北京航空航天大學
经济管理学院
BEIHANG UNIVERSITY
SCHOOL OF ECONOMICS AND MANAGEMENT

Generalized Linear Models

Lecture 1: Introduction

Outline

1 Unit information

2 How to install and use R

3 R basics

4 R coding style

5 Some nice R tips

Contact details

Lecturer

Yanfei Kang

- Email: `yanfeikang@buaa.edu.cn`
- `http://yanfei.site`

Tutor

Bohan Zhang

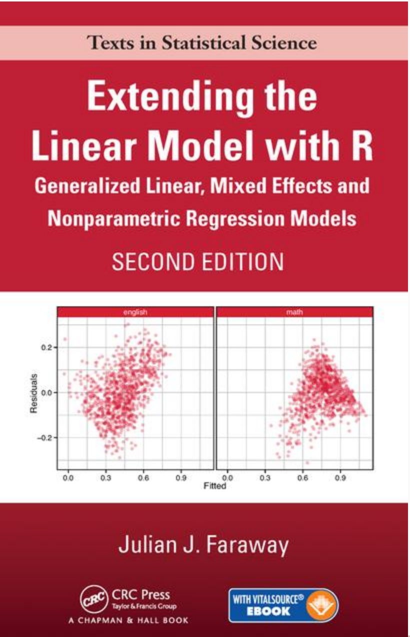
Unit objectives

1. provide an understanding of statistical models for handling common data analysis problems
2. develop skills for fitting, interpreting and assessing statistical models
3. develop computer skills for exploring and modelling different kinds of data.

Teaching and learning approach

- Two 50 min lectures (Thursdays 1:30pm - 15:20pm)
- At least 3 hours' learning after class

Key reference



Texts in Statistical Science

Extending the Linear Model with R

Generalized Linear, Mixed Effects and Nonparametric Regression Models

SECOND EDITION

Julian J. Faraway

CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK

WITH VITALSOURCE® EBOOK

pdf on opencourse

Note

- Thanks to Prof. Rob Hyndman for sharing his slides.
- The textbook and slides are not allowed to be put online.

Outline (tentative)

Week(s)	Topic	
1,2	GLM	Review of R and linear models
3	GLM	Binary responses
4,5	GLM	Binomial and proportional responses
6,7	GLM	Regression with count responses
8	GLM	Multinomial Data
9,10	GLM	Generalized linear model theory
11,12	GLMM	Random effects
13,14	GLMM	Mixed effectss for non-Gaussian responses
15	GAM	Extras
16	-	Revision

Assessment

Task	Value
Attendence	10%
Assignments	30%
Final exam	60%

Outline

1 Unit information

2 How to install and use R

3 R basics

4 R coding style

5 Some nice R tips

Getting and Installing R

- Totally free!
- Download R on its official website.

R User Interface

- Direct **R**
- Rstudio
  - One of the most popular ways to run R.
  - Free, open-source integrated development environment (IDE) for R.
  - Many additional fantastic features.
- Command line in Linux and Unix.

Editor

- What editor do you usually use?
- Use a good text editor such as vim, sublime text, text wrangler, notepad, etc
- With syntax highlighting, otherwise, it's hard to detect errors
- Or use an Integrated Development Environment (IDE) like RStudio

Use an IDE: Rstudio

- Syntax highlighting
- Able to evaluate R code
    - by line
    - by selection
    - entire file
- Command auto-completion

Outline

1 Unit information

2 How to install and use R

3 R basics

4 R coding style

5 Some nice R tips

R packages

- Standard R comes with some standard packages installed for basic data management, analysis, and graphical tools.
- More than 10,000 packages available on CRAN! See `http://cran.r-project.org`.
- `install.packages('formatR')` to install an package called ‘formatR’.
- `library(formatR)` before using the package.

Basic Operations

```
# simple maths
1 + 2 + 3
1 + 2 * 3

# assign a value to a variable
x <- 1
y <- 2
z <- c(x,y)
z
```

Basic Operations

```
# function examples
exp(1)
cos(3.141593)
log2(1)
```

Vectors

- Numerical vectors
- Logical vectors
- Character vectors
- Length of a vector
- Vector calculations
- Extract some elements of a vector

Vectors

```r
# vectors
c(0, 1, 1, 2, 3, 5, 8)
1:10
seq(1, 9, 2)
rep(1, 10)
length(rep(1, 10))

# character vectors
c("Hello world", "Hello R interpreter")
```

Vectors

```r
# vector calculation
c(1, 2, 3, 4) + c(10, 20, 30, 40)
c(1, 2, 3, 4) + 1
c(1, 2, 3, 4) * 2
```

Vectors

```r
# you can refer to elements by location
# in a vector
b <- c(1,2,3,4,5,6,7,8,9,10,11,12)
length(b)
b
b[7]
b[1:6]
b[c(1,6,11)]
b > 5
b[b > 5]
```

Matrix

- Create a matrix: `matrix()`
- Dimension of a matrix: `dim()`
- Transpose of a matrix: `t()`
- Extract elements from a matrix.
- Combine two or more matrices: `rbind()`, `cbind()`

Matrix - Example

```r
# create a matrix
m <- matrix(c(1:6), 2, 3)
n <- matrix(c(8:13), 2, 3)
dim(m)
t(m)
m[1, 2]
m[1, ]
cbind(m, n)
rbind(m, n)
```

List

- Special data structure that matrix could not handle.
  - Data length are not the same.
  - Data type are not the same.
- Create a list: `list()`
- Extract elements of a list: `[[]]` or $

```
l <- list(a = c(1, 2), b = 'apple')
```

Data frame

- `data.frame()`: tightly coupled collections of variables which share many of the properties of matrices and of lists, used as the fundamental data structure by most of R's modeling software.
- In most cases, the operation with a data frame is similar to matrix operation.

```r
L3 <- LETTERS[1:3]
fac <- sample(L3, 10, replace = TRUE)
d <- data.frame(x = 1, y = 1:10, fac = fac)
```

Functions

- Create a function

```
f <- function(x, y) {
  z <- c(x + 1, y + 1)
  return(z)
}
f(1, 2)
```

- Load the function: `source()`
- Execute your function

The if condition

Syntax

```
if (condition){
  do something
} else {
  do something
}
```

The if condition - Example

```r
x <- 0
if (x > 1) {
  print('x is larger than 1')
} else {
  print('x is not larger than 1')
}
```

Loops

```r
x <- 1:10
for(i in x) {
  print(i^2)
}
```

Your turn (1)

1. Write a function `MySummary()` where the input argument is x can be any vector and the output is a list that contains the basic summary (mean, variance, length, max and minimum values) of the vector you have supplied to the function.
2. Test your function with some vectors (that you make up by yourself).

Outline

1 Unit information

2 How to install and use R

3 R basics

4 R coding style

5 Some nice R tips

File names

- File names should end in .R and, of course, be *meaningful*.
- GOOD: `predict_ad_revenue.R`
- BAD: `foo.R`

Choose the names carefully

- The preferred form for variable names is all lower case letters and words separated with dots (variable.name), but variableName is also accepted. Generally, variable names should be nouns.
  - GOOD: avg.clicks
  - OK: avgClicks
  - BAD: avg_Clicks
- Function names have initial capital letters and no dots. Function names are mostly verbs.
  - GOOD: CalculateAvgClicks
  - BAD: calculate_avg_clicks , calculateAvgClicks
- Choose a consistent naming style

What we should not do

- Don’t use underscores (_) or hyphens (-).
- Avoid using names of existing functions and variables like `mean`, `median` etc.
- Avoid using meaningless names like a, b, c, . . . , aa, bb, cc, . . .

White Spaces

- around operators (=, +, -, <-, etc)
- put a space after a comma, and never before

```
x <- c(1:10)
x.average<-mean(x,na.rm=TRUE)
```

⇒

```
x.average <- mean(x, na.rm = TRUE)
```

- split long lines at meaningful places

White Spaces

Don’t be afraid of splitting one long line into individual pieces!

```r
n <- matrix(sample(1:100, 9),
           nrow = 3,
           ncol = 3,
           byrow = TRUE)
```

Curly braces

- An opening curly brace should never go on its own line and should always be followed by a new line.
- A closing curly brace should always go on its own line, unless it's followed by else.
- Always begin the body of a block on a new line.
- Always indent the code inside curly braces.

Curly braces

```
if (y < 0) {print("y is negative")}
```

⇒

```
if (y < 0) {
  print("y is negative")
}
```

Indenting

- Use two spaces
- Can help in detecting errors in your code because it can expose lack of symmetry
- Reindenting using RStudio

Indenting

```
if (y < 0) {
print("y is negative")
}
```

⇒

```
if (y < 0) {
  print("y is negative")
}
```

Make your code tidy in a second!

- Reformat and reindent in Rstudio.
- **formatR** package in **R**. You can even make a folder of `.R` files tidy using `tidy.dir()`.

Header, Line spaces and Comments

- Add a Header for your file
- Add lots of comments
- Use blank lines to separate blocks of code and comments to say what the block does. Remember that in a few months, you may not follow your own code any better than a stranger.

Function Documentation

- Functions should contain a comments section immediately below the function definition line.
- These comments should include
  - a one-sentence description of the function
  - a list of the function’s arguments, denoted by Args:, with a description of each (including the data type)
  - a description of the return value, denoted by Returns:.
  - The comments should be descriptive enough that a caller can use the function without reading any of the function’s code.

Outline

1 Unit information

2 How to install and use R

3 R basics

4 R coding style

5 Some nice R tips

How to find the right function

- Functions in installed packages

```
library(forecast)
help.search("auto.arima")
??auto.arima
```

- Functions in other CRAN packages

```
library(sos)
findFn("arima")
RSiteSearch("arima")
```

Digging into functions

- Type `?sort` for the usage of the function `sort()`.
- Typing the name of a function gives its definition.
- Type `forecast:::estmodel` for hidden functions.
- Download the tar.gz file from CRAN if you want to see any underlying **C** or **Fortran** code.

Organize your R projects

- Every paper, book or scientific report is a ‘project’.
- Every project has its own folder and R workspace.
- Every project is entirely scripted. That is, all analysis, graphs and tables must be able to be generated by running one script.
    - This script sources all other R files in the correct order and yields all the required results. This script could be in `main.R` or `main.Rmd`.
    - `functions.R` contains all non-packaged functions used in the project.
    - each function can not be too long.

Look at other people’s codes

- https://github.com/hadley
- https://github.com/yihui
- https://github.com/karthik
- https://github.com/kbroman
- https://github.com/cboettig
- https://github.com/garrettgman

Getting help

- For programming questions: StackOverflow.com
- For statistical questions: CrossValidated.com

Keep up-to-date

- RStudio blog: blog.rstudio.org
- R-bloggers: www.r-bloggers.com
- It takes time to develop your own style. Once it is developed, it is really hard to be changed. So please be careful at the beginning.

Your turn (2)

- Use `tidy_dir()` to make your code tidy.

References

- Official introduction to R
- Google R style guide
- Rob’s tips