Notes for STAT 5413 - Spatial Statistics

John Tipton

Fall 2020 Semester. Last Modified: 2020-01-13

Contents

Chapter 1

STAT 5413

These are the lecture notes for STAT 5413 Fall 2020.

Chapter 2

Day 1

```r
library(tidyverse)
```

2.1 Notation

The dimensions of different mathematical objects are very important for the study of spatial statistics. To communicate this, we use the following notation. A scalar random variable is represented by a lowercase alphanumeric letter (x, y, z, etc.), a vector random variable is respresented by a bold lowercase alphanumeric letter ($\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$, etc.), and a matrix random variable is respresented by a bold uppercase alphanumeric letter ($\mathbf{X}$, $\mathbf{Y}$, $\mathbf{Z}$, etc.). We use a similar notation for parameters as well where scalar parameters are represented by a lowercase Greek letter (μ, α, β, etc.), a vector parameter is respresented by a bold lowercase Greek letter (μ, α, β, etc.), and a matrix random variable is respresented by a bold uppercase Greek letter (Σ, Ψ, Γ, etc.).

2.2 Probability Distributions

We also need notation to explain probability distributions. We use the notation $[y]$ to denote the probability density function $p(y)$ of the random variable y and $[y|x]$ to denote the probability density function $p(y|x)$ of y given x. For example, if y is a Gaussian random variable with mean μ and standard deviation σ we write

$$[y|\mu, \sigma] = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2\sigma^2}(y-\mu)^2\right\}.$$

We can also denote that y has a Gaussian (normal) distribution given mean μ and variance σ^2 using the $\sim$ notation

$$y|\mu, \sigma \sim \mathrm{N}(\mu, \sigma^2).$$

2.2.1 Example: linear regression

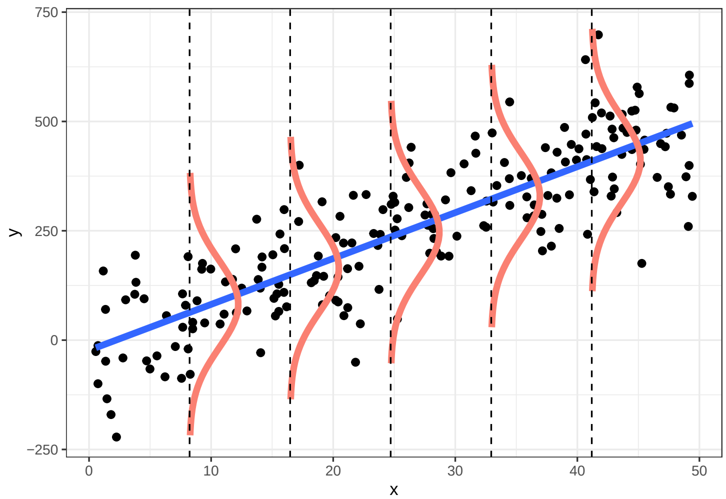
$$\begin{gathered}[y_i|\theta] \sim \mathrm{N}(X_i\beta, \sigma^2) \\ \theta = (\beta, \sigma^2)\end{gathered}$$

```r
## Sample data
set.seed(404)
dat <- data.frame(x=(x=runif(200, 0, 50)),
                  y=rnorm(200, 10 * x, 100))

## breaks: where you want to compute densities
breaks <- seq(0, max(dat$x), len=7)[-c(1, 7)]
dat$section <- cut(dat$x, breaks)

## Get the residuals
dat$res <- residuals(lm(y ~ x, data=dat))

## Compute densities for each section, and flip the axes, and add means of sections
## Note: the densities need to be scaled in relation to the section size (2000 here)
ys <- seq(-300, 300, length = 50)
xs <- rep(breaks, each = 50) + 1000 * dnorm(ys, 0, 100)
res <- matrix(0, 50, 5)
for (i in 1:5) {
  res[, i] <- 10 * breaks[i] + ys
}
dens <- data.frame(x = xs, y=c(res),
                   grouping = cut(xs, breaks))
ggplot(dat, aes(x, y)) +
  geom_point(size = 2) +
  geom_smooth(method="lm", fill=NA, lwd=2, se = FALSE) +
  geom_path(data=dens, aes(x, y, group = grouping),
            color="salmon", lwd=2) +
  theme_bw() +
  geom_vline(xintercept=breaks, lty=2)
```



2.3 Hierarchical modeling

- Follow Berliner (1996) framework for hierarchical probability models
- Model encodes our understanding of the scientific process of interest
- Model accounts for as much uncertainty as possible
- Model results in a probability distribution
  - Note: nature may be deterministic – often probabilistic models outperform physical models.
  - Example: model individual rain drops vs. probability/intensity of rain
- Update model with data
- Use the model to generate parameter estimates given data

2.3.1 Bayesian Hierarchical models (BHMs)

- Break the model into components:
  - Data Model.
  - Process Model.
  - Parameter Model.

- Combined, the data model, the process model, and the parameter model define a posterior distribution.

$$[\mathbf{z}, \theta_D, \theta_P|\mathbf{y}] \propto [\mathbf{y}|\theta_D, \mathbf{z}][\mathbf{z}|\theta_P][\theta_D][\theta_P]$$

2.3.2 Empirical Hierarchical models (EHMs)

- Break the model into components:
  - Data Model.
  - Process Model.
  - Parameter estimates (fixed values) are substituted before fitting the model
- Combined, the data model and the process model define a predictive distribution. Thus, numerical evaluation of the predictive distribution is typically required to estimate unceratinty (bootstrap, MLE asymptotics)
  - Note: the predictive distribution is not a posterior distribution because the normalizing constant is not known

$$[\mathbf{z}|\mathbf{y}] \propto [\mathbf{y}|\theta_D, \mathbf{z}][\mathbf{z}|\theta_P]$$

2.3.3 Data Model

$$[\mathbf{y}|\theta_D, \mathbf{z}]$$

- Describes how the data are collected and observed.
- Account for measurement process and uncertainty.
- Model the data in the manner in which they were collected.
- Data $\mathbf{y}$.
  - Noisy.
  - Expensive.
  - Not what you want to make inference on.
- Latent variables $\mathbf{z}$.

  - Think of **z** as the ideal data.
  - No measurement error - the exact quantity you want to observe but can't.

- Data model parameters θ_D.

2.3.4 Process Model

$$[\mathbf{z}|\theta_P]$$

- **Where the science happens!**
- Latent process **z** is modeled.
- Can be dynamic in space and/or time
- Process parameters θ_P.
- Virtually all interesting scientific questions can be made with inference about **z**

2.3.5 Parameter (Prior) Model (BMHs only)

$$[\theta_D][\theta_P]$$

- Probability distributions define "reasonable" ranges for parameters.
- Parameter models are useful for a variety of problems:
  - Choosing important variables.
  - Preventing over-fitting (regularization).
  - "Pooling" estimates across categories.

2.3.6 Posterior Distribution

$$[\mathbf{z}, \theta_D, \theta_P|\mathbf{y}] \propto [\mathbf{y}|\theta_D, \mathbf{z}][\mathbf{z}|\theta_P][\theta_D][\theta_P]$$

- Probability distribution over all unknowns in the model.
- Inference is made using the posterior distribution.
- Because the posterior distribution is a probability distribution (BHMs), uncertainty is easy to calculate. This is not true for EHMs.

2.3.7 Scientifically Motivated Statistical Modeling

- Criticize the model
- Does the model fit the data well?
- Do the predictions make sense?
- Are there subsets of the data that don't fit the model well?
- Make inference using the model.
- If the model fits the data, use the model fit for prediction or inference.

Chapter 3

Day 2

3.1 Spatial Data

All data occur at some location is space and time. For know we focus on spatial analyses and will later extend this to spatio-temporal analyses. Let $\mathcal{D}$ represent the spatial domain and let $\mathbf{s}$ be a spatial location. In general, we will let $\mathcal{A} \subset \mathcal{D}$ be a subdomain of the spatial region of $\mathbf{D}$.

Insert Diagram from class here

3.2 Types of spatial data

There are three primary types of spatial data that we are going to consider

- Geostatistical data
  - Occur everywhere
  - continuous support
  - examples: temperature, precipitation
- Areal data
  - Occur only over discrete areas
  - can be thought of as an integral of a continuous process over a subdomain $\mathcal{A} \in \mathcal{D}$
  - examples: cases of a disease by counties, votes in an election by congressional district
- Point process data
  - The count and location of the data are random

    - examples: tornados, lightning strikes

```
library(tidyverse)
library(here)
```

- Many different file types for spatial data
    - Typically data are in "flat files" like comma-seperated value (CSV) files

```
read.csv(here("path", "to", "file.csv"))
```

    - "shapefiles" which can be read using *rgdal* or *maptools* packages

```
library(rgdal)
library(maptools)
```

    - "NetCDF" files cane be read using *ncdf4* or *RNetCDF*

```
library(ncdf4)
library(RNetCDF)
```

3.3 Textbook package

To install the data from the textbook, go to https://spacetimewithr.org/ and follow the link to the code.

```
# install.packages("devtools")
library(devtools)
install_github("andrewzm/STRbook")
```

Note that this package is relatively large because it contains a decent amount of spatial data.

```
library(STRbook)
```

3.3.1 In Class Activity:

From Lab 2.1 on the textbook site

```r
## Wikle, C. K., Zammit-Mangion, A., and Cressie, N. (2019),
## Spatio-Temporal Statistics with R, Boca Raton, FL: Chapman & Hall/CRC
## Copyright (c) 2019 Wikle, Zammit-Mangion, Cressie
##
## This program is free software; you can redistribute it and/or
## modify it under the terms of the GNU General Public License
## as published by the Free Software Foundation; either version 2
## of the License, or (at your option) any later version.
##
## This program is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
## GNU General Public License for more details.

library("dplyr")
library("tidyr")
library("STRbook")
```

```r
## ------------------------------------------------------------------------
locs <- read.table(system.file("extdata", "Stationinfo.dat",
                               package = "STRbook"),
                   col.names = c("id", "lat", "lon"))
Times <- read.table(system.file("extdata", "Times_1990.dat",
                                package = "STRbook"),
                    col.names = c("julian", "year", "month", "day"))
Tmax <- read.table(system.file("extdata", "Tmax_1990.dat",
                               package = "STRbook"))

## ------------------------------------------------------------------------
names(Tmax) <- locs$id

## ------------------------------------------------------------------------
Tmax <- cbind(Times, Tmax)
head(names(Tmax), 10)
```

```
##  [1] "julian" "year"   "month"  "day"    "3804"   "3809"   "3810"   "3811"
##  [9] "3812"   "3813"
```

```r
## ------------------------------------------------------------------------
Tmax_long <- gather(Tmax, id, z, -julian, -year, -month, -day)
head(Tmax_long)
```

```
##   julian year month day   id  z
## 1 726834 1990     1   1 3804 35
```

```
## 2 726835 1990     1   2 3804 42
## 3 726836 1990     1   3 3804 49
## 4 726837 1990     1   4 3804 59
## 5 726838 1990     1   5 3804 41
## 6 726839 1990     1   6 3804 45
```

```
## ------------------------------------------------------------------------
Tmax_long$id <- as.integer(Tmax_long$id)

## -----------------------------------------------------------
nrow(Tmax_long)
```

```
## [1] 479208
```

```
Tmax_long <- filter(Tmax_long, !(z <= -9998))
nrow(Tmax_long)
```

```
## [1] 196253
```

```
## ------------------------------------------------------------------------
Tmax_long <- mutate(Tmax_long, proc = "Tmax")
head(Tmax_long)
```

```
##   julian year month day   id  z proc
## 1 726834 1990     1   1 3804 35 Tmax
## 2 726835 1990     1   2 3804 42 Tmax
## 3 726836 1990     1   3 3804 49 Tmax
## 4 726837 1990     1   4 3804 59 Tmax
## 5 726838 1990     1   5 3804 41 Tmax
## 6 726839 1990     1   6 3804 45 Tmax
```

```
## ------------------------------------------------------------------------
data(Tmin_long, package = "STRbook")
data(TDP_long, package = "STRbook")
data(Precip_long, package = "STRbook")

## ------------------------------------------------------------------------
NOAA_df_1990 <- rbind(Tmax_long, Tmin_long, TDP_long, Precip_long)

## ------------------------------------------------------------------------
summ <- group_by(NOAA_df_1990, year, proc) %>%  # groupings
        summarise(mean_proc = mean(z))          # operation
```

```r
## ----------------------------------------------------------------------------
NOAA_precip <- filter(NOAA_df_1990, proc == "Precip" & month == 6)
summ <- group_by(NOAA_precip, year, id) %>%
        summarise(days_no_precip = sum(z == 0))
head(summ)
```

```
## # A tibble: 6 x 3
## # Groups:   year [1]
##    year    id days_no_precip
##   <int> <int>          <int>
## 1  1990  3804             19
## 2  1990  3810             26
## 3  1990  3811             21
## 4  1990  3812             24
## 5  1990  3813             25
## 6  1990  3816             23
```

```r
## ----------------------------------------------------------------------------
median(summ$days_no_precip)
```

```
## [1] 20
```

```r
## ------------------------------------------------------------
grps <- group_by(NOAA_precip, year, id)
summ <- summarise(grps, days_no_precip = sum(z == 0))

## ----------------------------------------------------------------------------
NOAA_df_sorted <- arrange(NOAA_df_1990, julian, id)

## ----------------------------------------------------------------------------
df1 <- select(NOAA_df_1990, julian, z)
df2 <- select(NOAA_df_1990, -julian)

## ----------------------------------------------------------------------------
NOAA_df_1990 <- left_join(NOAA_df_1990, locs, by = "id")

## ----------------------------------------------------------------------------
Tmax_long_sel <- select(Tmax_long, julian, id, z)
Tmax_wide <- spread(Tmax_long_sel, id, z)
dim(Tmax_wide)
```

```
## [1] 1461  138
```

```
## ------------------------------------------------------------------------
M <- select(Tmax_wide, -julian) %>% as.matrix()

## -----------------------------------------------------------
library("sp")
library("spacetime")

## ------------------------------------------------------------------------
NOAA_df_1990$date <- with(NOAA_df_1990,
                       paste(year, month, day, sep = "-"))
head(NOAA_df_1990$date, 4)   # show first four elements
```

```
## [1] "1990-1-1" "1990-1-2" "1990-1-3" "1990-1-4"
```

```
## ------------------------------------------------------------------------
NOAA_df_1990$date <- as.Date(NOAA_df_1990$date)
class(NOAA_df_1990$date)
```

```
## [1] "Date"
```

```
## ------------------------------------------------------------------------
Tmax_long2 <- filter(NOAA_df_1990, proc == "Tmax")
STObj <- stConstruct(x = Tmax_long2,           # data set
                    space = c("lon", "lat"),  # spatial fields
                    time = "date")            # time field
class(STObj)
```

```
## [1] "STIDF"
## attr(,"package")
## [1] "spacetime"
```

```
## ------------------------------------------------------------------------
spat_part <- SpatialPoints(coords = Tmax_long2[, c("lon", "lat")])
temp_part <- Tmax_long2$date
STObj2 <- STIDF(sp = spat_part,
                time = temp_part,
                data = select(Tmax_long2, -date, -lon, -lat))
class(STObj2)
```

```
## [1] "STIDF"
## attr(,"package")
## [1] "spacetime"
```

```
## ----------------------------------------------------------------------
spat_part <- SpatialPoints(coords = locs[, c("lon", "lat")])
temp_part <- with(Times,
                  paste(year, month, day, sep = "-"))
temp_part <- as.Date(temp_part)

## ----------------------------------------------------------------------
Tmax_long3 <- gather(Tmax, id, z, -julian, -year, -month, -day)

## ----------------------------------------------------------------------
Tmax_long3$id <- as.integer(Tmax_long3$id)
Tmax_long3 <- arrange(Tmax_long3,julian,id)

## ----------------------------------------------------------------------
all(unique(Tmax_long3$id) == locs$id)
```

```
## [1] TRUE
```

```
## ----------------------------------------------------------------------
STObj3 <- STFDF(sp = spat_part,
                time = temp_part,
                data = Tmax_long3)
class(STObj3)
```

```
## [1] "STFDF"
## attr(,"package")
## [1] "spacetime"
```

```
## ----------------------------------------------------------------------
proj4string(STObj3) <- CRS("+proj=longlat +ellps=WGS84")

## ----------------------------------------------------------------------
STObj3$z[STObj3$z == -9999] <- NA
```

Chapter 4

Day 3

Chapter 5

Applications

Some *significant* applications are demonstrated in this chapter.

5.1 Example one

5.2 Example two

Chapter 6

Final Words

We have finished a nice book.

Bibliography

Berliner, L. M. (1996). Hierarchical Bayesian time series models. In *Maximum Entropy and Bayesian Methods*, pages 15–22. Springer.