

```r
s <- stack(idw, ok[[1]], tps, ensemble)
names(s) <- c('IDW', 'OK', 'TPS', 'Ensemble')
plot(s)
```
**Question 7:** Show where the largest difference exist between IDW and OK.

**Question 8:** Show where the difference between IDW and OK is within the 95% confidence limit of the OK prediction.

**Question 9:** Can you describe the pattern we are seeing, and speculate about what is causing it?
This page shows how you can use the Random Forest algorithm to make spatial predictions. This approach is widely used, for example to classify remote sensing data into different land cover classes. But here our objective is to predict the entire range of a species based on a set of locations where it has been observed. As an example, we use the hominid species *Imaginus magnapedum* (also known under the vernacular names of “bigfoot” and “sasquatch”). This species is so hard to find (at least by scientists) that its very existence is commonly denied by the mainstream media! For more information about this controversy, see the article by Lozier, Aniello and Hickerson: Predicting the distribution of Sasquatch in western North America: anything goes with ecological niche modelling.

We want to find out

a) What the complete range of the species might be.

b) How good (general) our model is by predicting the range of the Eastern sub-species, with data from the Western sub-species.

c) Predict where in Mexico the creature is likely to occur.

d) How climate change might affect its distribution.

In this context, this type of analysis is often referred to as ‘species distribution modeling’ or ‘ecological niche modeling’. Here is a more in-depth discussion of this technique.

5.1 Data

5.1.1 Observations

```r
if (!require("rspatial")) devtools::install_github('rspatial/rspatial')
library(rspatial)
bf <- sp_data('bigfoot')
dim(bf)
## [1] 3092   3
head(bf)
##          lon     lat Class
## 1 -142.9000 61.50000  A
## 2 -132.7982 55.18720  A
## 3 -132.8202 55.20350  A
## 4 -141.5667 62.93750  A
## 5 -149.7853 61.05950  A
## 6 -141.3165 62.77335  A
```

Plot the locations
5.1.2 Predictors

Supervised classification often uses predictor data obtained from satellite remote sensing. But here, as is common in species distribution modeling, we use climate data. Specifically, we use ‘bioclimatic variables’, see: http://www.worldclim.org/bioclim

```r
library(raster)
w <- raster::getData('worldclim', res=10, var='bio')
plot(w[[c(1, 12)]], nr=2)
```
Now extract climate data for the locations of our observations. That is, get data about the climate that the species likes, apparently.

```r
bfc <- extract(wc, bf[,1:2])
head(bfc)
```

```
## bio1  bio2  bio3  bio4  bio5  bio6  bio7  bio8  bio9  bio10  bio11  bio12  bio13
## [1,] -14  102  27  9672  174 -197  371   51  -11   108 -137  973  119
## [2,]  62  55  31  4136  157  -17  174   43   98  118  156  2602  385
## [3,]  62  55  31  4136  157  -17  174   43   98  118  156  2602  385
## [4,] -57  125  23  15138  206 -332  538  127 -129  127 -256  282  67
## [5,]  10  80  25  8308  174 -140  314   66   5  119  -91  532  81
## [6,] -59  128  23  14923 204 -334  538  122 -130  122 -255  322  75
## bio14 bio15 bio16 bio17 bio18 bio19
## [1,]  43   30  332  156  290  210
## [2,] 128  33  953  407  556  721
## [3,] 128  33  953  407  556  721
## [4,]  6  81  163  22  163  27
## [5,] 22  41  215  72  159  117
## [6,]  8  79  183  28  183  32
```

# Any missing values?
i <- which(is.na(bfc[,1]))

(continues on next page)
Here is a plot that illustrates a component of the ecological niche of our species of interest.

```r
plot(bfc[, 'bio1'] / 10, bfc[, 'bio12'], xlab='Annual mean temperature (C)',
     ylab='Annual precipitation (mm)')
```
5.1.3 Background data

Normally, one would build a model that would compare the values of the predictor variables as the locations where something was observed, with those values at the locations where it was not observed. But we do not have data from a systematic survey that determined presence and absence. We have presence-only data. (And, determining absence is not that simple. It is here now, it is gone tomorrow).

The common trick to deal with this is to not model presence versus absence, but presence versus a ‘random expectation’. This random expectation (also referred to as ‘background’, or ‘random-absence’ data) is what you would get if the species had no preference for any of the predictor variables (or to other variables that are not in the model, but correlated with the predictor variables).

There is not much point in taking absence data from very far away (tropical Africa or Antarctica). Typically they are taken from more or less the entire study area for which we have presences data.

```r
library(dismo)
# extent of all points
e <- extent(SpatialPoints(bf[, 1:2]))
e
## class : Extent
## xmin : -156.75
## xmax : -64.4627
```

(continues on next page)
## ymin : 25.141
## ymax : 69.5

# 5000 random samples (excluding NA cells) from extent e
set.seed(0)
bg <- sampleRandom(wc, 5000, ext=e)
dim(bg)
## [1] 5000 19
head(bg)
##
## bio1 bio2 bio3 bio4 bio5 bio6 bio7 bio8 bio9 bio10 bio11 bio12 bio13
## [1,] 157 126 60 2935 262 55 207 124 191 197 122 379 88
## [2,] -54 105 28 9244 142 -223 365 57 -62 68 -165 639 79
## [3,] -57 104 20 14227 198 -317 515 106 -227 118 -247 473 71
## [4,] 1 119 24 12335 231 -251 482 138 -91 150 -168 844 104
## [5,] 208 169 44 7641 404 28 376 304 239 307 114 198 31
## [6,] -89 111 23 12931 160 -316 476 78 -174 78 -248 476 76
##
## bio14 bio15 bio16 bio17 bio18 bio19
## [1,] 0 100 225 2 4 222
## [2,] 28 30 226 101 219 138
## [3,] 17 46 197 55 194 59
## [4,] 34 33 301 128 291 137
## [5,] 2 50 73 11 52 62
## [6,] 25 40 193 79 193 82

5.1.4 Combine presence and background

d <- rbind(cbind(pa=1, bfc), cbind(pa=0, bg))
d <- data.frame(d)
dim(d)
## [1] 8092 20

5.2 Fit a model

Now we have the data to fit a model. But I am going to split the data into East and West. Let’s say I believe these are actually are different, albeit related, sub-species (The Eastern Sasquatch is darker and less hairy). I am principally interested in the western sub-species.

de <- d[bf[,1] > -102,

dw <- d[bf[,1] <= -102,
5.2.1 CART

Let's first look at a Classification and Regression Trees (CART) model.

```r
library(rpart)
cart <- rpart(pa~., data=dw)
printcp(cart)
##
## Regression tree:
## rpart(formula = pa ~ ., data = dw)
##
## Variables actually used in tree construction:
## [1] bio10 bio14 bio15 bio18 bio3 bio4 bio5 bio6 bio8
##
## Root node error: 762.45/3246 = 0.23489
##
## n= 3246
##
## CP nsplit rel error xerror xstd
## 1 0.410197 0 1.00000 1.00071 0.008912
## 2 0.137588 1 0.58980 0.59037 0.014184
## 3 0.044259 2 0.45222 0.45958 0.016681
## 4 0.029121 3 0.40796 0.43353 0.016756
## 5 0.018954 4 0.37884 0.39714 0.016313
## 6 0.018324 5 0.35988 0.39714 0.016313
## 7 0.010113 6 0.34156 0.37834 0.015761
## 8 0.010008 7 0.33144 0.36700 0.016047
## 9 0.010000 9 0.31143 0.36700 0.016047

plotcp(cart)
```

5.2. Fit a model
And here is the tree

```r
plot(cart, uniform=TRUE, main="Regression Tree")
# text(cart, use.n=TRUE, all=TRUE, cex=.8)
text(cart, cex=.8, digits=1)
```
**Question 1:** Describe the conditions under which you have the highest probability of finding our beloved species?

### 5.2.2 Random Forest

CART gives us a nice result to look at that can be easily interpreted (as you just illustrated with your answer to Question 1). But the approach suffers from high variance (meaning that the model will be over-fit, it is different each time a somewhat different datasets are used). Random Forest does not have that problem as much. Above, with CART, we use regression, let’s do both regression and classification here. First classification.

```r
library(randomForest)
## randomForest 4.6-14
## Type rfNews() to see new features/changes/bug fixes.
# create a factor to indicated that we want classification
fpa <- as.factor(dw[, 'pa'])
```

Now fit the RandomForest model

```r
crf <- randomForest(dw[, 2:ncol(dw)], fpa)
crf
##
## Call:
## randomForest(x = dw[, 2:ncol(dw)], y = fpa)
```

(continues on next page)
Type of random forest: classification
Number of trees: 500
No. of variables tried at each split: 4

OOB estimate of error rate: 9.92%
Confusion matrix:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>class.error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1862</td>
<td>160</td>
<td>0.07912957</td>
</tr>
<tr>
<td>1</td>
<td>162</td>
<td>1062</td>
<td>0.13235294</td>
</tr>
</tbody>
</table>

The variable importance plot shows which variables are most important in fitting the model. This is computing by randomizing each variable one by one and then computing the decline in model prediction.

```r
varImpPlot(crf)
```
Now we use regression, rather than classification. First we tune a parameter.

```
trf <- tuneRF(dw[, 2:ncol(dw)], dw[, 'pa'])
# Warning in randomForest.default(x, y, mtry = mtryStart, ntree = ntreeTry, :
# The response has five or fewer unique values. Are you sure you want to do
# regression?
# mtry = 6  OOB error = 0.07223543
# Searching left ...
# Warning in randomForest.default(x, y, mtry = mtryCur, ntree = ntreeTry, :
# The response has five or fewer unique values. Are you sure you want to do
# regression?
# mtry = 3  OOB error = 0.07002109
# 0.03065438 0.05
# Searching right ...
# Warning in randomForest.default(x, y, mtry = mtryCur, ntree = ntreeTry, :
# The response has five or fewer unique values. Are you sure you want to do
# regression?
# mtry = 12  OOB error = 0.07146042
# 0.01072891 0.05
```

5.2. Fit a model
Question 2: What did tuneRF help us find? What does the values of mt represent?

```r
trf <- tuneRF(dw[, 2:ncol(d)], dw[, 'pa'], mtry=mt)
## Warning in tuneRF.default(dw[, 2:ncol(d)], dw[, "pa"], mtry = mt):
## The response has five or fewer unique values. Are you sure you want to do
## regression?
rrf <- randomForest(dw[, 2:ncol(d)], dw[, 'pa'], mtry=mt)
## Call:
## randomForest(x = dw[, 2:ncol(d)], y = dw[, "pa"], mtry = mt)
## Type of random forest: regression
## Number of trees: 500
## No. of variables tried at each split: 3
```

(continues on next page)
## Mean of squared residuals: 0.06842395
## % Var explained: 70.87

plot(rrf)

varImpPlot(rrf)
5.3 Predict

We can use the model to make predictions to any other place for which we have values for the predictor variables. Our climate data is global so we could find suitable places for bigfoot in Australia. At first I only want to predict to our study region, which I define as follows.

```r
# Extent of the western points
ew <- extent(SpatialPoints(bf[,1] <= -102, 1:2))
```

```r
ew
## class : Extent
## xmin : -156.75
## xmax : -102.3881
## ymin : 30.77722
## ymax : 69.5
```

![Graph showing IncNodePurity vs. various bioclimatic variables](image)

- Bio4
- Bio3
- Bio9
- Bio19
- Bio8
- Bio11
- Bio16
- Bio18
- Bio10
- Bio5
- Bio7
- Bio6
- Bio1
- Bio13
- Bio12
- Bio2
- Bio14
- Bio15
- Bio17

- IncNodePurity
- Values range from 0 to 80

---

Chapter 5. Spatial distribution models
5.3.1 Regression

\[
\text{rp} \leftarrow \text{predict}(\text{wc, rrf, ext=ew}) \\
\text{plot(rp)}
\]

Note that the regression predictions are well-behaved, in the sense that they are between 0 and 1. However, they are continuous within that range, and if you wanted presence/absence, you would need a threshold. To get the optimal threshold, you would normally have a hold out data set, but here I used the training data for simplicity.

\[
\text{eva} \leftarrow \text{evaluate}(\text{dw[dw$pa==1, ], dw[dw$pa==0, ], rrf})
\]

```
## class : ModelEvaluation
## n presences : 1224
## n absences : 2022
## AUC : 0.9993911
## cor : 0.964508
## max TPR+TNR at : 0.4982011
```

We can make a ROC plot

\[
\text{plot(eva, 'ROC')}
\]
Find a good threshold to determine presence/absence and plot the prediction.

```r
tr <- threshold(eva)
tr
## kappa spec_sens no_omission prevalence equal_sens_spec
## thresholds 0.4982011 0.4982011 0.3976191 0.3785858 0.5176767
## sensitivity
## thresholds 0.7221367
plot(rp > tr[1, 'spec_sens'])
```
5.3.2 Classification

We can also use the classification Random Forest model to make a prediction.

```r
cr <- predict(wc, crf, ext=ew)
plot(rc)
```
You can also get probabilities for the classes

```r
rc2 <- predict(wc, crf, ext=ew, type='prob', index=2)
plot(rc2)
```
5.4 Extrapolation

Now, let’s see if our model is general enough to predict the distribution of the Eastern species.

```r
de <- na.omit(de)
eva2 <- evaluate(de[de$pa==1, ], de[de$pa==0, ], rrf)
eva2
## class : ModelEvaluation
## n presences : 1866
## n absences : 2978
## AUC : 0.4271887
## cor : -0.260545
## max TPR+TNR at : 3e-04
plot(eva2, 'ROC')```

5.4. Extrapolation
We can also look at it on a map.

eus <- extent(SpatialPoints(bf[, 1:2]))
eus
## class : Extent
## xmin : -156.75
## xmax : -64.4627
## ymin : 25.141
## ymax : 69.5
rcusa <- predict(wc, rrf, ext=eus)
plot(rcusa)
points(bf[,1:2], cex=.25)
Question 3: Why would it be that the model does not extrapolate well?

An important question in the biogeography of the western species is why it does not occur in Mexico. Or if it does, where would that be?

Let's see.

```r
mex <- getData('GADM', country='MEX', level=1)
predict(wc, rrf, ext=mex)
plot(pm)
```
Question 4: Where in Mexico are you most likely to encounter western bigfoot?

We can also estimate range shifts due to climate change

```r
fut <- getData('CMIP5', res=10, var='bio', rcp=85, model='AC', year=70)
## Warning in showSRID(SRS_string, format = "PROJ", multiline = "NO", prefer_proj =
## prefer_proj): Discarded datum unknown in Proj4 definition
## Warning in showSRID(SRS_string, format = "PROJ", multiline = "NO", prefer_proj =
## prefer_proj): Discarded datum unknown in Proj4 definition
## Warning in showSRID(SRS_string, format = "PROJ", multiline = "NO", prefer_proj =
## prefer_proj): Discarded datum unknown in Proj4 definition
## Warning in showSRID(SRS_string, format = "PROJ", multiline = "NO", prefer_proj =
## prefer_proj): Discarded datum unknown in Proj4 definition
## Warning in showSRID(SRS_string, format = "PROJ", multiline = "NO", prefer_proj =
## prefer_proj): Discarded datum unknown in Proj4 definition
## Warning in showSRID(SRS_string, format = "PROJ", multiline = "NO", prefer_proj =
## prefer_proj): Discarded datum unknown in Proj4 definition
```

(continues on next page)
names(fut) <- names(wc)
futusa <- predict(fut, rrf, ext=eus, progress='window')

(continues on next page)
## Question 5: Make a map to show where conditions are improving for western bigfoot, and where they are not. Is the species headed toward extinction?

### 5.5 Further reading

More on Species distribution modeling with R; and on the use of boosted regression trees in the same context.
Regression models are typically “global”. That is, all data are used simultaneously to fit a single model. In some cases it can make sense to fit more flexible “local” models. Such models exist in a general regression framework (e.g. generalized additive models), where “local” refers to the values of the predictor values. In a spatial context local refers to location. Rather than fitting a single regression model, it is possible to fit several models, one for each location (out of possibly very many) locations. This technique is sometimes called “geographically weighted regression” (GWR). GWR is a data exploration technique that allows to understand changes in importance of different variables over space (which may indicate that the model used is misspecified and can be improved).

There are two examples here. One short example with California precipitation data, and than a more elaborate example with house price data.

### 6.1 California precipitation

```r
if (!require("rspatial")) devtools::install_github('rspatial/rspatial')

library(rspatial)
counties <- sp_data('counties')
p <- sp_data('precipitation')
head(p)
## ID NAME LAT LONG ALT JAN FEB MAR APR MAY JUN JUL
## 1 ID741 DEATH VALLEY 36.47 -116.87 -59 7.4 9.5 7.5 3.4 1.7 1.0 3.7
## 2 ID743 THERMAL/FAA AIRPORT 33.63 -116.17 -34 9.2 6.9 7.9 1.8 1.6 0.4 1.9
## 3 ID744 BRAWLEY 2SW 32.96 -115.55 -31 11.3 8.3 7.6 2.0 0.8 0.1 1.9
## 4 ID753 IMPERIAL/FAA AIRPORT 32.83 -115.57 -18 10.6 7.0 6.1 2.5 0.2 0.0 2.4
## 5 ID754 NILAND 33.28 -115.51 -18 9.0 8.0 9.0 3.0 0.0 1.0 8.0
## 6 ID758 EL CENTRO/NAF 32.82 -115.67 -13 9.8 1.6 3.7 3.0 0.4 0.0 3.0
## AUG SEP OCT NOV DEC
## 1 2.8 4.3 2.2 4.7 3.9
## 2 3.4 5.3 2.0 6.3 5.5
## 3 9.2 6.5 5.0 4.8 9.7
## 4 2.6 8.3 5.4 7.7 7.3
## 5 9.0 7.0 8.0 7.0 9.0
## 6 10.8 0.2 0.0 3.3 1.4

plot(counties)
## Warning in wkt(obj): CRS object has no comment
points(p[,c('LONG', 'LAT')], col='red', pch=20)
```
Compute annual average precipitation

```r
p$pan <- rowSums(p[,6:17])
```

Global regression model

```r
m <- lm(pan ~ ALT, data=p)
m
## Call:
## lm(formula = pan ~ ALT, data = p)
## Coefficients:
## (Intercept) ALT
## 523.60 0.17
```

Create Spatial* objects with a planar crs.

```r
alb <- CRS("+proj=aea +lat_1=34 +lat_2=40.5 +lat_0=0 +lon_0=-120 +x_0=0 +y_0=-4000000 +ellps=GRS80 +datum=NAD83 +units=m +no_defs")
sp <- p
coordinates(sp) = ~ LONG + LAT
crs(sp) <- "+proj=longlat +datum=NAD83"
```

(continues on next page)
Spatial Data Analysis with R

spt <- spTransform(sp, alb)
c tst <- spTransform(counties, alb)

## Warning in spTransform(xSP, CRSobj, ...): NULL source CRS comment, falling back
## to PROJ string
## Warning in wkt(obj): CRS object has no comment

Get the optimal bandwidth

library(spgwr)

## NOTE: This package does not constitute approval of GWR
## as a method of spatial analysis; see example(gwr)
bw <- gwr.sel(pan ~ ALT, data=spt)

## Bandwidth: 526221.1 CV score: 64886883
## Bandwidth: 850593.6 CV score: 74209073
## Bandwidth: 325747.9 CV score: 54001118
## Bandwidth: 201848.6 CV score: 44611213
## Bandwidth: 125274.7 CV score: 35746320
## Bandwidth: 77949.39 CV score: 29181737
## Bandwidth: 48700.74 CV score: 22737197
## Bandwidth: 30624.09 CV score: 17457161
## Bandwidth: 19452.1 CV score: 15163436
## Bandwidth: 12547.43 CV score: 19452191
## Bandwidth: 22792.75 CV score: 15512988
## Bandwidth: 17052.67 CV score: 15709960
## Bandwidth: 20218.99 CV score: 15167438
## Bandwidth: 19767.99 CV score: 15156913
## Bandwidth: 19790.05 CV score: 15156906
## Bandwidth: 19781.39 CV score: 15156902
## Bandwidth: 19781.48 CV score: 15156902
## Bandwidth: 19781.47 CV score: 15156902
## Bandwidth: 19781.47 CV score: 15156902
## Bandwidth: 19781.47 CV score: 15156902
## Bandwidth: 19781.47 CV score: 15156902
## Bandwidth: 19781.47 CV score: 15156902
## Bandwidth: 19781.47 CV score: 15156902
bw

## [1] 19781.47

Create a regular set of points to estimate parameters for.

r <- raster(ctst, res=10000)
r <- rasterize(ctst, r)
newpts <- rasterToPoints(r)

Run the gwr function

g <- gwr(pan ~ ALT, data=spt, bandwidth=bw, fit.points=newpts[, 1:2])

## Warning in proj4string(data): CRS object has comment, which is lost in output

g

## Call:
## gwr(formula = pan ~ ALT, data = spt, bandwidth = bw, fit.points = newpts[, 
##     1:2])
## Kernel function: gwr.Gauss
## Fixed bandwidth: 19781.47
## Fit points: 4087
### Summary of GWR coefficient estimates at fit points:

<table>
<thead>
<tr>
<th></th>
<th>Min.</th>
<th>1st Qu.</th>
<th>Median</th>
<th>3rd Qu.</th>
<th>Max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>X.Intercept.</td>
<td>-702.40121</td>
<td>79.54254</td>
<td>330.48807</td>
<td>735.42718</td>
<td>3468.8702</td>
</tr>
<tr>
<td>ALT</td>
<td>-3.91270</td>
<td>0.03058</td>
<td>0.20461</td>
<td>0.41542</td>
<td>4.6133</td>
</tr>
</tbody>
</table>

Link the results back to the raster

```r
slope <- r
intercept <- r
slope[!is.na(slope)] <- g$SDF$ALT
intercept[!is.na(intercept)] <- g$SDF$'Intercept'
s <- stack(intercept, slope)
names(s) <- c('intercept', 'slope')
plot(s)
```

![Image of raster plots showing intercept and slope](image)

### 6.2 California House Price Data


```r
houses <- sp_data("houses1990.csv")
dim(houses)
## [1] 20640 9
head(houses)
## houseValue income houseAge rooms bedrooms population households latitude
## 1 452600 8.3252 41 880 129 322 126 37.88
## 2 358500 8.3014 21 7099 1106 2401 1138 37.86
## 3 352100 7.2574 52 1467 190 496 177 37.85
```

(continues on next page)
Each record represents a census “blockgroup”. The longitude and latitude of the centroids of each block group are available. We can use that to make a map and we can also use these to link the data to other spatial data. For example to get county-membership of each block group. To do that, let’s first turn this into a SpatialPointsDataFrame to find out to which county each point belongs.

```r
library(sp)
coordinates(houses) <- ~longitude+latitude
plot(houses, cex=0.5, pch=1, axes=TRUE)
```

Now get the county boundaries and assign CRS of the houses data matches that of the counties (because they are both
in longitude/latitude!)

```r
library(raster)
crs(houses) <- crs(counties)
```

Do a spatial query (points in polygon)

```r
cnty <- over(houses, counties)
head(cnty)
##  STATE COUNTY NAME LSAD LSAD_TRANS
## 1  06  001 Alameda 06 County
## 2  06  001 Alameda 06 County
## 3  06  001 Alameda 06 County
## 4  06  001 Alameda 06 County
## 5  06  001 Alameda 06 County
## 6  06  001 Alameda 06 County
```

### 6.3 Summarize

We can summarize the data by county. First combine the extracted county data with the original data.

```r
hd <- cbind(data.frame(houses), cnty)
```

Compute the population by county

```r
totpop <- tapply(hd$population, hd$NAME, sum)
totpop
##          Alameda   Alpine   Amador    Butte    Calaveras
##        1241779   1113   30039  182120      31998
##          Colusa   Contra    Costa Del Norte   El Dorado    Fresno
##         16275   799017   16045   128624      662261
##          Glenn   Humboldt   Imperial      Inyo     Kern
##         24798   116418  108633     18281      52895
##         Kings    Lake   Lassen  Los Angeles      Madera
##        91842   50631    27214   8721937      88089
##          Marin   Mariposa   Mendocino      Merced    Modoc
##        204241   14302   75061    176457      9678
##          Mono   Monterey      Napa   Nevada    Orange
##         9956   342314  108030     78510  2340204
##          Placer   Plumas   Riverside    Sacramento    San Benito
##       170761   19739  1162787  1038540      36697
##        San Bernardino   San Diego   San Francisco   San Joaquin   San Luis Obispo
##      1409740  2425153    683068  477184       203764
##          San Mateo   Santa Barbara   Santa Clara   Santa Cruz    Shasta
##       614816   335177   1486054    216732     147036
##          Sierra   Siskiyou   Solano   Sonoma    Stanislaus
##         3318   43531   337429    385296     370821
##          Sutter   Tehama   Trinity   Tulare    Tuolumne
##        63689   49625   13063    309073     48456
##          Ventura    Yolo      Yuba
##        649935   138799     58954
```
Income is harder because we have the median household income by blockgroup. But it can be approximated by first computing total income by blockgroup, summing that, and dividing that by the total number of households.

```r
# total income
hd$suminc <- hd$income * hd$households
# now use aggregate (similar to tapply)
csum <- aggregate(hd[, c('suminc', 'households')], list(hd$NAME), sum)
# divide total income by number of households
# sort
csum$income <- 10000 * csum$suminc / csum$households
csum <- csum[order(csum$income), ]
head(csum)
## Group.1 suminc households income
## 53 Trinity 11198.985 5156 21720.30
## 58 Yuba 43739.708 19882 21999.65
## 25 Modoc 8260.597 3711 22259.76
## 47 Siskiyou 38769.952 17302 22407.79
## 17 Lake 47612.899 20805 22885.32
## 11 Glenn 20497.683 8821 23237.37
tail(csum)
## Group.1 suminc households income
## 56 Ventura 994094.8 210418 47243.81
## 7 Contra Costa 1441734.6 299123 48198.72
## 30 Orange 3938638.1 800968 49173.48
## 43 Santa Clara 2621895.6 518674 50553.87
## 41 San Mateo 1169145.6 230674 50683.89
## 21 Marin 436808.4 85869 50869.17
```

### 6.4 Regression

Before we make a regression model, let’s first add some new variables that we might use, and then see if we can build a regression model with house price as dependent variable. The authors of the paper used a lot of log transforms, so you can also try that.

```r
hd$roomhead <- hd$rooms / hd$population
hd$bedroomhead <- hd$bedrooms / hd$population
hd$hhsize <- hd$population / hd$households
```

Ordinary least squares regression:

```r
# OLS
m <- glm( houseValue ~ income + houseAge + roomhead + bedroomhead + population, data=hd)
summary(m)
## Call:
## glm(formula = houseValue ~ income + houseAge + roomhead + bedroomhead +
##     population, data = hd)
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1226134 -48590  -12944  34425  461948
```

(continues on next page)
## Coefficients:

|                     | Estimate | Std. Error | t value | Pr(>|t|) |
|---------------------|----------|------------|---------|----------|
| (Intercept)         | -6.508e+04 | 2.533e+03  | -25.686 | < 2e-16 *** |
| income              | 5.179e+04  | 3.833e+02  | 135.092 | < 2e-16 *** |
| houseAge            | 1.832e+03  | 4.575e+01  | 40.039  | < 2e-16 *** |
| roomAge             | -4.720e+04 | 1.489e+03  | -31.688 | < 2e-16 *** |
| bedroomhead         | 2.648e+05  | 6.820e+03  | 38.823  | < 2e-16 *** |
| population          | 3.947e+00  | 5.081e-01  | 7.769   | 8.27e-15 *** |

---

# Signif. codes:   ***   **   *   .  1

# (Dispersion parameter for gaussian family taken to be 6022427437)

## Null deviance: 2.7483e+14 on 20639 degrees of freedom
## Residual deviance: 1.2427e+14 on 20634 degrees of freedom
## AIC: 523369

## Number of Fisher Scoring iterations: 2

### coefficients(m)

<table>
<thead>
<tr>
<th>(Intercept)</th>
<th>income</th>
<th>houseAge</th>
<th>roomhead</th>
<th>bedroomhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>-65075.701407</td>
<td>51786.005862</td>
<td>1831.685266</td>
<td>-47198.908765</td>
<td></td>
</tr>
<tr>
<td>population</td>
<td>3.947461</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Geographically Weighted Regression

#### 6.5 Geographically Weighted Regression

##### 6.5.1 By county

Of course we could make the model more complex, with e.g. squared income, and interactions. But let's see if we can do Geographically Weighted regression. One approach could be to use counties.

First I remove records that were outside the county boundaries

```r
hd2 <- hd[!is.na(hd$NAME), ]
```

Then I write a function to get what I want from the regression (the coefficients in this case)

```r
regfun <- function(x) {
  dat <- hd2[hd2$NAME == x, ]
  m <- glm(houseValue~income+houseAge+roomhead+bedroomhead+population, data=dat)
  coefficients(m)
}
```

And now run this for all counties using `sapply`:

```r
countynames <- unique(hd2$NAME)
res <- sapply(countynames, regfun)
```

Plot of a single coefficient
6.5. Geographicaly Weighted Regression

```r
dotchart(sort(res['income'], ), cex=0.65)
```
Chapter 6. Local regression
Spatial Data Analysis with R

There clearly is variation in the coefficient ($\beta$) for income. How does this look on a map?

First make a data.frame of the results

```r
resdf <- data.frame(NAME=colnames(res), t(res))
head(resdf)
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>X.Intercept.</th>
<th>income</th>
<th>houseAge</th>
<th>roomhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alameda</td>
<td>-62373.62</td>
<td>35842.330</td>
<td>591.1001</td>
<td>24147.3182</td>
</tr>
<tr>
<td>Contra Costa</td>
<td>-61759.84</td>
<td>43668.442</td>
<td>465.8897</td>
<td>-356.6085</td>
</tr>
<tr>
<td>Alpine</td>
<td>-77605.93</td>
<td>40850.588</td>
<td>5595.4113</td>
<td>NA</td>
</tr>
<tr>
<td>Amador</td>
<td>-12480.71</td>
<td>3234.519</td>
<td>-771.587</td>
<td>37997.0069</td>
</tr>
<tr>
<td>Butte</td>
<td>50935.36</td>
<td>15577.745</td>
<td>-380.5824</td>
<td>9078.9315</td>
</tr>
<tr>
<td>Calaveras</td>
<td>91364.72</td>
<td>7126.668</td>
<td>-929.4065</td>
<td>16843.3456</td>
</tr>
</tbody>
</table>

Fix the counties object. There are too many counties because of the presence of islands. I first aggregate (‘dissolve’ in GIS-speak) the counties such that a single county becomes a single (multi-)polygon.

```r
dim(counties)
dcounties <- aggregate(counties, vars='NAME')
```

Now we can merge this SpatialPolygonsDataFrame with data.frame with the regression results.

```r
cnres <- merge(dcounties, resdf, by='NAME')
spplot(cnres, 'income')
```

6.5. Geographically Weighted Regression
To show all parameters in a ‘conditioning plot’, we need to first scale the values to get similar ranges.

```r
# a copy of the data
cnres2 <- cnres

# scale all variables, except the first one (county name)
# assigning values to a "@data" slot is risky, but (I think) OK here
cnres2@data = data.frame(scale(data.frame(cnres)[, -1]))
spplot(cnres2)
```

## Warning in wkt(obj): CRS object has no comment

## Warning in wkt(obj): CRS object has no comment
Is this just random noise, or is there spatial autocorrelation?

```r
library(spdep)
b <- poly2nb(cnres)
pplot(cnres)
## Warning in wkt(obj): CRS object has no comment
pplot(nb, coordinates(cnres), add=T, col='red')
```
lw <- nb2listw(nb)
moran.test(cnres$income, lw)
##
## Moran I test under randomisation
##
## data: cnres$income
## weights: lw
##
## Moran I statistic standard deviate = 2.2473, p-value = 0.01231
## alternative hypothesis: greater
## sample estimates:
## Moran I statistic Expectation Variance
## 0.173419996 -0.017543860 0.007220867

moran.test(cnres$roomhead, lw, na.action=na.omit)
##
## Moran I test under randomisation
##
## data: cnres$roomhead
## weights: lw
## omitted: 2
##
## Moran I statistic standard deviate = 1.3929, p-value = 0.08183
## alternative hypothesis: greater
## sample estimates:
## Moran I statistic Expectation Variance
## 0.102596252 -0.017857143 0.007478348

### 6.5.2 By grid cell

An alternative approach would be to compute a model for grid cells. Let’s use the ‘Teale Albers’ projection (often used when mapping the entire state of California).

```r
TA <- CRS("+proj=aea +lat_1=34 +lat_2=40.5 +lat_0=0 +lon_0=-120 +x_0=0 +y_0=-400000 +datum=NAD83 +units=m +no_defs +ellps=GRS80 +towgs84=0,0,0")

countiesTA <- spTransform(counties, TA)

## Warning in spTransform(xSP, CRSobj, ...): NULL source CRS comment, falling back
## to PROJ string
## Warning in wkt(obj): CRS object has no comment

Create a RasteLayer using the extent of the counties, and setting an arbitrary resolution of 50 by 50 km cells

```r
library(raster)
r <- raster(countiesTA)
res(r) <- 50000

Get the xy coordinates for each raster cell:

```r
xy <- xyFromCell(r, 1:ncell(r))

For each cell, we need to select a number of observations, let’s say within 50 km of the center of each cell (thus the data that are used in different cells overlap). And let’s require at least 50 observations to do a regression.

First transform the houses data to Teale-Albers

```r
housesTA <- spTransform(houses, TA)

## Warning in spTransform(xSP, CRSobj, ...): NULL source CRS comment, falling back
## to PROJ string
## Warning in wkt(obj): CRS object has no comment

Set up a new regression function.

```r
regfun2 <- function(d) {
  m <- glm(houseValue~income+houseAge+roomhead+bedroomhead+population, data=d)
  coefficients(m)
}

Run the model for all cells if there are at least 50 observations within a radius of 50 km.

```r
res <- list()
for (i in 1:nrow(xy)) {
  d <- sqrt((xy[i,1]-crds[,1])^2 + (xy[i,2]-crds[,2])^2)
  j <- which(d < 50000)
  if (length(j) > 49) {
    res[i] <- regfun2(housesTA[j,])
  }
}
```

(continues on next page)
For each cell get the income coefficient:

```r
ingc <- sapply(res, function(x) x['income'])
```

Use these values in a RasterLayer

```r
rinc <- setValues(r, inc)
plot(rinc)
plot(countiesTA, add=T)
```

So that was a lot of ‘home-brew-GWR’.

**Question 1**: Can you comment on weaknesses (and perhaps strengths) of the approaches I have shown?
Now use the `spgwr` package (and the `gwr` function) to fit the model. You can do this with all data, as long as you supply and argument `fit.points` (to avoid estimating a model for each observation point. You can use a raster similar to the one I used above (perhaps disaggregate with a factor 2 first).

This is how you can get the points to use:

Create a RasterLayer with the correct extent

```r
r <- raster(countiesTA)
```

Set to a desired resolution. I choose 25 km

```r
res(r) <- 25000
```

I only want cells inside of CA, so I add some more steps.

```r
c <- rasterize(countiesTA, r)
```

Extract the coordinates that are not NA.

```r
fitpoints <- rasterToPoints(c)
```

I don’t want the third column

```r
fitpoints <- fitpoints[, -3]
```

Now specify the model

```r
gwr.model <- ______
```

`gwr` returns a list-like object that includes (as first element) a `SpatialPointsDataFrame` that has the model coefficients. Plot these using `spplot`, and after that, transfer them to a `RasterBrick` object.

To extract the `SpatialPointsDataFrame`:

```r
sp <- gwr.model$SDF
spplot(sp)
```

To reconnect these values to the raster structure (etc.)

```r
cells <- cellFromXY(r, fitpoints)
zd <- as.matrix(data.frame(sp))
b <- brick(r, values=FALSE, nl=ncol(dd))
b[cells] <- zd
names(b) <- colnames(dd)
plot(b)
```

**Question 2:** Briefly comment on the results and the differences (if any) with the two home-brew examples.
CHAPTER
SEVEN

SPATIAL REGRESSION MODELS

7.1 Introduction

This chapter deals with the problem of inference in (regression) models with spatial data. Inference from regression models with spatial data can be suspect. In essence this is because nearby things are similar, and it may not be fair to consider individual cases as independent (they may be pseudo-replicates). Therefore, such models need to be diagnosed before reporting them. Specifically, it is important to evaluate the for spatial autocorrelation in the residuals (as these are supposed to be independent, not correlated). If the residuals are spatially autocorrelated, this indicates that the model is misspecified. In that case you should try to improve the model by adding (and perhaps removing) important variables. If that is not possible (either because there is no data available, or because you have no clue as to what variable to look for), you can try formulating a regression model that controls for spatial autocorrelation. We show some examples of that approach here.

7.2 Reading & aggregating data

We use California house price data from the 2000 Census.

7.2.1 Get the data

```r
if (!require("rspatial")) devtools::install_github('rspatial/rspatial')
library(rspatial)
h <- sp_data('houses2000')
```

I have selected some variables on on housing and population. You can get more data from the American Fact Finder http://factfinder2.census.gov (among other web sites).

```r
library(raster)
dim(h)
## [1] 7049 29
names(h)
## [1] "TRACT"  "GEOID"  "label"  "houseValue"  "nhousingUn"
## [6] "recHouses" "nMobileHom" "yearBuilt" "nBadPlumbi" "nBadKitche"
## [11] "nRooms"  "nBedrooms" "medHHinc"  "Population" "Males"
## [16] "Females"  "Under5"  "MedianAge" "White"  "Black"
## [21] "AmericanIn" "Asian"  "Hispanic"  "PopInHouse"  "nHousehold"
## [26] "Families"  "householdS" "familySize"  "County"
```
These are the variables we have:

<table>
<thead>
<tr>
<th>variable</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>nhousingUn</td>
<td>number of housing units</td>
</tr>
<tr>
<td>recHouses</td>
<td>number of houses for recreational use</td>
</tr>
<tr>
<td>nMobileHom</td>
<td>number of mobile homes</td>
</tr>
<tr>
<td>nBadPlumbi</td>
<td>number of houses with incomplete plumbing</td>
</tr>
<tr>
<td>nBadKitche</td>
<td>number of houses with incomplete kitchens</td>
</tr>
<tr>
<td>Population</td>
<td>total population</td>
</tr>
<tr>
<td>Males</td>
<td>number of males</td>
</tr>
<tr>
<td>Females</td>
<td>number of females</td>
</tr>
<tr>
<td>Under5</td>
<td>number of persons under five</td>
</tr>
<tr>
<td>White</td>
<td>number of persons identifying themselves as white (only)</td>
</tr>
<tr>
<td>Black</td>
<td>number of persons identifying themselves African-american (only)</td>
</tr>
<tr>
<td>AmericanIn</td>
<td>number of persons identifying themselves American Indian (only)</td>
</tr>
<tr>
<td>Asian</td>
<td>number of persons identifying themselves as American Indian (only)</td>
</tr>
<tr>
<td>Hispanic</td>
<td>number of persons identifying themselves as hispanic (only)</td>
</tr>
<tr>
<td>PopInHouse</td>
<td>number of persons living in households</td>
</tr>
<tr>
<td>nHousehold</td>
<td>number of households</td>
</tr>
<tr>
<td>Families</td>
<td></td>
</tr>
</tbody>
</table>
number of families
houseValue
value of the house
yearBuilt
year house was built
nRooms
median number of rooms per house
nBedrooms
median number of bedrooms per house
medHHinc
median household income
MedianAge
median age of population
householdS
median household size
familySize
median family size

First some data massaging. These are values for Census tracts. I want to analyze these data at the county level. So we need to aggregate the values.

```r
hh <- aggregate(h, "County")
```

Now we have the county outlines, but we also need to get the values of interest at the county level. Although it is possible to do everything in one step in the aggregate function, I prefer to do this step by step. The simplest case is where we can sum the numbers. For example for the number of houses.

```r
d1a <- aggregate(d1, list(County=h$County), sum, na.rm=TRUE)
```

In other cases we need to use a weighted mean. For example for houseValue

```r
d2 <- data.frame(h)[, c("houseValue", "yearBuilt", "nRooms", "nBedrooms", "medHHinc", "MedianAge", "householdS", "familySize")]
d2 <- cbind(d2 * h$nHousehold, hh=h$nHousehold)
```
d2a <- aggregate(d2, list(County=h$County), sum, na.rm=TRUE)
d2a[, 2:ncol(d2a)] <- d2a[, 2:ncol(d2a)] / d2a$hh

Combine these two groups:

d12 <- merge(d1a, d2a, by='County')

And merge the aggregated (from census tract to county level) attribute data with the aggregated polygons

hh <- merge(hh, d12, by='County')

Let’s make some maps, at the original Census tract level. We are using a bit more advanced (and slower) plotting methods here. First the house value, using a legend with 10 intervals.

library(latticeExtra)
## Loading required package: lattice
library(RColorBrewer)

grps <- 10
brks <- quantile(h$houseValue, 0:(grps-1)/(grps-1), na.rm=TRUE)

p <- spplot(h, "houseValue", at=brks, col.regions=rev(brewer.pal(grps, "RdBu")), col="transparent")
## Warning in wkt(obj): CRS object has no comment
## Warning in wkt(obj): CRS object has no comment
p + layer(sp.polygons(hh))
This takes very long. `spplot` (levelplot) is a bit slow when using a large dataset...

A map of the median household income.

```r
brks <- quantile(h$medHHinc, 0:(grps-1)/(grps-1), na.rm=TRUE)
p <- spplot(h, "medHHinc", at=brks, col.regions=rev(brewer.pal(grps, "RdBu")), col="transparent")
## Warning in wkt(obj): CRS object has no comment
## Warning in wkt(obj): CRS object has no comment
p + layer(sp.polygons(hh))
```

7.2. Reading & aggregating data
7.3 Basic OLS model

I'll now make some models with the county-level data. I first compute some new variables (that I might not all use).

```r
hh$fBadP <- pmax(hh$nBadPlumbi, hh$nBadKitche) / hh$nhousingUn
hh$fWhite <- hh$White / hh$Population
hh$age <- 2000 - hh$yearBuilt

f1 <- houseValue ~ age + nBedrooms
m1 <- lm(f1, data=hh)
summary(m1)
```

(continues on next page)
Just for illustration, here is how you can do OLS with matrix algebra. First set up the data. I add a constant variable ‘1’ to X, to get an intercept.

```r
y <- matrix(hh$houseValue)
X <- cbind(1, hh$age, hh$nBedrooms)
```

Then use matrix algebra

```r
ols <- solve(t(X) %*% X) %*% t(X) %*% y
rownames(ols) <- c('intercept', 'age', 'nBedroom')
ols
```

So, according to this simple model, “age” is highly significant. The older a house, the more expensive. You pay 1,269,475 dollars more for a house that is 100 years old than a for new house! While the p-value for the number of bedrooms is not impressive, but every bedroom adds about 200,000 dollars to the value of a house.

**Question 1:** What would be the price be of a house built in 1999 with three bedrooms?

Let’s see if the errors (model residuals) appear to be randomly distributed in space.
What do think? Is this random? Let’s see what Mr. Moran would say. First make a neighborhoods list. I add two links: between San Francisco and Marin County and vice versa (to consider the Golden Gate bridge).

```r
library(spdep)
nb <- poly2nb(hh)
nb[[21]] <- sort(as.integer(c(nb[[21]], 38)))
nb[[38]] <- sort(as.integer(c(21, nb[[38]])))
nb
```

```
## Neighbour list object:
## Number of regions: 58
## Number of nonzero links: 278
## Percentage nonzero weights: 8.263971
## Average number of links: 4.793103
```

```r
par(mai=c(0,0,0,0))
plot(hh)
```

```
## Warning in wkt(obj): CRS object has no comment
```

```r
plot(nb, coordinates(hh), col='red', lwd=2, add=TRUE)
```
We can use the neighbour list object to get the average value for the neighbors of each polygon.

```r
resnb <- sapply(nb, function(x) mean(hh$residuals[x]))
cor(hh$residuals, resnb)
## [1] 0.6311218
plot(hh$residuals, resnb, xlab='Residuals', ylab='Mean adjacent residuals')
```
lw <- nb2listw(nb)

That does not look independent.

moran.mc(hh$residuals, lw, 999)
##
## Monte-Carlo simulation of Moran I
##
## data: hh$residuals
## weights: lw
## number of simulations + 1: 1000
##
## statistic = 0.41428, observed rank = 1000, p-value = 0.001
## alternative hypothesis: greater

Clearly, there is spatial autocorrelation. Our $p$-values and regression model coefficients cannot be trusted. So let’s try SAR models.
Here I show a how to do spatial regression with a spatial lag model (lagsarlm), using the spatialreg package.

```r
library(spatialreg)

m1s = lagsarlm(f1, data=hh, lw, tol.solve=1e-30)

summary(m1s)
```

```
## Call: lagsarlm(formula = f1, data = hh, listw = lw, tol.solve = 1e-30)
## ## Residuals:
##    Min     1Q  Median     3Q    Max
## -108145.2 -49816.3  -1316.3  44604.9 171536.0
## ## Type: lag
## ## Coefficients: (asymptotic standard errors)
##                Estimate  Std. Error z value Pr(>|z|)
## (Intercept) -418674.1    153693.6  -2.724 0.00645
## age          5533.6        1698.2   3.258 0.00112
## nBedrooms    127912.8     50859.7   2.515 0.01190
## ## Rho: 0.77413, LR test value: 34.761, p-value: 3.7282e-09
## ## Asymptotic standard error: 0.08125
## ## z-value: 9.5277, p-value: < 2.22e-16
## ## Wald statistic: 90.778, p-value: < 2.22e-16
## ## Log likelihood: -727.9964 for lag model
## ## ML residual variance (sigma squared): 3871700000, (sigma: 62223)
## ## Number of observations: 58
## ## Number of parameters estimated: 5
## ## AIC: 1466, (AIC for lm: 1498.8)
## ## LM test for residual autocorrelation
## ## test value: 0.12431, p-value: 0.72441
## ## Monte-Carlo simulation of Moran I
## ## data: hh$residuals
## ## weights: lw
## ## number of simulations + 1: 1000
## ## statistic = -0.016, observed rank = 509, p-value = 0.491
## ## alternative hypothesis: greater
```

```r
hh$residuals <- residuals(m1s)
moran.mc(hh$residuals, lw, 999)
```

```r
## Monte-Carlo simulation of Moran I
## data: hh$residuals
## weights: lw
## number of simulations + 1: 1000
## statistic = -0.016, observed rank = 509, p-value = 0.491
## alternative hypothesis: greater
```

```r
brks <- quantile(hh$residuals, 0:(grps-1)/(grps-1), na.rm=TRUE)
p <- sspplot(hh, "residuals", at=brks, col.regions=rev(brewer.pal(grps, "RdBu")), col="transparent")
```

(continues on next page)
And now with a “Spatial error” (or spatial moving average) models (errorsarlm)

```r
m1e <- errorsarlm(f1, data=hh, lw, tol.solve=1.0e-30)
summary(m1e)
```

## Warning in wkt(obj): CRS object has no comment
print( p + layer(sp.polygons(hh)) )

7.5 Spatial error model

And now with a “Spatial error” (or spatial moving average) models (errorsarlm)

```r
m1e <- errorsarlm(f1, data=hh, lw, tol.solve=1.0e-30)
summary(m1e)
```

## Warning in wkt(obj): CRS object has no comment
print( p + layer(sp.polygons(hh)) )

```r
m1e <- errorsarlm(f1, data=hh, lw, tol.solve=1.0e-30)
summary(m1e)
```
## Estimate Std. Error z value Pr(>|z|)
## (Intercept) -185443.8 180133.7 -1.0295 0.30325
## age 4313.6 2214.9 1.9475 0.05147
## nBedrooms 117864.4 51564.7 2.2858 0.02227
##
## Lambda: 0.82151, LR test value: 29.781, p-value: 4.8373e-08
## Asymptotic standard error: 0.071111
## z-value: 11.552, p-value: < 2.22e-16
## Wald statistic: 133.46, p-value: < 2.22e-16
##
## Log likelihood: -730.4863 for error model
## ML residual variance (sigma squared): 4.07e+09, (sigma: 63797)
## Number of observations: 58
## Number of parameters estimated: 5
## AIC: 1471, (AIC for lm: 1498.8)

hh$residuals <- residuals(m1e)
moran.mc(hh$residuals, lw, 999)

## Monte-Carlo simulation of Moran I
##
## data: hh$residuals
## weights: lw
## number of simulations + 1: 1000
##
## statistic = 0.039033, observed rank = 743, p-value = 0.257
## alternative hypothesis: greater

brks <- quantile(hh$residuals, 0:(grps-1)/(grps-1), na.rm=TRUE)
p <- spplot(hh, "residuals", at=brks, col.regions=rev(brewer.pal(grps, "RdBu")),
col="transparent")
## Warning in wkt(obj): CRS object has no comment
## Warning in wkt(obj): CRS object has no comment
print( p + layer(sp.polygons(hh)) )
Are the residuals spatially autocorrelated for either of these models? Let’s plot them for the spatial error model.

```r
brks <- quantile(hh$residuals, 0:(grps-1)/(grps-1), na.rm=TRUE)

p <- spplot(hh, "residuals", at=brks, col.regions=rev(brewer.pal(grps, "RdBu")),
            col="transparent")

## Warning in wkt(obj): CRS object has no comment

## Warning in wkt(obj): CRS object has no comment

print( p + layer(sp.polygons(hh)) )
```
7.6 Questions

**Question 2:** The last two maps still seem to show a lot of spatial autocorrelation. But according to the tests there is none. Now why might that be?

**Question 3:** One of the most important, or perhaps THE most important aspect of modeling is variable selection. A misspecified model is never going to be any good, no matter how much you do to, e.g., correct for spatial autocorrelation.

a) Which variables would you choose from the list?

b) Which new variables could you propose to create from the variables in the list.

c) Which other variables could you add, created from the geometries/location (perhaps other geographic data).

d) add a lot of variables and use stepAIC to select an ‘optimal’ OLS model

e) check for spatial autocorrelation in the residuals of that model
8.1 Introduction

We are using a dataset of crimes in a city. Start by reading in the data.

```r
if (!require("rspatial")) devtools::install_github('rspatial/rspatial')
library(rspatial)
city <- sp_data('city')
crime <- sp_data('crime')
```

Here is a map of both datasets.

```r
plot(city, col='light blue')
## Warning in wkt(obj): CRS object has no comment
points(crime, col='red', cex=.5, pch='+')
```

A sorted table of the incidence of crime types.

```r
tb <- sort(table(crime$CATEGORY))[-1]
tb
##
```

(continues on next page)
## Arson Weapons Robbery
## 9 15 49
## Auto Theft Drugs or Narcotics Commercial Burglary
## 86 134 143
## Grand Theft Assaults DUI
## 143 172 212
## Residential Burglary Vehicle Burglary Drunk in Public
## 219 221 232
## Vandalism Petty Theft
## 355 665

Let’s get the coordinates of the crime data, and for this exercise, remove duplicate crime locations. These are the ‘events’ we will use below (later we’ll go back to the full data set).

```r
xy <- coordinates(crime)
dim(xy)
## [1] 2661  2
xy <- unique(xy)
dim(xy)
## [1] 1208  2
head(xy)
## coords.x1 coords.x2
## [1,] 6628868 1963718
## [2,] 6632796 1964362
## [3,] 6636855 1964873
## [4,] 6626493 1964343
## [5,] 6639506 1966094
## [6,] 6640478 1961983
```

### 8.2 Basic statistics

Compute the mean center and standard distance for the crime data.

```r
# mean center
mc <- apply(xy, 2, mean)
# standard distance
sd <- sqrt(sum((xy[,1] - mc[1])^2 + (xy[,2] - mc[2])^2) / nrow(xy))
```

Plot the data to see what we’ve got. I add a summary circle (as in Fig 5.2) by dividing the circle in 360 points and compute bearing in radians. I do not think this is particularly helpful, but it might be in other cases. And it is always fun to figure out how to do tis.

```r
plot(city, col='light blue')
## Warning in wkt(obj): CRS object has no comment
points(crime, cex=.5)
points(cbind(mc[1], mc[2]), pch='*', col='red', cex=5)

# make a circle
bearing <- 1:360 * pi/180
cx <- mc[1] + sd * cos(bearing)
```

(continues on next page)
8.3 Density

Here is a basic approach to computing point density.

```r
CityArea <- raster::area(city)
dens <- nrow(xy) / CityArea
```

**Question 1a:** What is the unit of ‘dens’?

**Question 1b:** What is the number of crimes per square km?

To compute quadrat counts I first create quadrats (a RasterLayer). I get the extent for the raster from the city polygon, and then assign an an arbitrary resolution of 1000. (In real life one should always try a range of resolutions, I think).

```r
r <- raster(city)
res(r) <- 1000
```

```
# class : RasterLayer
# dimensions : 15, 34, 510  (nrow, ncol, ncell)
# resolution : 1000, 1000  (x, y)
# extent   : 6620591, 6654591, 1956519, 1971519  (xmin, xmax, ymin, ymax)
# crs      : +proj=lcc +lat_1=38.33333333333334 +lat_2=39.83333333333334 +lat_0=37.66666666666666 +lon_0=-122 +x_0=2000000 +y_0=500000.00000001 +datum=NAD83 +units=us-ft +no_defs +ellps=GRS80 +towgs84=0,0,0
```
To find the cells that are in the city, and for easy display, I create polygons from the RasterLayer.

```r
r <- rasterize(city, r)
plot(r)
quads <- as(r, 'SpatialPolygons')
plot(quads, add=TRUE)
points(crime, col='red', cex=.5)
```

The number of events in each quadrat can be counted using the `rasterize` function. That function can be used to summarize the number of points within each cell, but also to compute statistics based on the `marks` (attributes). For example we could compute the number of different crime types by changing the `fun` argument to another function (see `?rasterize`).

```r
nc <- rasterize(coordinates(crime), r, fun='count', background=0)
plot(nc)
plot(city, add=TRUE)
```
nc has crime counts. As we only have data for the city, the areas outside of the city need to be excluded. We can do that with the mask function (see ?mask).

```r
ncrimes <- mask(nc, r)
plot(ncrimes)
plot(city, add=TRUE)
```

Better. Now the frequencies.

```r
f <- freq(ncrimes, useNA='no')
head(f)
## value count
## [1,] 0 48
```
## 

```
## [2,] 1 29
## [3,] 2 24
## [4,] 3 22
## [5,] 4 19
## [6,] 5 16
```

```
plot(f, pch=20)
```

Does this look like a pattern you would have expected? Now compute average number of cases per quadrat.

```
# number of quadrats
quadrats <- sum(f[,2])
# number of cases
cases <- sum(f[,1] * f[,2])
mu <- cases / quadrats
```

```
## [1] 9.261484
```

And create a table like Table 5.1 on page 130

```
ff <- data.frame(f)
colnames(ff) <- c('K', 'X')
ff$Kmu <- ff$K - mu
```

(continues on next page)
ff$Kmu2 <- ff$Kmu^2
ff$XKmu2 <- ff$Kmu2 * ff$X
head(ff)

<table>
<thead>
<tr>
<th></th>
<th>K</th>
<th>X</th>
<th>Kmu</th>
<th>Kmu2</th>
<th>XKmu2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>48</td>
<td>-9.261484</td>
<td>85.77509</td>
<td>4117.2042</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>29</td>
<td>-8.261484</td>
<td>68.25212</td>
<td>1979.3115</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>24</td>
<td>-7.261484</td>
<td>52.72915</td>
<td>1265.4996</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>22</td>
<td>-6.261484</td>
<td>39.20618</td>
<td>862.5360</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>19</td>
<td>-5.261484</td>
<td>27.68321</td>
<td>525.9811</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>16</td>
<td>-4.261484</td>
<td>18.16025</td>
<td>290.5639</td>
</tr>
</tbody>
</table>

The observed variance $s^2$ is

```
s2 <- sum(ff$XKmu2) / (sum(ff$X)-1)
s2
## [1] 276.5555
```

And the VMR is

```
VMR <- s2 / mu
VMR
## [1] 29.86082
```

**Question 2:** What does this VMR score tell us about the point pattern?

### 8.4 Distance based measures

As we are using a *planar coordinate system* we can use the `dist` function to compute the distances between pairs of points. If we were using longitude/latitude we could compute distance via spherical trigonometry functions. These are available in the sp, raster, and notably the geosphere package (among others). For example, see `raster::pointDistance`.

```
d <- dist(xy)
class(d)
## [1] "dist"
```

I want to coerce the `dist` object to a matrix, and ignore distances from each point to itself (the zeros on the diagonal).

```
dm <- as.matrix(d)
dm[1:5, 1:5]
##   1   2   3   4   5
## 1  0.000 3980.843 8070.429 2455.809 10900.016
## 2  3980.843 0.000 4090.992 6303.450 6929.439
## 3  8070.429 4090.992 0.000 10375.958 2918.349
## 4 2455.809 6303.450 10375.958 0.000 13130.236
## 5 10900.016 6929.439 2918.349 13130.236 0.000
diag(dm) <- NA
dm[1:5, 1:5]
##   1   2   3   4   5
## 1 NA 3980.843 8070.429 2455.809 10900.016
## 2 3980.843 NA 4090.992 6303.450 6929.439
## 3 8070.429 4090.992 NA 10375.958 2918.349
## 42455.809 6303.450 10375.958 NA 13130.236
## 510900.016 6929.439 2918.349 13130.236 NA
```

(continues on next page)
To get, for each point, the minimum distance to another event, we can use the `apply` function. Think of the rows as each point, and the columns of all other points (vice versa could also work).

```r
dmin <- apply(dm, 1, min, na.rm=TRUE)
head(dmin)
## 1 2 3 4 5 6
## 266.07892 293.58874 47.90260 140.80688 40.06865 510.41231
```

Now it is trivial to get the mean nearest neighbour distance according to formula 5.5, page 131.

```r
mdmin <- mean(dmin)
```

Do you want to know, for each point, Which point is its nearest neighbour? Use the `which.min` function (but note that this ignores the possibility of multiple points at the same minimum distance).

```r
wdmin <- apply(dm, 1, which.min)
```

And what are the most isolated cases? That is the furthest away from their nearest neighbour. I plot the top 25. A bit complicated.

```r
plot(city)
## Warning in wkt(obj): CRS object has no comment
points(crime, cex=.1)
ord <- rev(order(dmin))

far25 <- ord[1:25]
neighbors <- wdmin[far25]

points(xy[far25, ], col='blue', pch=20)
points(xy[neighbors, ], col='red')

# drawing the lines, easiest via a loop
for (i in far25) {
  lines(rbind(xy[i, ], xy[wdmin[i], ]), col='red')
}
```
Note that some points, but actually not that many, are used as isolated and as a neighbor to an isolated points.

Now on to the G function

```r
max(dmin)
## [1] 1829.738
# get the unique distances (for the x-axis)
distance <- sort(unique(round(dmin)))
# compute how many cases there with distances smaller that each x
Gd <- sapply(distance, function(x) sum(dmin < x))
# normalize to get values between 0 and 1
Gd <- Gd / length(dmin)
plot(distance, Gd)
```
# using xlim to exclude the extremes
plot(distance, Gd, xlim=c(0, 500))
Here is a function to show these values in a more standard way.

```r
define stepplot <- function(x, y, type='l', add=FALSE, ...) {
  x <- as.vector(t(cbind(x, c(x[-1], x[length(x)]))))
  y <- as.vector(t(cbind(y, y)))
  if (add) {
    lines(x, y, ...)
  } else {
    plot(x, y, type=type, ...)
  }
}
```

And use it for our G function data.

```r
stepplot(distance, Gd, type='l', lwd=2, xlim=c(0,500))
```
The steps are so small in our data, that you hardly see the difference.

I use the centers of previously defined raster cells to compute the $F$ function.

```r
# get the centers of the 'quadrats' (raster cells)
p <- rasterToPoints(r)
# compute distance from all crime sites to these cell centers
d2 <- pointDistance(p[,1:2], xy, longlat=FALSE)

# the remainder is similar to the G function
Fdistance <- sort(unique(round(d2)))
mind <- apply(d2, 1, min)
Fd <- sapply(Fdistance, function(x) sum(mind < x))
Fd <- Fd / length(mind)
plot(Fdistance, Fd, type='l', lwd=2, xlim=c(0,3000))
```
Compute the expected distribution (5.12 on page 145)

```r
ef <- function(d, lambda) {
  E <- 1 - exp(-1 * lambda * pi * d^2)
}
expected <- ef(0:2000, dens)
```

Now, let's combine F and G on one plot.

```r
plot(distance, Gd, type='l', lwd=2, col='red', las=1,
     ylab='F(d) or G(d)', xlab='Distance', yaxs="i", xaxs="i")
lines(Fdistance, Fd, lwd=2, col='blue')
lines(0:2000, expected, lwd=2)
legend(1200, .3,
       c(expression(italic("G")["d"]), expression(italic("F")["d"]), 'expected'),
       lty=1, col=c('red', 'blue', 'black'), lwd=2, bty="n")
```

8.4. Distance based measures
Question 3: What does this plot suggest about the point pattern?

Finally, let’s compute K. Note that I use the original distance matrix ‘d’ here.

```r
distance <- seq(1, 30000, 100)
Kd <- sapply(distance, function(x) sum(d < x)) # takes a while
Kd <- Kd / (length(Kd) * dens)
plot(distance, Kd, type='l', lwd=2)
```
Question 4: Create a single random pattern of events for the city, with the same number of events as the crime data (object xy). Use function ‘spsample’

Question 5: Compute the G function, and plot it on a single plot, together with the G function for the observed crime data, and the theoretical expectation (formula 5.12).

Question 6: (Difficult!) Do a Monte Carlo simulation (page 149) to see if the ‘mean nearest distance’ of the observed crime data is significantly different from a random pattern. Use a ‘for loop’. First write ‘pseudo-code’. That is, say in natural language what should happen. Then try to write R code that implements this.

8.5 Spatstat package

Above we did some ‘home-brew’ point pattern analysis, we will now use the spatstat package. In research you would normally use spatstat rather than your own functions, at least for standard analysis. I showed how you make some of these functions in the previous sections, because understanding how to go about that may allow you to take things in directions that others have not gone. The good thing about spatstat is that it very well documented (see http://spatstat.github.io/). The bad thing is that it uses an entirely different sets of classes (ways to represent spatial data) that we will use in all other labs (classes from sp and raster); but it is not hard to get used to that.

```
library(spatstat)
```

We start with making a Kernel Density raster. I first create a ‘ppp’ (point pattern) object, as defined in the spatstat package.
Spacial Data Analysis with R

A `ppp` object has the coordinates of the points and the analysis ‘window’ (study region). To assign the points locations we need to extract the coordinates from our `SpatialPoints` object. To set the window, we first need to to coerce our `SpatialPolygons` into an ‘owin’ object. We need a function from the `maptools` package for this coercion.

Coerce from `SpatialPolygons` to an object of class “owin” (observation window)

```r
library(maptools)
cityOwin <- as.owin(city)
## Warning in wkt(obj): CRS object has no comment
## Warning in wkt(obj): CRS object has no comment
## Warning in wkt(obj): CRS object has no comment
## Warning in wkt(obj): CRS object has no comment
class(cityOwin)
## [1] "owin"
cityOwin
## window: polygonal boundary
## enclosing rectangle: [6620591, 6654380] x [1956729.8, 1971518.9] units
```

Extract coordinates from `SpatialPointsDataFrame`:

```r
pts <- coordinates(crime)
head(pts)
## coords.x1 coords.x2
## [1,] 6628868 1963718
## [2,] 6632796 1964362
## [3,] 6636855 1964873
## [4,] 6626493 1964343
## [5,] 6639506 1966094
## [6,] 6640478 1961983
```

Now we can create a ‘ppp’ (point pattern) object

```r
p <- ppp(pts[,1], pts[,2], window=cityOwin)
## Warning: 20 points were rejected as lying outside the specified window
## Warning: data contain duplicated points
class(p)
## [1] "ppp"
p
## Planar point pattern: 2641 points
## window: polygonal boundary
## enclosing rectangle: [6620591, 6654380] x [1956729.8, 1971518.9] units
## *** 20 illegal points stored in attr(,"rejects") ***
plot(p)
## Warning in plot.ppp(p): 20 illegal points also plotted
```
Note the warning message about ‘illegal’ points. Do you see them and do you understand why they are illegal?

Having all the data well organized, it is now easy to compute Kernel Density

```r
ds <- density(p)
class(ds)
# [1] "im"
plot(ds, main='crime density')
```
Density is the number of points per unit area. Let’s check if the numbers make sense, by adding them up and multiplying with the area of the raster cells. I use raster package functions for that.

```r
nrow(pts)
## [1] 2661
r <- raster(ds)
s <- sum(values(r), na.rm=TRUE)
s * prod(res(r))
## [1] 2640.556
```

Looks about right. We can also get the information directly from the “im” (image) object.

```r
str(ds)
## List of 10
##$ v : num [1:128, 1:128] NA NA NA NA NA NA NA NA NA NA ...
##$ dim : int [1:2] 128 128
##$ xrange: num [1:2] 6620591 6654380
##$ yrange: num [1:2] 1956730 1971519
##$ xstep : num 264
##$ ystep : num 116
##$ xcol : num [1:128] 6620723 6620987 6621251 6621515 6621779 ...
##$ yrow : num [1:128] 1956788 1956903 1957019 1957134 1957250 ...
##$ type : chr "real"
```

(continues on next page)
Spatial Data Analysis with R

## units :List of 3
## ..$ singular : chr "unit"
## ..$ plural : chr "units"
## ..$ multiplier: num 1
## ..- attr(*, "class")= chr "unitname"
## - attr(*, "class")= chr "im"
## - attr(*, "sigma")= num 1849
## - attr(*, "kernel")= chr "gaussian"

sum(ds$v, na.rm=TRUE) * ds$xstep * ds$ystep
## [1] 2640.556

p$n
## [1] 2641

Here’s another, lengthy, example of generalization. We can interpolate population density from (2000) census data; assigning the values to the centroid of a polygon (as explained in the book, but not a great technique). We use a shapefile with census data.

census <- sp_data("census2000.rds")

To compute population density for each census block, we first need to get the area of each polygon. I transform density from persons per feet\(^2\) to persons per mile\(^2\), and then compute population density from POP2000 and the area.

census$area <- area(census)

Now to get the centroids of the census blocks we can use the ‘coordinates’ function again. Note that it actually does something quite different (with a SpatialPolygons* object) then in the case above (with a SpatialPoints* object).

p <- coordinates(census)
head(p)
## [,1]      [,2]
## 0 6666671 1991720
## 1 6655379 1986903
## 2 6604777 1982474
## 3 6612242 1981881
## 4 6613488 1986776
## 5 6616743 1986446

To create the ‘window’ we dissolve all polygons into a single polygon.

win <- aggregate(census)

Let’s look at what we have:
Now we can use ‘Smooth.ppp’ to interpolate. Population density at the points is referred to as the ‘marks’

```r
owin <- as.owin(win)
pp <- ppp(p[,1], p[,2], window=owin, marks=census$dens)
```

Note the warning message: “1 point was rejected as lying outside the specified window”. That is odd, there is a polygon that has a centroid that is outside of the polygon. This can happen with, e.g., kidney shaped polygons.

Let’s find and remove this point that is outside the study area.
sp <- SpatialPoints(p, proj4string=CRS(proj4string(win)))
## Warning in proj4string(win): CRS object has comment, which is lost in output
library(rgeos)
i <- gIntersects(sp, win, byid=TRUE)
which(!i)
## [1] 588

Let's see where it is:

plot(census)
## Warning in wkt(obj): CRS object has no comment
points(sp)
points(sp[i,], col='red', cex=3, pch=20)

You can zoom in using the code below. After running the next line, click on your map twice to zoom to the red dot, otherwise you cannot continue:

zoom(census)

And add the red points again

points(sp[i,], col='red')

To only use points that intersect with the window polygon, that is, where ‘i == TRUE’:
Spatial Data Analysis with R

```
pp <- ppp(p[i,1], p[i,2], window=owin, marks=census$dens[i])
plot(pp)
plot(city, add=TRUE)
```

And to get a smooth interpolation of population density.

```
s <- Smooth.ppp(pp)
## Warning: Least Squares Cross-Validation criterion was minimised at right-hand
## end of interval [89.7, 3350]; use arguments 'hmin', 'hmax' to specify a wider
## interval for bandwidth 'sigma'
plot(s)
plot(city, add=TRUE)
```
Population density could establish the “population at risk” (to commit a crime) for certain crimes, but not for others. Maps with the city limits and the incidence of ‘auto-theft’, ‘drunk in public’, ‘DUI’, and ‘Arson’.

```r
par(mfrow=c(2,2), mai=c(0.25, 0.25, 0.25, 0.25))
for (offense in c("Auto Theft", "Drunk in Public", "DUI", "Arson")) {
  plot(city, col='grey')
  acrime <- crime[crime$CATEGORY == offense, ]
  points(acrime, col = "red")
  title(offense)
}
## Warning in wkt(obj): CRS object has no comment
## Warning in wkt(obj): CRS object has no comment
## Warning in wkt(obj): CRS object has no comment
## Warning in wkt(obj): CRS object has no comment
## Warning in wkt(obj): CRS object has no comment
```

8.5. Spatstat package

137
Create a marked point pattern object (ppp) for all crimes. It is important to coerce the marks to a factor variable.

```r
crime$fcat <- as.factor(crime$CATEGORY)
w <- as.owin(city)
## Warning in wkt(obj): CRS object has no comment
## Warning in wkt(obj): CRS object has no comment
## Warning in wkt(obj): CRS object has no comment
## Warning in wkt(obj): CRS object has no comment
xy <- coordinates(crime)
mpp <- ppp(xy[,1], xy[,2], window = w, marks=crime$fcat)
## Warning: 20 points were rejected as lying outside the specified window
## Warning: data contain duplicated points
```

We can split the mpp object by category (crime)

```r
spp <- split(mpp)
plot(spp[1:4], main='')
```

Create a marked point pattern object (ppp) for all crimes. It is important to coerce the marks to a factor variable.
The crime density by category:

```r
plot(density(spp[1:4]), main='')
```

And produce K-plots (with an envelope) for ‘drunk in public’ and ‘Arson’. Can you explain what they mean?

```r
spatstat.options(checksegments = FALSE)
ktheft <- Kest(spp$"Auto Theft")
ketheft <- envelope(spp$"Auto Theft", Kest)
```
## Generating 99 simulations of CSR ...  
## 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99.  
## Done.

```r
ktheft <- Kest(spp$"Arson")
ketheft <- envelope(spp$"Arson", Kest)
## Generating 99 simulations of CSR ...  
## 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99.  
## Done.
```

```r
par(mfrow=c(1,2))
plot(ktheft)
plot(ketheft)
```

Let's try to answer the question you have been wanting to answer all along. Is population density a good predictor of being (booked for) “drunk in public” and for “Arson”? One approach is to do a Kolmogorov-Smirnov (‘kstest’) on ‘Drunk in Public’ and ‘Arson’, using population density as a covariate:

```r
KS.arson <- cdf.test(spp$"Arson", ds)
KS.arson
```

(continues on next page)
**Question 7**: *Why is the result surprising, or not surprising?*

We can also compare the patterns for “drunk in public” and for “Arson” with the KCross function.

```r
kc <- Kcross(mpp, i = "Drunk in Public", j = "Arson")
ekc <- envelope(mpp, Kcross, nsim = 50, i = "Drunk in Public", j = "Arson")
## Generating 50 simulations of CSR ...
## 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50.
## Done.
plot(ekc)
```
Much more about point pattern analysis with spatstat is available here.