Estadística Computacional en R

Publicación 0.1

Saní

27 de June de 2017

Índice general

El uso de R tiene dos puntos de vista, uno de computación dirigido hacia el uso de R como lenguaje de programación, y otro estadístico, según el cual R sirve de herramienta para modelado estadístico.

Ambos enfoques son fundamentales para el correcto uso del entorno R, este taller pretende ofrecer un balance equilibrado entre R como lenguaje de programación y como herramienta de modelado estadístico.

Contenidos:

CAPÍTULO 1

Introducción a R

Conceptos básicos

R es un dialecto del lenguaje S. Este último fue desarrollado en 1976 por John Chambers y sus colegas en Bell Labs, como un entorno para análisis estadístico interno, y se implementó inicialmente como librerías de Fortran.

Los fundamentos básicos de S no han cambiado desde 1998, año en el cual, como lenguaje, ganó el Premio de la Association for Computing Machinerys Software System.

Para ese mismo año, y habiéndose hecho público R como lenguaje en 1993, se forma el Core Group R a fin de controlar el código fuente de R, integrado por varias personas asociadas con S-PLUS.

La versión más reciente de R es la 3.0.2 publicada en Diciembre del 2013.

El lenguaje R tiene como características básicas:

- R es un Lenguaje y Entorno de programación dinámico para análisis estadístico y generación de gráficos.
- Puede ser ejecutado en casi cualquier plataforma estándar de sistema operativo (aún PlayStation 3!!)
- Puede utilizarse en modo interactivo o script.
- Es Software Libre, con una amplia comunidad de desarrolladores, investigadores y usuarios.

Instalación de R y RStudio

Windows

Descargar R desde http://cran.r-project.org/bin/windows/base/

Seleccionar el enlace donde dice: Download R X.X.X for Windows. Donde X.X.X es la versión de R más reciente. La actual es 3.1.1

http://cran.r-project.org/bin/windows/base/R-3.1.1-win.exe

Ejecutar el instalador de R

Descargar la versión más reciente de RStudio desde la dirección: http://rstudio.org/download/desktop

La versión más reciente para el 9 de octubre es 0.98.162

http://download1.rstudio.org/RStudio-0.98.1062.exe

Ejecutar el instalador de RStudio

Linux

Debian estable

Añadir el repositorio (a `/etc/apt/sources.list`)

deb http://cran.rstudio.com/bin/linux/debian wheezy-cran3/

Añadir claves backport de Debian

```
$ apt-key adv --keyserver keys.gnupg.net --recv-key 381BA480
```

Finalmente instalar

```
$ apt-get update
$ apt-get install r-base r-base-dev
```

Ubuntu o Mint

Para instalar la versión mas actualizada de R añadir el repositorio: (a `/etc/apt/sources.list` o mediante "Orígenes de software")

deb http://cran.rstudio.com/bin/linux/ubuntu trusty/

O la versión de Ubuntu que corresponda

Añadir claves para Ubuntu archive

```
$ sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys E084DAB9
```

E instalar finalmente

```
$ sudo apt-get update
$ sudo apt-get install r-base r-base-dev
```

Dependiendo del caso, descargar uno de los siguientes:

http://download1.rstudio.org/rstudio-0.98.1062-i386.deb

http://download1.rstudio.org/rstudio-0.98.1062-amd64.deb

Según la arquitectura sea de 32 o 64 bits, e instalar utilizando

sudo dpkg -i rstudio-0.98.1062-???.deb

O con gdebi-gtk

Interfaz gráfica de RStudio

Consta de 4 paneles configurables:

- Editor
- Consola
- Entorno
- Ayuda

El *Editor* por defecto está oculto y se activa cuando se crea un nuevo archivo que puede ser código R, R Markdown, un archivo de texto u otros.

La *Consola* es el terminal interactivo de R donde se envían los comandos para ser ejecutados.

En el *Entorno* se encuentra tanto la información de los objetos de R en la memoria con funcionalidad para importar datos, y existen pestañas para manejo del historial y del sistema de control de versiones.

Finalmente en el panel *Ayuda*, aparte de acceso integrado a la ayuda de R y los paquetes activos, se ofrece un explorador de archivos, el gestor de paquetes y el gestor de gráficos.

Comparación con EXCEL, SAS, SPSS, Stata

En la actualidad, el SAS Institute e IBM SPSS, y otras compañías trabajan para extender sus sistemas usando R. Al igual que hay complementos de EXCEL para integrarlo con las capacidades de R. En buena medida, R es una alternativa para Stata como lenguaje de programación y modelado estadístico.

Entre los beneficios directos de usar R se tiene:

- Acceso a una gran variedad de métodos de análisis.
- Acceso temprano a nuevos métodos.
- Muchos paquetes computacionales permiten ejecutar programas en R. Puede realizar el manejo de datos en el sistema de su preferencia y después lanzar los análisis usando R.
- La rápida difusión de R como lenguaje estadístico de referencia.
- La gran calidad y flexibilidad de los gráficos generados por R.
- La capacidad para analizar datos en una gran cantidad de formatos.
- Cuenta con capacidades de orientación a objetos.
- Facilidades para implantar sus propios métodos de análisis.
- Posibilidad de revisar en detalle como están implantados los métodos de análisis.
- Los métodos propios están desarrollados en el mismo lenguaje que la mayoría de los métodos del sistema.
- Provee capacidades de álgebra matricial similares a Matlab.
- Se ejecuta en prácticamente cualquier sistema operativo, ya sea Windows, Mac, Linux, o Unix.
- R es libre.

El soporte que se espera de R se ofrece mediante las listas de discusión vía email, y foros como stackoverflow. Estos espacios tienen dinámicas propias y demandan de los usuarios destrezas para plantear preguntas y entender las respuestas, y desarrollar criterios para distinguir entre distintas opciones.

En sistemas como SAS y SPSS se distinguen 5 componentes tales como:

- Gestión de datos, que permiten leer, transformar y organizar los datos.
- Procedimientos estadísticos y gráficos.
- Sistemas de extracción de salidas que permiten extraer salidas de unos procedimientos para utilizarlos como entradas en otros: SAS Output Delivery System (ODS) y SPSS Output Management System (OMS).
- Un lenguaje de macros que facilita el uso de los anteriores componentes.
- Un lenguaje matricial para implantar nuevos métodos: SAS/IML y SPSS Matrix.

La diferencia es que R realiza estas funciones de una forma tal que las integra a todas. En particular facilita la gestión de salidas, una característica poco utilizada por los usuarios de los otros sistemas.

Instalación de paquetes

Cuando se descarga R del Comprehensive Archive Network (CRAN), se obtiene el sistema *base* que ofrece la funcionalidad básica del lenguaje R.

Se encuentra disponible una gran cantidad de paquetes que extienden la funcionalidad básica de R. Estos paquetes son desarrollados y publicados por la comunidad de usuarios de R.

La principal ubicación para obtener paquetes de R es CRAN. Se dispone de muchos paquetes para aplicaciones de bioinformática,del Proyecto Bioconductor.

Se puede obtener información de los paquetes disponibles en los repositorios configurados mediante la función `available_packages()`.

En la actualidad se dispone de caso 6 mil paquetes en CRAN que cubren una gran diversidad de temas. Una buena orientación inicial se puede encontrar en el enlace Task Views (Vista por tareas) de la página principal de CRAN, que agrupa los paquetes de R por área de aplicación.

Instalar paquetes de R

Los paquetes se pueden instalar con la función de R `install.packages()`. Para instalar un paquete se pasa su nombre como primer argumento. El código a continuación instala el paquete **knitr** desde CRAN.

```
install.packages("knitr")
```

Este comando descarga el paquete **knitr** desde CRAN y lo instala en su computadora. De igual manera, se descargan e instalan todas sus dependencias.

Si se introduce como parámetro un vector tipo carácter se pueden instalar varios paquetes en simultáneo.

```
install.packages(c("knitr", "dplyr", "ggplot2"))
```

Instalación desde RStudio

Desde la interfaz de RStudio se pueden instalar desde el menu `Tools>Install Packages...`, o bien desde la pestaña *Packages* del panel *Ayuda*.

En ambos casos se despliega un diálogo de instalación de paquetes que permite indicar el nombre del paquete en una caja de texto. Si el paquete se encuentra en el repositorio después de escribir unas pocas letras del nombre debería autocompletarse.

Instalación desde Bioconductor

Para instalar un paquete desde Bioconductor se deben instalar las funciones básicas de este repositorio mediante las instrucciones:

```
source("http://bioconductor.org/biocLite.R")
biocLite()
```

El primer comando carga funciones de R desde el script `biocLite.R`, el segundo ejecuta una función contenida en este.

A partir de este momento, Bioconductor queda configurado como repositorio y es posible instalar paquetes del mismo utilizando la función `install_packages()`.

Cargar paquetes

Para que las funcionalidades de los paquetes estén disponibles en la sesión de R tienen que ser *cargados* en la memoria. Esto se realiza mediante la función `library()`. Por ejemplo, para cargar el paquete `reshape`:

```
library(reshape)
```

Nótese que a diferencia de la instalación, en este caso no son necesarias las comillas para introducir el nombre del paquete.

Este comando carga tanto el paquete indicado como todas sus dependencias.

Al cargar un paquete, todos los objetos contenidos en el mismo quedan disponibles en el entorno, y su documentación es incluida en el sistema de ayuda.

```
> library("rstudio", lib.loc="~/R/x86_64-pc-linux-gnu-library/3.1")
> search()
 [1] ".GlobalEnv"        "package:rstudio"   "tools:rstudio"
 [4] "package:stats"     "package:graphics"  "package:grDevices"
 [7] "package:utils"     "package:datasets"  "package:methods"
[10] "Autoloads"         "package:base"
```

Desde la interfaz de RStudio en la pestaña *Packages* del Panel Ayuda, se pueden cargar paquetes haciendo clic en la casilla de verificación que se encuentra a la izquierda del nombre correspondiente.

CAPÍTULO 2

Elementos de Programación con R I

Tipos de datos y objetos básicos

Objetos

R cuenta con 5 clases de objetos "atómicos":

- character: Cadenas de carácter.
- numeric: Valores numéricos de punto flotante.
- integer: Valores numéricos enteros.
- complex: Valores numéricos complejos. Es decir con parte real e imaginaria.
- logical: Valores de verdadero o falso.

El tipo mas básico de objeto es un *vector*, el cual solamente puede contener objetos de la misma clase.

Se pueden crear objetos vacíos con la función `vector()`.

Valores numéricos

En R los números se tratan por defecto como objetos numéricos (es decir, valores de punto flotante de doble precisión). Si se desea establecer un valor numérico como entero se le debe agregar el sufijo `L`.

Por ejemplo: `1` denota a un objeto numérico; mediante `1L` el usuario establece explícitamente un entero.

El valor numérico especial `Inf` denota simbólicamente al infinito. Por ejemplo:

```
> 1 / 0
[1] Inf
```

El valor `NaN` (Not a number) representa un valor indefinido.

```
> 0 / 0
[1] NaN
```

Atributos

Los objetos de R pueden tener atributos descriptivos, tales como:

- names, dimnames: nombres de las columnas y de los ejes.

- dimensions: número de dimensiones de un arreglo o matriz.
- class: clase de un objeto.
- length: número de elementos que contiene un objeto.
- otros metadatos/atributos definidos por el usuario o la usuaria.

Se puede acceder a los atributos de un objeto mediante la función `attributes()`.

Asignación

En el terminal/consola de R se escriben expresiones. El símbolo <− es el operador de asignación.

```
> x <- 1
> print(x)
[1] 1
> x
[1] 1
> msg <- "hola"
```

La sintaxis del lenguaje determina si una expresión es completa o no.

```
> x <- # Expresión incompleta
+
```

El operador de asignación requiere del valor a asignar. El símbolo # indica un comentario, es decir todo lo que se encuentra a la derecha del símbolo es ignorado por R. Es el mecanismo empleado para añadir textos explicativos.

Evaluación

Cuando se introduce una sentencia completa en la consola, esta es evaluada. En la consola interactiva, el contenido de los objetos de imprime directamente.

```
> N <- 6
> N           # impresión automática.
[1] 6
> print(N)    # impresión explícita.
[1] 6
```

El `[1] 6` indica que es el primer elemento de `N`.

Creación de vectores

Se utiliza la función `c()` "combinar", para crear objetos de vectores.

```
> x <- c(3.3, 8) # numérico
> x <- c(FALSE, TRUE) # lógico
> x <- c("x", "y", "z") # carácter
> x <- 5:10 # entero (secuencia)
> x <- c(2+3i, 4-5i) # complejo
```

Se puede utilizar la función `vector()` o una función de cuyo nombre sea clase del vector a crear:

```
> vector("character", 5)
[1] "" "" "" "" ""
```

```
> character(5)
[1] "" "" "" "" ""
```

Mezcla de objetos

Si se mezclan valores de distintas clases en un vector, estos se "coercen". Es decir, se cambia la clase de los valores para obligar que todos sean de la misma clase.

```
> x <- c("hola", 4) # character
> x
[1] "hola" "4"
> y <- c(TRUE, 5) # numerical
> y
[1] 1 5
```

Funciones del tipo `as.numeric()` o `as.logical` se pueden utilizar para realizar una coerción explícita del vector. Cuando los valores no pueden ser coercidos al tipo indicado devuelve valores especiales del tipo `NA` (No disponible, "Not Available").

```
> z <- as.numeric(x)
Mensajes de aviso perdidos
NAs introducidos por coerción
> z
[1] NA  4
```

Matrices

Son arreglos con un atributo de dimensión de dos valores enteros que hacen referencia al número de filas y columnas.

```
> A <- matrix(1:12, nrow = 3, ncol = 4)
> A
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> dim(A)
[1] 3 4
```

Del ejemplo anterior, se puede observar que los valores se ordenan por defecto en la matriz por columnas. A menos que se establezca lo contrario mediante el argumento `byrow = TRUE`.

Se puede crear una matriz añadiendo un atributo de dimensión a un vector.

```
> w <- 1:10
> dim(w) <- c(2,5)
> w
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
```

También se pueden crear matrices uniendo vectores como filas (`rbind()`, row binding) o como columnas (`cbind()`, column binding).

```
> x <- 3:5
> y <- 10:12
> rbind(x, y)
```

```
  [,1] [,2] [,3]
x    3    4    5
y   10   11   12
```

Listas

Las listas son objetos que contienen elementos de distintas clases. Son tipos de datos muy importantes en R, ya que son la base de datos estructurados.

```
> l <- list("a", 5, TRUE, 1 + 4i)
> l
[[1]]
[1] "a"

[[2]]
[1] 5

[[3]]
[1] TRUE

[[4]]
[1] 1+4i
```

Factores

Los factores se utilizan para representar datos categóricos ordenados o desordenados. Se pueden concebir como valores enteros asociados a etiquetas. A diferencia de los enteros los factores son descriptivos.

Son especialmente importantes para especificar modelos estadísticos, ya sea en modelos de regresión, o para la generación de gráficos.

Los distintos valores de un factor se conocen como "niveles" (levels).

```
> t <- factor(c("f", "e", "d"))
> t
[1] f e d
Levels: d e f
> levels(t) <- c("c", "d", "e", "f")
> t
[1] e d c
Levels: c d e f
```

La función `levels()` puede utilizarse para especificar el orden de los niveles, y los posibles valores que puede tener un factor.

```
> t[3] <- "o"
Mensajes de aviso perdidos
In ``[<-.factor``(``*tmp*``, 3, value = "o") :
  invalid factor level, NA generated
```

Valores Faltantes

Son aquellos denotados por `NA` o `NaN` para los que se generan por operaciones matemáticas indefinidas.

Para verificar si los objetos son `NA` o `NaN` se pueden utilizar respectivamente las funciones `is.na()` o `is.nan()`.

Los valores NA pertenecen a una clase, por esta razón se tienen valores NA enteros, NA lógicos, etc.

Un valor NaN es NA pero no a la inversa.

```
> s <- c(NA, 5, 0 / 0)
> s
[1]  NA   5 NaN
> is.na(s)
[1]  TRUE FALSE  TRUE
> is.nan(s)
[1] FALSE FALSE  TRUE
```

Data Frames

La traducción literal sería "marcos de datos. Representan datos tabulares.

- Son un tipo especial de lista en la que todos sus elementos tienen la misma longitud.
- Cada elemento puede concebirse como una columna. Y cada fila denota a los objetos que están en la misma posición en todas las columnas.
- A diferencia de las matrices pueden tener distintas clases de objetos en cada columna.

Los objetos de R tienen nombres `names()` que se asocian a cada elemento.

```
> f <- 1:5
> f
[1] 1 2 3 4 5
> names(f) <- c("a", "b", "c", "d", "e")
> f
a b c d e
1 2 3 4 5
```

Los data frames tienen el atributo especial `row.names()` que permite asociar nombres a las filas.

```
> data.frame(label=c("a", "b", "c"),
+ value= 1:3,
+ row.names = c("uno", "dos", "tres"))
     label value
uno      a     1
dos      b     2
tres     c     3
```

Del mismo modo, se pueden asociar nombres a las filas y a las columnas de una matriz con las funciones `rownames()` y `colnames()` respectivamente. O de forma simultánea con `dimnames()`.

```
> f <- matrix(1:8, nrow = 2, ncol = 4,
+ dimnames = list(c("uno", "dos"),1:4))
> f
    1 2 3 4
uno 1 3 5 7
dos 2 4 6 8
> colnames(f) <- c("I", "II", "III", "IV")
> f
    I II III IV
uno 1  3   5  7
dos 2  4   6  8
```

Lectura y escritura de datos

Funciones básicas

Hay unas cuantas funciones básicas para introducir datos a R:

- `read.table()`, `read.csv()`, para leer datos de forma tabular desde archivos de texto.
- `readLines()`, para leer información de archivos de texto como un vector de clase carácter.
- `source()`, para ejecutar código R. El contrario de `dump()`.
- `dget()`, carga un objeto de R guardado como una representación en texto almacenado con `dput()`.
- `load()`, para cargar *espacios de trabajo* almacenados en formato `.RData`.
- `unserialize()`, para leer objetos de R individuales guardados en formato binario.

Existen las siguientes funciones análogas para escribir datos:

- `write.data()`
- `writeLines()`
- `dump()`
- `dput()`
- `save()`
- `serialize()`

Leer archivos con `read.table()`

La función `read.table()` es una de las mas utilizadas, entre sus argumentos mas importantes tenemos:

- `file`, nombre de un archivo o conexión.
- `header`, valor lógico que indica si el archivo tiene una línea de cabecera.
- `sep`, la cadena de caracteres usada como separador de columnas.
- `colClasses`, un vector clase carácter que indica la clase de cada columna.
- `nrows`, el número de filas de un conjunto de datos.
- `comment.char`, la cadena de caracteres usada como indicador de comentarios.
- `skip`, el número de líneas a saltar al principio.
- `stringsAsFactors`, valor lógico que indica si las columnas de tipo carácter serán codificadas como factores.

Para conjuntos de datos pequeños y medianos, se pueden ejecutar `read.table()` sin ningún otro argumento.

```
data <- read.table("chiguire.txt")
```

La función automáticamente:

- Saltará todas las líneas que empiezan con #.
- Determinará cuantas líneas son y cuanta memoria necesitará.
- Determinará la clase mas conveniente para cada columna.
- `read.csv()` es similar, pero asume que el separador es una coma.

Para conjuntos de datos mas grandes, las siguientes recomendaciones pueden ser útiles:

- Leer la página de ayuda de `help.table()`, que contiene muchas pistas.
- Hacer un cálculo grueso de la memoria requerida, si excede la cantidad de RAM disponible es hora de pensar en otro método.
- Establecer `comment.char = ""` si no hay líneas comentadas en el archivo.
- Especificar el argumento `colClasses`, hará la lectura mucho mas rápida.

```
initial <- read.table("datatable.txt", nrows = 100)
classes <- sapply(initial, class)
tabAll <- read.table("datatable.txt", colClasses = classes)
```

En este caso se utilizan las clases que el propio R estima leyendo las primeras 100 filas para leer el archivo completo.

- Establecer el argumento `nrows`, lo que permite controlar el uso de memoria. Puede usarse para leer un archivo muy grande por partes.

Todo pasa por conocer nuestro sistema, las especificaciones de hardware, la arquitectura del procesador, el sistema operativo utilizado, las aplicaciones en memoria y los usuarios con sesiones abiertas.

Por ejemplo, un data.frame de millón y medio de filas y 120 columnas de datos numéricos (8 bytes por valor) requerirá aproximadamente de:

```
> mem <- 1500000 * 120 * 8 # bytes
> mem <- mem / 2^20 # megabytes
> mem
[1] 1373.291
> mem <- mem / 1024 # gigabytes
> mem
[1] 1.341105
```

Representaciones de texto

Utilizando `dput()` se pueden obtener representaciones de los datos en archivos de texto, que se pueden editar y recuperar.

Estas representaciones preservan los metadatos, y funcionan mejor con sistemas de control de versiones y la "filosofía Unix" en general

Tienen el problema que pueden requerir un gran espacio de almacenamiento.

```
> y <- data.frame(a = c(1, 2), b = c("uno", "dos"))
> dput(y)
structure(list(a = c(1, 2), b = structure(c(2L, 1L),
.Label = c("dos", "uno"), class = "factor")),
.Names = c("a", "b"), row.names = c(NA, -2L), class = "data.frame")
> dput(y, file = "y.R")
> y.nuevo <- dget("y.R")
> y.nuevo
  a   b
1 1 uno
2 2 dos
```

Se utiliza `dump()` para almacenar representaciones de texto de objetos como asignaciones que pueden ser cargados en memoria por lotes con `source()`.

```
> x <- pi
> y <- data.frame(a = c(1, 2), b = c("uno", "dos"))
> dump(c("x", "y"), file = "data.R")
> rm(x, y)
> source("data.R")
> x
[1] 3.141593
> y
  a   b
1 1 uno
2 2 dos
```

Interfaces con el mundo exterior

Se pueden obtener datos utilizando *interfaces* de conexión. La conexiones pueden ser archivos u otras cosas mas exóticas:

- `file()`, abre una conexión a un archivo.
- `gzfile()`, abre una conexión a un archivo comprimido como `gzip`.
- `bzfile()`, abre una conexión a un archivo comprimido como `bzip2`.
- `url()`, abre una conexión a un recurso en Internet, usualmente un sitio web.

Las funciones de conexión en general tienen los argumentos:

- `description`, para `file()` y otras conexiones a archivo es la ruta y nombre del archivo, para `url()` la dirección web.
- `open`, es el tipo de la conexión, `"r"` para solo lectura, `"w"` para iniciar un nuevo archivo y escribir, `"a"` para añadir, y `"rb"`, `"wb"` y `"ab"` los equivalentes en modo binario (Windows).

Las conexiones permiten leer archivos de forma secuencial. Por ejemplo, si tenemos el archivo de texto comprimido `"palabras.txt.gz"`. Se podría leer como sigue:

```
> con <- gzfile("palabras.txt.gz")
> x <- readLines(con)
> x
[1] "hola" "chao" "ula"  "luna"
```

La función `writeLines()` toma como argumento un vector de clase carácter y escribe cada elemento como una línea de una archivo de texto.

Se puede igualmente utilizar una conexión para obtener el código de una página web.

```
> con <- url("http://www.ine.gob.ve", "r")
> y <- readLines(con)
Mensajes de aviso perdidos
...
> head(y)
[1] ""
[2] "<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Transitional//EN\" \"http://www.w3.org/TR/xhtml1/D
[3] "<html xmlns=\"http://www.w3.org/1999/xhtml\">"
[4] "<head>"
[5] ""
[6] "<meta http-equiv=\"Content-Type\" content=\"text/html; charset=utf-8\" />"
```

Estructuras de control

Las estructuras de control básicas de R son:

- `if`, `else`: ejecuta un bloque de código si se satisface una condición.
- `for`: ejecuta un bloque de código un número fijo de veces.
- `while`: ejecuta un bloque de código mientras se cumpla una condición.
- `repeat`: ejecuta un boque de código hasta encontrar un `break`.
- `next`: salta a la siguiente iteración en un `for`, `while` o `repeat`.
- `return`: devuelve el resultado de una función y sale.

La mayoría de las estructuras de control no se utilizan en sesiones interactivas sino en programas de R.

if

Un estructura `if` valida es como sigue:

```
if (x > 3) {
    y <- 10
} else if (x < -3) {
    y <- -10
} else {
    y <- 0
}
```

Esto es: si la condición `x > 3` se satisface se ejecuta el código a continuación encerrado entre llaves. La siguientes sentencias son opcionales, se pueden colocar tantos `else if(<condición>)` "de lo contrario si" como sean necesarios, y de ser necesario una sentencia `else` final.

Debido a la naturaleza funcional de R, la siguiente expresión es equivalente:

```
y <- if (x > 3) {
    10
} else if (x < -3) {
    -10
} else {
    0
}

# los bloques de una sola línea pueden prescindir de las llaves

y <- if (x > 3) 10 else if (x < -3) -10 else 0
```

for

En los bucles `for` a una variable *iteradora* le asignan valores sucesivos de un vector o secuencia.

Los siguientes bucles son equivalentes:

```
x <- c("a", "b", "c", "d")

for (i in 1:4) {
    print(x[i])
```

```
}
for (i in seq_along(x)) {
    print(x[i])
}
for (letter in x) {
    print(letter)
}
for (i in 1:4) print(x[i])
```

Es posible escribir bucles dentro de bucles, esto es, bucles anidados.

Funciones

Las funciones se utilizan para reorganizar el código, ya sea para contener secuencias de expresiones utilizadas de forma reiterada, o para separar el código en componentes mas comprensibles.

Se crean utilizando la directiva `function()` y se almacenan como cualquier otro objeto. Son, de hecho, objetos de la clase *function*.

Tienen la siguiente sintaxis básica:

```
function( arglist )
    expr
    return(value)
```

- `arglist` es una lista de argumentos.
- Si `expr` consta de más de una expresión debe estar encerrado entre llaves.
- la sentencia `return` es opcional, por defecto las funciones en R devuelven el valor de la última expresión.

En R, en virtud de su naturaleza funcional, las funciones son *objetos de primera clase*, lo que implica que:

- Pueden pasarse como argumentos de otras funciones.
- Pueden anidarse, esto es, definir funciones dentro de funciones.
- Devuelven el valor de la última expresión, a menos que hay una indicación explícita con `return()`

Argumentos

Las funciones tienen argumentos con nombre a los que pueden asignarse valores por defecto.

- Los argumentos que aparecen en la definición de la función se denominan *argumentos formales*.
- La función `formals` devuelve una lista con los argumentos formales de una función.

```
> f <- function(a, b) a + b
> formals(f)
$a


$b
```

- Las llamadas a las funciones de R no tienen que utilizar todos los argumentos. Algunos pueden ser quedar *faltantes* y otros tener valores por defecto.

Coincidencia de argumentos

Los argumentos pueden coincidir por posición o por el nombre. Todas las llamadas a continuación de la función `sd` son equivalentes:

```
> midata <- rnorm(100)
> sd(midata)
> sd(x = midata)
> sd(x = midata, na.rm = FALSE)
> sd(na.rm = FALSE, x = midata)
> sd(na.rm = FALSE, midata)
```

Cuando un argumento coincide por nombre, se saca de la lista de argumentos. De manera que los restantes mantienen el mismo orden.

Por ejemplo, en el caso de la función `lm()` que se utiliza para ajustar modelos lineales que tiene los siguientes argumentos:

```
> args(lm)
function (formula, data, subset, weights, na.action, method = "qr",
    model = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE,
    contrasts = NULL, offset, ...)
```

Las dos llamadas siguientes son equivalentes:

```
lm(data = mydata, y ~ x, model = FALSE, 1:100)
lm(y ~ x, mydata, 1:100, model = FALSE)
```

Los argumentos pueden tener una *coincidencia parcial*. Esto es, se pueden hacer coincidir los argumentos por nombre sin tener que escribir el argumento completo siempre que no haya ambigüedad.

Los siguientes llamados son equivalentes:

```
seq.int(0, 1, len = 11)
seq.int(0, 1, length.out = 11)

ls(all = TRUE)
ls(all.names = TRUE)
```

Los argumentos de las funciones de R también emplean *evaluación perezosa*, esto implica que solamente se consideran necesarias en la medida que se utilizan dentro de la función. Por ejemplo, el siguiente código corre sin problemas.

```
f <- function(a, b) {
    a^2
}
f(2)
```

En este caso, como b nunca es utilizado, no genera error. De hecho, se ejecutarían todas las sentencias hasta encontrar una referencia a `b`.

Se puede utilizar `...` para indicar un número de argumentos variable, o el pase de argumentos de forma implícita. Generalmente se utilizan para extender funciones.

```
myplot <- function(x, y, type = "l", ...) {
plot(x, y, type = type, ...)
}
```

Los argumentos formales que aparecen después de `...` deben ser explícitos y no admiten coincidencias parciales.

```
> args(paste)
function (..., sep = " ", collapse = NULL)
> paste("a", "b", sep = ":")
[1] "a:b"
> paste("a", "b", se = ":")
[1] "a b :"
```

Reglas de alcance

¿Cómo determina R que valor asignar a cada símbolo?

Por ejemplo si se redefine una función existente como `mean()`.

```
> mean <- 2 * pi
> mean
[1] 6.283185
> mean(c(4,5,6))
[1] 5
> mean <- function(x) { x + 2 }
> mean(c(4,5,6))
[1] 6 7 8
```

¿Cómo sabe R a cuál objeto llamar en cada caso?

Para R asociar un símbolo a un objeto, busca dentro de una serie de *entornos*. Cuando se ejecuta el comando, R empieza buscando en el entorno global, y después dentro de los espacios de nombre de cada uno de los paquetes de la lista de búsqueda.

Esta lista se determina utilizando la función `search()`

```
> search()
 [1] ".GlobalEnv"        "tools:rstudio"     "package:stats"
 [4] "package:graphics"  "package:grDevices" "package:utils"
 [7] "package:datasets"  "package:methods"   "Autoloads"
[10] "package:base"
```

Enlazando valores a símbolos

- El *entorno global* o espacio de trabajo del usuario siempre es el primer elemento de la lista de búsqueda y el paquete `base` siempre es el último.
- Los objetos se buscan en el orden de la lista de búsqueda hasta que se encuentra una coincidencia.
- Cuando se carga un usuario usando `library()` el espacio de nombres de ese paquete se sitúa en la segunda posición y todos los demás quedan debajo.
- Nótese que R tiene un espacio de nombres separado para funciones y no funciones por lo que es posible tener el objeto de nombre “c” y una función de nombre “c”.

Las *reglas de alcance* determinan como un valor es asociado con una variable libre en una función.

Alcance léxico

R utiliza *alcance léxico* o *alcance estático*. Una alternativa es el *alcance dinámico*.

El alcance léxico es particularmente útil para simplificar cálculos estadísticos. En R significa:

Los valores de las variables libres se buscan en el entorno donde fue definido la variable.

Consideremos el siguiente ejemplo:

```
make.power <- function(n) {
    pow <- function(x) {
        x^n
    }
    pow
}
```

Luego se podría tener:

```
> cube <- make.power(3)
> square <- make.power(2)
> cube(3)
[1] 27
> square(3)
[1] 9
```

Como se ve, las funciones `cube()` y `square()` toman el valor de `n` de las asignación que se hace al llamar a `make.power()`.

Manejo de datos temporales

R cuenta con una representación especial para fechas y el tiempo.

Las fechas se representan mediante la clase `Date`. Y se almacenan internamente como el número de días desde `1970-01-01`.

Una cadena de caracteres puede ser coercida a la clase `Date` utilizando la función `as.date()`.

```
> x <- as.Date("1970-01-01")
> x
[1] "1970-01-01"
> x + 1
[1] "1970-01-02"
> y <- as.Date("1970-02-01")
> y - x
Time difference of 31 days
```

Tener las variables en formato de fecha y tiempo facilita operaciones de sumar y sustraer fechas, y comparar fechas.

El tiempo se representa usando las clases `POSIXct` o `POSIXlt`. Y se almacenan internamente como el número de segundos desde `1970-01-01`.

`POSIXct` es un entero grande, útil para almacenar valores temporales en un data frame;

`POSIXlt` es una lista que almacena información adicional como el día de la semana, el día del año, mes, y día del mes.

Hay funciones genéricas que funcionan en fechas y tiempos:

- `weekdays()` : devuelve el día de la semana.
- `months()` : devuelve el nombre del mes.
- `quarters()` : devuelve el número del trimestre ("Q1", "Q2", "Q3", or "Q4")

Las funciones `as.POSIXct()` y `as.POSIXlt()` pueden utilizarse para coercer una cadena de carácter en un objeto temporal.

```
> x <- Sys.time()
> x
[1] "2014-10-11 14:14:45 VET"
> p <- as.POSIXlt(x)
> names(unclass(p))
 [1] "sec"    "min"    "hour"   "mday"   "mon"    "year"   "wday"   "yday"
 [9] "isdst"  "zone"   "gmtoff"
> p$zone
[1] "VET"
```

Se utiliza la función `strptime()` para convertir cadenas de carácter que se encuentran en distintos formatos en objetos de clase fecha o tiempo.

```
> datestring <- c("Enero 10, 2012 10:40", "Diciembre 9, 2011 9:10")
> x <- strptime(datestring, "%B %d, %Y %H:%M")
> x
[1] "2012-01-10 10:40:00 VET" "2011-12-09 09:10:00 VET"
```

Es importante notar que los nombres de los meses se indican en español porque R con la opción `%B` lee las nombres de mes de acuerdo a la configuración local del sistema.

Los argumentos de `strptime()` indican de forma resumida el formato de los datos temporales, por ejemplo: `"%Y %H:%M"` indica el año en cuatro dígitos, un espacio, la hora como número decimal, dos puntos y los minutos como un número decimal.

Utilizando la función `Sys.setlocale()` se posible establecer la configuración local de la sesión de R según sea necesario.

```
> x <- c("1jan1960", "2jan1960", "31mar1960", "30jul1960")
> z <- strptime(x, "%d%b%Y")
> z
[1] NA               NA               "1960-03-31 VET" "1960-07-30 VET"
> lct <- Sys.getlocale("LC_TIME")
> Sys.setlocale("LC_TIME", "C")
[1] "C"
> x <- c("1jan1960", "2jan1960", "31mar1960", "30jul1960")
> z <- strptime(x, "%d%b%Y")
> z
[1] "1960-01-01 VET" "1960-01-02 VET" "1960-03-31 VET" "1960-07-30 VET"
> Sys.setlocale("LC_TIME", lct)
[1] "es_VE.UTF-8"
```

CAPÍTULO 3

Elementos de Programación con R II

A continuación, avanzaremos en las estructuras de repetición, procesos de simulación, herramientas de depuración y de mejora de rendimiento del código.

Bucles en la línea de comando

Hacer bucles `for`, `while` es muy útil cuando se programa, pero no resulta especialmente sencillo cuando se trabaja interactivamente con la línea de comando. Es por ello que hay algunas funciones que implementan bucles para hacer el trabajo más sencillo.

- `lapply`: Bucle a lo largo de una lista que evalúa una función en cada elemento
- `sapply`: Funciona del mismo modo que `lapply` pero trata de simplificar el resultado
- `apply`: Aplica una función sobre los márgenes de un arreglo
- `tapply`: Aplica una función sobre un subconjunto de un vector
- `mapply`: Una versión multivariante de `lapply`

Existe una función auxiliar llamada `split` que resulta también útil particularmente de modo conjunto con `lapply`

`lapply`

`lapply` toma tres argumentos: (1) una lista d `X`; (2) una función (o el nombre de una función) FUN; (3) otros argumentos a través de su ... argumento. Si `X` no es una lista, será forzado a serlo utilizando as.list.

`lapply`

```
## function (X, FUN, ...)
## {
##     FUN <- match.fun(FUN)
##     if (!is.vector(X) || is.object(X))
##         X <- as.list(X)
##     .Internal(lapply(X, FUN))
## }
## <bytecode: 0x7ff7a1951c00>
## <environment: namespace:base>
```

El bucle actual se realiza internamente en código C

`lapply` siempre devuelve una lista, dependiendo de la clase de la salida.

```
x <- list(a = 1:5, b = rnorm(10))
lapply(x, mean)

## $a
## [1] 3
##
## $b
## [1] 0.4671

x <- list(a = 1:4, b = rnorm(10), c = rnorm(20, 1), d = rnorm(100, 5))
lapply(x, mean)

## $a
## [1] 2.5
##
## $b
## [1] 0.5261
##
## $c
## [1] 1.421
##
## $d
## [1] 4.927

> x <- 1:4
> lapply(x, runif)
[[1]]
[1] 0.2675082

[[2]]
[1] 0.2186453 0.5167968

[[3]]
[1] 0.2689506 0.1811683 0.5185761

[[4]]
[1] 0.5627829 0.1291569 0.2563676 0.7179353

> x <- 1:4
> lapply(x, runif, min = 0, max = 10)
[[1]]
[1] 3.302142

[[2]]
[1] 6.848960 7.195282

[[3]]
[1] 3.5031416 0.8465707 9.7421014

[[4]]
[1] 1.195114 3.594027 2.930794 2.766946
```

`lapply` y sus amigas tienen un uso muy importante en funciones *anónimas*.

```
> x <- list(a = matrix(1:4, 2, 2), b = matrix(1:6, 3, 2))
> x
$a

     [,1] [,2]
```

```
[1,]  1   3
[2,]  2   4

$b
     [,1] [,2]
[1,]  1   4
[2,]  2   5
[3,]  3   6
```

Una función anónima para extraer la primera columna de cada matriz

```
> lapply(x, function(elt) elt[,1])
$a
[1] 1 2

$b
[1] 1 2 3
```

sapply

`sapply` trata de simplificar el resultado de `lapply` si es posible.

- Si el resultado es una lista donde cada elemento tiene longitud 1, entonces se devuelve un vector
- Si el resultado es una lita donde cada elemento es un vector de la misma longitud (> 1), se devuelve una matriz
- Si no se puede encontrar, se devuelve una lista.

```
> x <- list(a = 1:4, b = rnorm(10), c = rnorm(20, 1), d = rnorm(100, 5))
> lapply(x, mean)
$a
[1] 2.5

$b
[1] 0.06082667

$c
[1] 1.467083

$d
[1] 5.074749

> sapply(x, mean)
        a           b       c       d
2.50000000 0.06082667 1.46708277 5.07474950

> mean(x)
[1] NA
Warning message:
In mean.default(x) : argument is not numeric or logical: returning NA
```

Simulación

Generación de números aleatorios

Las funciones para distribuciones aleatorias en R son:

- `rnorm`: genera variables aleatorias de tipo Normal con una media y desviación estándar dadas.
- `dnorm`: evalúa la densidad de probabilidad de una Normal (con una media y desviación estándar dadas) en un punto (o vector de puntos)
- `pnorm`: evalúa la función de la distribución acumulada para una distribución Normal
- `rpois`: genera variables aleatorias Poisson con un índice dado.

Las funciones de distribución de probabilidad, generalmente tienen cuatro funciones asociadas a ellas. Estas son:

- `d` para densidad
- `r` para generación de número aleatorio
- `p` para distribución acumulada
- `q` para función cuantil.

El trabajo con distribuciones tipo Normal requiere el uso de estas cuatro funciones

```
dnorm(x, mean = 0, sd = 1, log = FALSE)
pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
rnorm(n, mean = o, sd = 1)
```

If Φ es la función de distribución acumulada de una distribución Normal estandar, entonces `pnorm(q)` = $\Phi(q)$ y `qnorm(p)` = $\Phi^{-1}(p)$

```
> x <- rnorm(10)
> x
  [1] 1.38380206 0.48772671 0.53403109 0.66721944
  [5] 0.01585029 0.37945986 1.31096736 0.55330472
  [9] 1.22090852 0.45236742
> x <- rnorm(10, 20, 2)
> x
  [1] 23.38812 20.16846 21.87999 20.73813 19.59020
  [6] 18.73439 18.31721 22.51748 20.36966 21.04371
> summary(x)
   Min.   1st Qu. Median  Mean    3rd Qu.  Max.
  18.32   19.73   20.55   20.67   21.67    23.39
```

Configurar semillas de números aleatorios con `set.seed` asegura su replicación

```
> set.seed(1)
> rnorm(5)
[1] -0.6264538  0.1836433  -0.8356286  1.5952808
[5]  0.3295078
> rnorm(5)
[1] -0.8204684  0.4874291   0.7383247  0.5757814
[5] -0.3053884
> set.seed(1)
> rnorm(5)
[1] -0.6264538  0.1836433  -0.8356286  1.5952808
[5]  0.3295078
```

¡Recuerde siempre establecer la semilla del número aleatorio cuando ejecuta una simulación!

Para generar datos Pisson

```
> rpois(10, 1)
 [1] 3 1 0 1 0 0 1 0 1 1
> rpois(10, 2)
```

```
[1] 6 2 2 1 3 2 2 1 1 2
> rpois(10, 20)
[1] 20 11 21 20 20 21 17 15 24 20


> ppois(2, 2)  ## Cumulative distribution
[1] 0.6766764  ## Pr(x <= 2)
> ppois(4, 2)
[1] 0.947347   ## Pr(x <= 4)
> ppois(6, 2)
[1] 0.9954662  ## Pr(x <= 6)
```

Generando números aleatorios a partir de un modelo lineal

Supongamos que queremos hacer una simulación a partir del siguiente modelo lineal

$$y = \beta_0 + \beta_1 x + \epsilon$$

donde $\epsilon \sim N(0, 2^2)$. Se asume x $N(0, 1^2)$, $\beta_0 = 0{,}5$ y $\beta_1 = 2$

```
> set.seed(20)
> x <- rnorm(100)
> e <- rnorm(100, 0, 2)
> y <- 0.5 + 2 * x + e
> summary(y)
   Min.     1st Qu.   Median
 -6.4080   -1.5400    0.6789  0.6893  2.9300   6.5050
> plot(x, y)
```

¿Qué pasa si `x` es binaria?

```
> set.seed(10)
> x <- rbinom(100, 1, 0.5)
> e <- rnorm(100, 0, 2)
> y <- 0.5 + 2 * x + e
> summary(y)
   Min.   1st Qu.   Median
-3.4940  -0.1409    1.5770    1.4320   2.8400   6.9410
> plot(x, y)
```

Generando números aleatorios a partir de un Modelo Lineal Generalizado

Supongamos que queremos hacer una simulación a partir de un modelo Poisson donde

$Y \sim Poisson(\mu)$

$log\mu = \beta_0 + \beta_1 x$

y $\beta_0 = 0{,}5$ y $\beta_1 = 0{,}3$

En ese caso, se requiere el uso de la función `rpois`

```
> set.seed(1)
> x <- rnorm(100)
> log.mu <- 0.5 + 0.3 * x
> y <- rpois(100, exp(log.mu))
> summary(y)
   Min.  1st Qu. Median    Mean  3rd Qu.  Max.
```

```
   0.00    1.00    1.00    1.55    2.00    6.00
> plot(x, y)
```

Muestreo Aleatorio

La función `sample` hace un gráfico aleatorio a partir de un conjunto específico de objetos (escalares) permitiéndole hacer un muestreo a partir de distribuciones arbitrarias.

```
> set.seed(1)
> sample(1:10, 4)
[1] 3 4 5 7
> sample(1:10, 4)
[1] 3 9 8 5
> sample(letters, 5)
[1] "q" "b" "e" "x" "p"
> sample(1:10)  ## permutation
 [1] 4 710 6 9 2 8 3 1 5
> sample(1:10)
 [1] 2 3 4 1 9 5 10 8 6 7
> sample(1:10, replace = TRUE) ## Sample w/replacement
 [1] 2 9 7 8 2 8 5 9 7 8
```

Resumen

- Para realizar gráficos de muestras para distribuciones específicas de probabilidad pueden utilizarse las funciones `r*`
- Las distribuciones estandar son: Normal, Poisson, Binomial, Exponencial, Gamma, etc.
- La función `sample` puede utilizarse para graficar muestras aleatorias a partir de vectores arbitrarios
- Para la replicación del modelo, resulta muy importante configurar el generador de números aleatorios a través de `set.seed`

Herramientas de depuración

Algunos indicadores de que algo no está funcionando bien:

- `message`: Un mensaje genérico de notificación/diagnóstico producido por la función `message`. La ejecución de la función continúa
- `warning`: Un indicador de que hay algún problema aunque no neesariamente es fatal. Es generado por la función `warning`. La función continúa ejecutándose.
- `error`: Un indicador de que ocurrió un problema fatal. La ejecución se interrumpe. Se produce por la función `stop`
- `condition`: Un concepto genérico para indicar que algo inesperado puede ocurrir. Los programadores pueden crear sus propias condiciones.

warning

```
log(-1)

## Warning: NaNs produced

## [1] NaN

printmessage <- function(x) {
        if(x > 0)
                print("x is greater than zero")
        else
                print("x is less than or equal to zero")
        invisible(x)
}

printmessage <- function(x) {
    if (x > 0)
        print("x is greater than zero") else print("x is less than or equal to zero")
    invisible(x)
}
printmessage(1)

## [1] "x es mayor que cero"

printmessage(NA)

## Error: missing value where TRUE/FALSE needed

printmessage2 <- function(x) {
        if(is.na(x))
                print("x es un valor desconocido!")
        else if(x > 0)
                print("x es mayor que cero")
        else
                print("x es menor o igual a cero")
        invisible(x)
}

printmessage2 <- function(x) {
    if (is.na(x))
        print("x is a missing value!") else if (x > 0)
        print("x is greater than zero") else print("x is less than or equal to zero")
    invisible(x)
}
x <- log(-1)

## Warning: NaNs produced

printmessage2(x)

## [1] "x es un valor desconocido!"
```

¿Cómo saber que algo está mal en una función?

- ¿Cuál es la entrada? ¿Puede ser llamada la función?
- ¿Qué se espera recibir? Saludos, mensajes otros resultados
- ¿Qué se obtuvo?
- ¿En qué difiere el resultado obtenido del esperado?

- ¿Las expectativas iniciales fueron correctas?
- ¿El problema es reproducible (exactamente)?

Herramientas de depuración en R

Las herramientas básicas para depuración en R son

- `traceback`: imprime la lista de llamadas a una función después que ocurre un error. No produce ningun resultado si no hay error.
- `debug`: marca una función para el modo "depuración" lo cual permite seguir la ejecución de una función una línea por vez.
- `browser`: suspende la ejecución de una función donde sea que sea llamada y coloca la función en modo depuración.
- `trace`: permite insertar un código de depuración en una función en lugares específicos.
- `recover`: permite modificar el comportamiento del error de modo que pueda navegar por la lista de llamados a la función

Estas son herramientas interactivas específicamente diseñadas para permitir escoger a través de una función.

Existe también técnicas más contundentes como la inserción de declaraciones `print/cat` en la función.

traceback

```
> mean(x)
Error in mean(x) : object 'x' not found
> traceback()
1: mean(x)
>

> lm(y ~ x)
Error in eval(expr, envir, enclos) : object 'y' not found
> traceback()
7: eval(expr, envir, enclos)
6: eval(predvars, data, env)
5: model.frame.default(formula = y ~ x, drop.unused.levels = TRUE)
4: model.frame(formula = y ~ x, drop.unused.levels = TRUE)
3: eval(expr, envir, enclos)
2: eval(mf, parent.frame())
1: lm(y ~ x)
```

debug

```
> debug(lm)
> lm(y ~ x)
    debugging in: lm(y ~ x)
    debug: {
    ret.x <- x
    ret.y <- y
    cl <- match.call()
    ...
    if (!qr)
        z$qr <- NULL
```

```
    z
}
Browse[2]>

Browse[2]> n
debug: ret.x <- x
Browse[2]> n
debug: ret.y <- y
Browse[2]> n
debug: cl <- match.call()
Browse[2]> n
debug: mf <- match.call(expand.dots = FALSE)
Browse[2]> n
debug: m <- match(c("formula", "data", "subconjunto", "pesos", "na.accion",
    "offset"), names(mf), 0L)
```

recover

```
> options(error = recover)
> read.csv("nosuchfile")
Error in file(file, "rt") : cannot open the connection
In addition: Warning message:
In file(file, "rt") :
    cannot open file 'nosuchfile': No such file or directory

Enter a frame number, or 0 to exit


1: read.csv("nosuchfile")
2: read.table(file = file, header = header, sep = sep, quote = quote, dec =
3: file(file, "rt")


Selection:
```

Resumen

- Hay tres indicadores principales de una condición/problema: `message`, `warning`, `error`
  - sólo un `error` es fatal
- Cuando se analiza una función con un problema, hay que asegurarse que el problema puede ser reproducido, clarificar el estatus de las expectativas y cómo la salida difiere de las expectativas iniciales.
- Las herramientas interactivas de depuración `traceback`, `debug`, `browser`, `trace` y `recover` pueden usarse para encontrar código con problemas en funciones.
- ¡Las herramientas de depuración no sustituyen al razoamiento!

Mejora del rendimiento del código

¿Por qué el código es tan lento?

- La refinación es una forma sistemática de examinar cuánto tiempo se demoran las distintas partes de un programa.
- Es útil cuando se intenta optimizar el código
- A menudo el código se ejecuta bien una vez, pero ¿qué ocurre cuando debe generarse un bucle para 1.000 iteraciones? ¿es lo suficientemente rápido?
- La refinación es mejor que el tanteo.

Sobre optimizar el código

- Obtener el mayor impacto en la aceleración del código depende de conocer en dónde se demora el código más tiempo.
- No puede realizarse la optimización del código sin un analisis de desempeño o un refinamiento.

> *Debemos olvidarnos de las pequeñas eficiencias, casi un 97 % de las veces decir: optimización prematura es la raíz de todos los males Donald Knuth*

Principios generales de optimización

- Diseñe primero, luego optimice
- Recuerde: la optimización prematura es la raíz de todos los males
- Mida (recolecte datos), no tantee.
- ¡Si va a hacer ciencia, debe utilizar los mismos principios!

Utilizando `system.time()`

- Tome una expresión arbitraria de R como entrada (puede estar encerrada entre llaves) y observe la cantidad de tiempo que se toma en evaluar esa expresión.
- Calcule, en segundos, el tiempo necesitado para ejecutar una expresión
  - Si hay un error, de tiempo hasta que ocurra
- Devuelva un objeto de clase `proc_time`
  - `user time`: tiempo asignado al(os) CPU(s) para esta expresión
  - `elapsed time`: tiempo de "reloj de pared"
- Con frecuencia, el `user time` y el `elapsed time`, para tareas correctas de cómputo, tienen valores relativamente similares.
- `elapsed time` puede ser *mayor* que `user time`, si el CPU gasta mucho tiempo esperando.
- `elapsed time` puede ser *menor* que `user time` si la máquina es multicore o multi procesador (y los utiliza)
  - Librerías BLAS multiciclo (vcLib/Accelerate, ATLAS, ACML, MKL)

- Procesamiento paralelo a través del paquete `parallel`

```
## Elapsed time > user time
system.time(readLines("http://www.jhsph.edu"))
    user    system  elapsed
    0.004   0.002    0.431


## Elapsed time < user time
hilbert <- function(n) {
        i <- 1:n
        1 / outer(i - 1, i, "+")
}
x <- hilbert(1000)
system.time(svd(x))
    user   system   elapsed
    1.605  0.094    0.742
```

Tiempo en expresiones largas

```
system.time({
    n <- 1000
    r <- numeric(n)
    for (i in 1:n) {
        x <- rnorm(n)
        r[i] <- mean(x)
    }
})
```

```
##    user  system  elapsed
##   0.097   0.002    0.099
```

Más allá del `system.time()`

- Utilizar `system.time()` permite provar ciertas funciones o bloques de código para ver si toman excesivo tiempo en su ejecución.
- Si se conoce dónde está el problema, puede hacerse la llamada a la función `system.time()` en ese punto.
- Pero ¿Qué pasa si no se sabe por dónde comenzar?

El refinador de R

- La función `Rprof()` inicia el refinador de R
  - R debe compilarse con el soporte para refinador
- La función `summaryRprof()` sumariza la salida de `Rprof()`
- NO utilice `system.time()` y `Rprof()` juntas o se entristecerá
- `Rprof()` hace seguimiento de la función a intervalos de muestreo regulares, y tabula cuánto tiempo se utiliza en cada función.
- El intervalo de muestreo por defecto es 0.02 segs.
- NOTA: si el código se ejecuta con rapidez, el refinador no es útil, de hecho, puede que no sea necesario usarlo.

Salida en bruto del refinador de R

```
## lm(y ~ x)


sample.interval=10000
"list" "eval" "eval" "model.frame.default" "model.frame" "eval" "eval" "lm"
"list" "eval" "eval" "model.frame.default" "model.frame" "eval" "eval" "lm"
"list" "eval" "eval" "model.frame.default" "model.frame" "eval" "eval" "lm"
"list" "eval" "eval" "model.frame.default" "model.frame" "eval" "eval" "lm"
"na.omit" "model.frame.default" "model.frame" "eval" "eval" "lm"
"na.omit" "model.frame.default" "model.frame" "eval" "eval" "lm"
"na.omit" "model.frame.default" "model.frame" "eval" "eval" "lm"
"na.omit" "model.frame.default" "model.frame" "eval" "eval" "lm"
"na.omit" "model.frame.default" "model.frame" "eval" "eval" "lm"
"na.omit" "model.frame.default" "model.frame" "eval" "eval" "lm"
"na.omit" "model.frame.default" "model.frame" "eval" "eval" "lm"
"lm.fit" "lm"
"lm.fit" "lm"
"lm.fit" "lm"
```

Utilizando `summaryRprof()`

- La función `summaryRprof()` tabula la salida del refinador de R y calcula cuánto tiempo demora cada una de las funciones
- Hay dos métodos para normalizar los datos
- `by.total` divide el tiempo utilizado en cada función entre el total del tiempo de ejecución.
- `by.self` hacelo mismo pero primero sustrae el tiempo que es utilizado en funciones antes de la lista de llamados.

By Total

```
$by.total

                      total.time total.pct  self.time  self.pct
"lm"                        7.41    100.00       0.30      4.05
"lm.fit"                    3.50     47.23       2.99     40.35
"model.frame.default"       2.24     30.23       0.12      1.62
"eval"                      2.24     30.23       0.00      0.00
"model.frame"               2.24     30.23       0.00      0.00
"na.omit"                   1.54     20.78       0.24      3.24
"na.omit.data.frame"        1.30     17.54       0.49      6.61
"lapply"                    1.04     14.04       0.00      0.00
"[.data.frame"              1.03     13.90       0.79     10.66
"["                         1.03     13.90       0.00      0.00
"as.list.data.frame"        0.82     11.07       0.82     11.07
"as.list"                   0.82     11.07       0.00      0.00
```

by self

```
$by.self
```

	self.time	self.pct	total.time	total.pct
"lm.fit"	2.99	40.35	3.50	47.23
"as.list.data.frame"	0.82	11.07	0.82	11.07
"[.data.frame"	0.79	10.66	1.03	13.90
"structure"	0.73	9.85	0.73	9.85
"na.omit.data.frame"	0.49	6.61	1.30	17.54
"list"	0.46	6.21	0.46	6.21
"lm"	0.30	4.05	7.41	100.00
"model.matrix.default"	0.27	3.64	0.79	10.66
"na.omit"	0.24	3.24	1.54	20.78
"as.character"	0.18	2.43	0.18	2.43
"model.frame.default"	0.12	1.62	2.24	30.23
"anyDuplicated.default"	0.02	0.27	0.02	0.27

Salida de `summaryRprof()`

```
$sample.interval
[1] 0.02

$sampling.time
[1] 7.41
```

Resumen

- `Rprof()` ejecuta el refinador de desempeño de analisis del código R
- `summaryRprof()` sumariza la salida de `Rprof()` y asigna un porcentaje al tiempo utilizado en cada función (con dos tipos de normalización)
- Es sano cortar el código en funciones de modo que el refinador pueda aportar información útil sobre donde se está usando el tiempo.
- Los códigos C y Fortran no se refinan

CAPÍTULO 4

Gestión de Datos

Datos crudos y datos ordenados

Se sabe que se está trabajando con *datos crudos* porque:

- No se ha ejecutado software sobre los datos
- No se ha manipulado ningún valor de los datos
- No se ha quitado ningún valor de los datos
- No se han resumido ni agregado los datos de ninguna manera.

El objetivo es obtener *datos ordenados*, los cuales se caracterizan por:

- Cada variable medida debe estar en una columna diferente.
- Cada observación diferente de esa variable debe estar en una fila diferente
- Debe haber una tabla para cada "tipo" de variable.
- Si tiene varias tablas, debe tener una columna en la tabla que permita enlazarlas.

Observar las siguientes sugerencias:

- Es conveniente añadir una fila en la parte superior de cada archivo (cabecera) con los nombres de las variables.
- Los nombres de las variables deben ser legibles. Por ejemplo, es preferible "EdadIngreso" en lugar de "EdadI".
- En general, se debe guardar una tabla por archivo.

Se debe contar con o construir un *libro de códigos*. Esto es, una descripción del proceso de conversión de datos crudos a ordenados que contenga:

- Información acerca de las variables y sus unidades.
- Información sobre las decisiones de resumen o agregación de las variables
- Información sobre el diseño experimental utilizado.

Algunas recomendaciones:

- Debería haber una sección denominada *Diseño del estudio* que contenga una descripción del proceso de recolección de los datos.
- Debe haber una sección denominada *Libro de códigos* que describa cada variable y sus unidades.

El proceso de conversión de los datos debe ser automatizado mediante un script.

- La entrada del script deben ser los datos crudos.

- La salida del script, los datos ordenados.
- El script no debe tener parámetros.

En los casos que el proceso no pueda ser completamente automático se deben proveer instrucciones para la conversión.

Gestión de archivos

Por defecto R trabaja con los archivos sobre el *directorio de trabajo*, los dos comandos básicos relacionados son:

- `getwd()` que devuelve el directorio de trabajo (*working directory*) actual.
- `setwd(<ruta>)` que establece el directorio de trabajo en la ruta indicada.

Considere la diferencia entre rutas relativas y absolutas.

- Rutas *relativas*: `setwd("./data")`, `setwd("../")`
- Rutas *absolutas*: `setwd("/home/usuario/datos")`

Nota: En Windows las rutas se escriben: `setwd("C:\\Usuarios\\usuario\\descargas")`

Es frecuente al automatizar procesos verificar si existe un archivo y crear carpetas, para esto se cuenta con las funciones:

- `file.exists()` para verificar si un archivo o carpeta existe.
- `dir.create()` creará una carpeta si no existe

Un uso frecuente de estas funciones es como sigue:

```
if !file.exists("datos") {
    dir.create("datos")
}
```

Descargar archivos de Internet

Para descargar archivos de Internet se puede utilizar la función `download.file()`.

Este tipo de funciones son muy útiles para automatizar tareas rutinarias y para reproducir tareas.

```
fileURL <- "http://api.worldbank.org/v2/es/country/ven?downloadformat=csv")
download.file(fileURL, destfile = "./data/ve_worldbank.zip", method = "curl")
list.files("./data")
```

Si la ruta empieza con `http` se puede utilizar `download.file()` con las opciones por defecto. Si la conexión comienza en `https` en Mac o Linux puede necesitar `method = "curl"`.

Registre la fecha de su descarga, tomando la fecha actual con la función `date()`.

```
if (!file.exists("data")) {
    dir.create("data")
}
fileUrl <- "https://data.baltimorecity.gov/api/views/dz54-2aru/rows.csv?accessType=DOWNLOAD"
download.file(fileUrl, destfile = "cameras.csv", method = "curl")
dateDownloaded <- date()
```

Lectura de datos desde distintos tipos de fuentes

Datos en texto plano

Para leer datos en texto plano la función básica es `read_table()`. Es la función mas flexible y robusta pero la que requiere el mayor número de parámetros. Los datos se cargan a la memoria RAM, conjuntos grandes de datos pueden ser problemáticos.

- Parámetros importantes: `file`, `header`, `sep`, `row.names`, `nrows`.
- Algunas funciones relacionadas son: `read.csv()`, `read.csv2()`.

```
cameraData <- read.table("./data/cameras.csv", sep = ",", header = TRUE)
head(cameraData)
```

Si se utiliza `read.csv`, se establece `sep = ","`, `header = TRUE`.

```
cameraData <- read.csv("./data/cameras.csv")
head(cameraData)
```

Archivos de Excel

Los archivos de Excel son el estándar de facto para el intercambio de datos.

```
if(!file.exists("data")){dir.create("data")}
fileUrl <- "http://www.ine.gov.ve/documentos/Demografia/EstadisticasVitales/xls/Nacimientos_Entidad.x
download.file(fileUrl, destfile="./data/nacimientos_entidad.xlsx", method="curl")
dateDownloaded <- date()
```

Para leer datos desde archivos de Excel es necesario instalar el paquete `xlsx`, el cual provee las funciones `read.xlsx()` y `read.xlsx2()`.

```
library(xlsx)
nacimientosData <- read.xlsx("./data/nacimientos_entidad.xlsx",
                             sheetIndex=1, header=TRUE)
head(cameraData)
```

Con frecuencia se requiere leer los datos de filas y columnas especificas.

```
colIndex <- 2:14
rowIndex <- 10:32
nacimientosFiltro <- read.xlsx("./data/nacimientos_entidad.xlsx",
                               sheetIndex=1,
                               colIndex=colIndex, rowIndex=rowIndex)
nacimientosFiltro
```

- La función `write.xlsx()` puede utilizarse para generar archivos de Excel con argumentos similares.
- La función `read.xlsx2()` es mucho mas rápida que `read.xlsx()` pero puede ser inestable para leer de subconjuntos de filas.
- El paquete XLConnect tiene mas opciones para escribir y manipular archivos de Excel. La viñeta de XLConnect es un buen lugar para empezar con ese paquete.

Importante: Se recomienda encarecidamente almacenar los datos ya sea en bases de datos o archivos separados por comas (.csv) ya que permiten distribuir y procesar la información con mucha mayor facilidad.

Leyendo datos de la web

La tarea de extraer datos mediante programas de computadora del código HTML de los sitios web se denomina Webscraping

Hay muchos sitios web que publican periódicamente datos de interés que se pueden leer de forma automática.

Intentar leer muchas páginas en un corto período de tiempo puede ocasionar que su número IP sea bloqueado.

Por ejemplo, para descargar datos de páginas en Google Scholar. Se pueden descargar los datos de un sitio web aplicando la función `readLines()` sobre una conexión a un sitio web.

```
con = url("http://scholar.google.com/citations?user=HI-I6C0AAAAJ&hl=en")
htmlCode = readLines(con)
close(con)
htmlCode
```

Como sabemos, HTML es un dialecto particular de XML, de manera que se pueden extraer elementos de un sitio web con el paquete `XML`

```
library(XML)
url <- "http://scholar.google.com/citations?user=HI-I6C0AAAAJ&hl=en"
html <- htmlTreeParse(url, useInternalNodes=T)

xpathSApply(html, "//title", xmlValue)
xpathSApply(html, "//td[@id='col-citedby']", xmlValue)
```

la función *xpathSApply()* permiten extraer elementos de la estructura del código HTML del sitio web.

El paquete httr puede utilizarse de forma similar para extraer elementos desde las etiquetas de los sitios web. Este paquete provee varias funciones útiles:

- la función *GET()* realiza una solicitud al sitio web, lo cual incluye metadatos de la conexión.
- la función `content()` toma el contenido de la solicitud
- y finalmente, la función `htmlParse()` del paquete `XML` nos permite extraer datos de las etiquetas del código HTML.

```
library(httr); html2 = GET(url)
content2 = content(html2, as="text")
parsedHtml = htmlParse(content2, asText=TRUE)
xpathSApply(parsedHtml, "//title", xmlValue)
```

Cuando se requiere autenticación para acceder a un sitio web se puede especificar el argumento `authenticate` de la función `GET()`.

```
pg1 = GET("http://httpbin.org/basic-auth/user/passwd")
pg1


pg2 = GET("http://httpbin.org/basic-auth/user/passwd",
          authenticate("user","passwd"))
pg2
names(pg2)
```

Utilizando "manejadores".

```
google = handle("http://google.com")
pg1 = GET(handle=google,path="/")
pg2 = GET(handle=google,path="search")
```

Los manejadores preservan las configuraciones y los "cookies" a los largo de solicitudes múltiples. Es la base de todas las solicitudes hechas a través del paquete `httr`

El blog R Bloggers tiene una gran cantidad de ejemplos de webscraping. La ayuda del paquete httr ofrece muchos ejemplos útiles.

Leyendo desde APIs

Las interfaces de programación de aplicaciones (APIs por su acrónimo en inglés) ofrecen mecanismos potentes y flexibles para acceder a los datos que subyacen en las aplicaciones web.

La gran ventaja es que permiten hacer solicitudes específicas en lugar de descargar conjuntos completos de datos, y porque permite compartir datos entre sistemas distintos.

Es de particular interés extraer datos de las aplicaciones de las "redes sociales". Para esto generalmente hay que crear una aplicación en uno de estos sistemas y obtener las claves de autenticación de dicha aplicación.

El ejemplo de una conexión al API sería como sigue:

```
myapp = oauth_app("twitter",
                  key="yourConsumerKeyHere",
                  secret="yourConsumerSecretHere")
sig = sign_oauth1.0(myapp,
                    token = "yourTokenHere",
                    token_secret = "yourTokenSecretHere")
homeTL = GET("https://api.twitter.com/1.1/statuses/home_timeline.json", sig)
```

El estándar de facto de transferencia de información entre aplicaciones mediante sus APIs es JSON (Javascript Object Notation), es una forma de expresar información como valores del lenguaje javascript.

El paquete `jsonlite` provee funciones para el manejo de información en formato JSON.

```
json1 = content(homeTL)
json2 = jsonlite::fromJSON(toJSON(json1))
json2[1,1:4]
```

- Las solicitudes de `httr`, `GET()`, `POST()`, `PUT()`, *DELETE()* ofrecen opciones para conexiones con autenticación.
- La mayoría de los APIs actuales utilizan mecanismos estándares de autenticación como OAUTH.
- `httr` funciona bien con Facebook, Google, Twitter, Github, etc.

Herramientas básicas para limpiar y manipular datos

Un repaso sobre filtros

```
set.seed(13435)
X <- data.frame("var1"=sample(1:5),"var2"=sample(6:10),"var3"=sample(11:15))
X <- X[sample(1:5),]; X$var2[c(1,3)] = NA
X
```

Se puede acceder a los datos de un data frame con una notación matricial `data[num_fila, num_col|nombre_col]`. O bien de la forma `data$nombre_col[num_fila]`.

Pasar un vector como índice permite reordenar o seleccionar (incluso repetidas) filas y/o columnas del data frame.

Las columnas se pueden acceder ya sea por posición o por nombre.

```
X[,1]
X[,"var1"]
X[1:2,"var2"]
```

También se pueden aplicar filtros como expresiones lógicas. Estas devuelven vectores lógicos.

```
X[(X$var1 <= 3 & X$var3 > 11), ]
X[(X$var1 <= 3 | X$var3 > 15), ]
```

La función `which()` devuelve los índices para los que se cumple una condición.

```
X[which(X$var2 > 8), ]
```

Para ordenar un vector se puede utilizar la función `sort()`. El argumento `decreasing` indica si se ordena en orden decreciente, y `na.last` si los valores faltantes se colocan al final.

```
sort(X$var1)
sort(X$var1, decreasing=TRUE)
sort(X$var2, na.last=TRUE)
```

La función `order()` devuelve una lista con los índices que permiten reordenar una columna. Se pueden introducir columnas adicionales para romper empates. Es útil para ordenar un data frame completo.

```
X[order(X$var1), ]
X[order(X$var1,X$var3), ]
```

El paquete `plyr` ofrece un conjunto de herramientas para facilitar tareas de separar, aplicar y combinar datos. La función `arrange()` de `plyr` ofrece un mecanismo mas comprensible para ordenar datos.

```
library(plyr)
arrange(X, var1)
arrange(X, desc(var1))
```

Para añadir un nueva variable a un data frame simplemente se asigna un vector a un nuevo nombre de columna.

```
X$var4 <- rnorm(5)
X
```

Se pueden añadir las funciones `cbind()` y `rbind()` para unir vectores o data frames como columnas o filas respectivamente. Las dimensiones de los objetos deben coincidir.

Y <- cbind(X,rnorm(5)) Y

Mas sobre este tema en la notas de Andrew Jaffe.

Haciendo resúmenes de sus datos

Para los siguientes ejemplos, empezar por descargar algunos datos de la web:

```
if(!file.exists("./data")){dir.create("./data")}
fileUrl <- "https://data.baltimorecity.gov/api/views/k5ry-ef3g/rows.csv?accessType=DOWNLOAD"
download.file(fileUrl,destfile="./data/restaurants.csv",method="curl")
restData <- read.csv("./data/restaurants.csv")
```

Un pequeño vistazo a los datos descargados:

```
head(restData, n=3)
tail(restData, n=3)
```

Se obtiene un resumen descriptivo:

```
summary(restData)
```

Información en mayor profundidad:

```
str(restData)
```

Los cuantiles de las variables cuantitativas:

```
quantile(restData$councilDistrict, na.rm=TRUE)
quantile(restData$councilDistrict, probs=c(0.5,0.75,0.9))
```

Se construye una tabla de frecuencias

```
table(restData$zipCode, useNA="ifany")
```

Ahora una tabla de frecuencias cruzadas:

```
table(restData$councilDistrict, restData$zipCode)
```

Se verifica la existencia de valores faltantes:

```
sum(is.na(restData$councilDistrict))
any(is.na(restData$councilDistrict))
all(restData$zipCode > 0)
```

Valores faltantes por columna:

```
colSums(is.na(restData))
all(colSums(is.na(restData))==0)
```

Frecuencia de valores con características particulares.

```
table(restData$zipCode %in% c("21212"))
table(restData$zipCode %in% c("21212", "21213"))
```

Valores con características particulares.

```
restData[restData$zipCode %in% c("21212", "21213"), ]
```

Tablas cruzadas

Se toma como ejemplo de referencia la tabla *UCBAdmissions*. Se convierte a data frame.

```
data(UCBAdmissions)
DF = as.data.frame(UCBAdmissions)
summary(DF)
```

Se crea una tabla de referencia cruzada, nótese como se utilizan los factores `Gender` y `Admit`.

```
xt <- xtabs(Freq ~ Gender + Admit, data=DF)
xt
```

Para crear *tablas planas*. Empezamos por crear una columna que nos permita tener observaciones únicas. Luego se aplica la función `ftable()` (flat table)

```
warpbreaks$replicate <- rep(1:9, len = 54)
xt = xtabs(breaks ~., data=warpbreaks)
xt
ftable(xt)
```

Para obtener una medida del uso de memoria de cualquier objeto mediante la función `object.size()`.

```
fakeData = rnorm(1e5)
object.size(fakeData)
print(object.size(fakeData),units="Mb")
```

Conectar con bases de datos

MySQL

- MySQL Sistema de base de datos libre ampliamente usada.
- Ampliamente utilizado por aplicaciones de Internet
- Los dato están estructurados en: * Bases de datos * Tablas dentro de bases de datos * Campos dentro de tablas
- Cada fila es un registro

Desde la adquisición de SUN por Oracle, existe una versión comunitaria de MySQL denominada MariaDB.

Instalación de RMySQL

- En Linux o Mac: `` `install.packages("RMySQL")` ``
- En Windows:
  - Instrucciones oficiales - http://biostat.mc.vanderbilt.edu/wiki/Main/RMySQL (tambien puede ser útil para los usuarios Mac/Linux)
  - Guía potencialmente útil - http://www.ahschulz.de/2013/07/23/installing-rmysql-under-windows/

Conexión a bases de datos. Obtener una lista de las bases de datos disponibles.

```
ucscDb <- dbConnect(MySQL(),user="genome",
                    host="genome-mysql.cse.ucsc.edu")
result <- dbGetQuery(ucscDb,"show databases;"); dbDisconnect(ucscDb);
result
```

Conexión a la base de datos "hg19" y obtener una lista de sus tablas.

```
hg19 <- dbConnect(MySQL(),user="genome", db="hg19",
                    host="genome-mysql.cse.ucsc.edu")
allTables <- dbListTables(hg19)
length(allTables)
allTables[1:5]
```

Para obtener las dimensiones de una tabla en particular

```
dbListFields(hg19,"affyU133Plus2")
dbGetQuery(hg19, "select count(*) from affyU133Plus2")
```

Finalmente, obtener datos de la base de datos.

```
affyData <- dbReadTable(hg19, "affyU133Plus2")
head(affyData)
```

Si se quiere obtener un subconjunto de la tabla.

```
query <- dbSendQuery(hg19, "select * from affyU133Plus2 where misMatches between 1 and 3")
affyMis <- fetch(query); quantile(affyMis$misMatches)
affyMisSmall <- fetch(query,n=10); dbClearResult(query);
dim(affyMisSmall)
```

No hay que olvidar cerrar la conexión.

```
dbDisconnect(hg19)
```

Recursos adicionales:

- RMySQL vignette http://cran.r-project.org/web/packages/RMySQL/RMySQL.pdf
- Lista de comandos http://www.pantz.org/software/mysql/mysqlcommands.html
  - En ningún caso borrar, añadir, o enlazar tablas desde ensembl. Solamente select.
  - En general, tener cuidados con los comandos de MySQL
- Un post con un buen resúmen de comandos http://www.r-bloggers.com/mysql-and-r/

CAPÍTULO 5

Probabilidad y Distribuciones

Conceptos de probabilidades basados en R

Definición de probabilidad

Considerando un enfoque de probabilidad como proporción de ocurrencia de eventos, entonces es necesario definir un conjunto de eventos posibles.

Sea Ω denominado **espacio muestral**.

Entonces, un **evento** E se definiría como cualquier subconjunto de Ω.

Un evento simple o elemental ω es un elemento de Ω.

La no ocurrencia de ningún evento, se denomina **evento nulo** y se denota por $\emptyset$.

Si se realiza una serie de experimentos en cada uno de los cuales puede ocurrir algún evento elemental perteneciente a Ω. Para un gran número de experimentos se define la probabilidad de ω como el cociente entre el número de veces que ocurre dicho evento y el número total de experimentos.

Medida de probabilidad

Una, **medida de probabilidad** P, es una función sobre un conjunto de eventos posibles tal que:

1. Para un evento $E \subset \Omega$, $0 \leq P(E) \leq 1$
2. $P(\Omega) = 1$
3. Si E_1 y E_2 son eventos mutuamente excluyentes, $P(E_1 \cup E_2) = P(E_1) + P(E_2)$.

El punto 3 implica **aditividad finita**

$$P(\cup_{i=1}^{n} A_i) = \sum_{i=1}^{n} P(A_i)$$

donde los math:*{A_i}* son mutuamente exclusivos.

Variable aleatoria

Una **variable aleatoria** es el resultado numérico de un experimento.

Puede ser:

- *Discreta*: cuando solamente puede tomar valores de un conjunto enumerable. math:*P(X = k)*
- *Continua*: cuando puede tomar cualquier valor de la recta real o un subconjunto sobre esta. math:*P(X in A)*

Función de masa de probabilidad

Una **función de masa de probabilidad** (FMP) evaluada en un valor corresponde a la probabilidad que una variable aleatoria tome ese valor.

Un FMP válida debe satisfacer:

1. $p(x) \geq 0$, para todo valor posible de x
2. $\sum_x p(x) = 1$

La suma se toma sobre todos los posibles valores de x:

Un **función de densidad de probabilidad** (FDP), es una función asociada con una variable aleatoria continua.

Las áreas bajo las FDP se corresponden con las probabilidades de esa variable

Para ser una FDP, una función f debe satisfacer:

1. $f(x) \geq 0$ para todo x
2. El área bajo $f(x)$ es uno.

Un ejemplo:

Suponga que la proporción de llamadas de ayuda que se reciben cualquier día por una línea de ayuda está dada por:

$$f(x) = \begin{cases} 2x & \text{for } 1 > x > 0 \\ 0 & \text{otherwise} \end{cases}$$

¿Es una densidad matemáticamente valida?

$$x < -c(-0{,}5, 0, 1, 1, 1{,}5); y < -c(0, 0, 2, 0, 0) plot(x, y, lwd = 3, frame = FALSE, type = "l")$$

¿Cuál es la probabilidad que 75 % o menos de las llamadas sean atendidas?

$plot(x, y, lwd = 3, frame = FALSE, type = "l") polygon(c(0, ,75, ,75, 0), c(0, 0, 1{,}5, 0), lwd = 3, col = "lightblue") 1{,}5 * ,75/2 pl$

La **función de distribucion acumulada** (FDA) de una variable aleatoria :math:‘X‘‘está definida como la función:

$$F(x) = P(X \leq x)$$

- Esta definición aplica sin importar que X sea discreta o continua.

La **función de supervivencia** de una variable aleatoria X está definida como:

..math:: S(x) = P(X > x)

- Note que $S(x) = 1 - F(x)$
- Para variables aleatorias continuas, la FDP es la derivada de la FDA.

Ejemplo,

¿Cuál es la función de supervivencia y la FDA de la función de densidad considerada antes?

para $1 \geq x \geq 0$

$$F(x) = P(X \leq x) = \frac{1}{2} Base \times Altura = \frac{1}{2}(x) \times (2x) = x^2$$
$$S(x) = 1 - x^2$$

```
pbeta(c(0.4, 0.5, 0.6), 2, 1)
```

El **cuantil** $alpha^{simo}$ de una distribución con función de distribución F es el punto x_α tal que:

$$F(x_\alpha) = \alpha$$

- Un **percentil** es simplemente un cuantil con α expresado como un porcentaje.
- La **mediana** es el 50^{avo} percentile

. Ejemplo

- Se quiere resolver $0{,}5 = F(x) = x^2$
- Resulta en la solución:

```
sqrt(0.5)
```

- En consecuencia, alrededor de *r sqrt(0.5)* de las llamadas que se responden un día cualquiera es la mediana.
- R puede aproximar cuantiles para las distribuciones mas comunes.

```
qbeta(0.5, 2, 1)
```

Notas

- Cuando se refiere **medidas de la población** se trata de la mediana de los datos. La **mediana de la población** es la que se obtiene integrando la función de densidad.
- Un modelo de probabilidad conecta los datos a la población en base a supuestos.
- La mediana de la que se habla es el **estimando**, la mediana de la muestra será el **estimador**

Valores esperados

El **valor esperado** o la **media** de una variable aleatoria es el centro de su distribución.

Para variables aleatorias discretas X con FMP $p(X)$, se define como sigue:

$$E[X] = \sum_x xp(x).$$

donde la suma se toma sobre todo los posibles valores de x.

$E[X]$ representa el centro de masa de una colección de posiciones y pesos $\{x, p(x)\}$

Ejemplo, encuentre el centro de masa de las barras

```
library(UsingR); data(galton)
par(mfrow=c(1,2))
hist(galton$child,col="blue",breaks=100)
hist(galton$parent,col="blue",breaks=100)
```

Utilizando `manipulate`

```
library(manipulate)
myHist <- function(mu){
  hist(galton$child,col="blue",breaks=100)
  lines(c(mu, mu), c(0, 150),col="red",lwd=5)
  mse <- mean((galton$child - mu)^2)
  text(63, 150, paste("mu = ", mu))
```

```
  text(63, 140, paste("Imbalance = ", round(mse, 2)))
}
manipulate(myHist(mu), mu = slider(62, 74, step = 0.5))
```

El centro de masa es la media empírica

```
hist(galton$child, col="blue", breaks=100)
meanChild <- mean(galton$child)
lines(rep(meanChild,100), seq(0,150,length=100), col="red", lwd=5)
```

Ejemplo, suponga que se lanza una moneda y X se denota 0 o :math:‘1 que corresponden a cara y sello, respectivamente.

¿Cuál es el valor esperado de X?

$$E[X] = ,5 \times 0 + ,5 \times 1 = ,5$$

Note, que si se piensa de forma geométrica, la respuesta es obvia; si se colocan dos pesos iguales en 0 y 1, el centro de masa será $0{,}5$.

```
barplot(height = c(.5, .5), names = c(0, 1),
        border = "black",
        col = "lightblue",
        space = .75)
```

Ejemplo, suponga que se lanza un dado y X es el número que queda boca arriba.

¿Cuál es el valor esperado de X?

$$E[X] = 1 \times \frac{1}{6} + 2 \times \frac{1}{6} + 3 \times \frac{1}{6} + 4 \times \frac{1}{6} + 5 \times \frac{1}{6} + 6 \times \frac{1}{6} = 3{,}5$$

De nuevo, el argumento geométrico hace que la respuesta sea obvia sin cálculos.

Variables aleatoria continuas

Para una variable aleatoria continua, X con densidad f, el valor esperado se define como:

$$E[X] = \text{the area under the function } \; tf(t)$$

Esta definición está derivada de la definición del centro de masa para un cuerpo continuo.

Ejemplo, considere una densidad donde $f(x) = 1$ para x entre cero y uno.

¿Es una densidad válida?

Suponga que X sigue esta densidad; ¿Cuál es el valor esperado?

```
par(mfrow = c(1, 2))
plot(c(-0.25, 0, 0, 1, 1, 1.25), c(0, 0, 1, 1, 0, 0),
     type = "l", lwd = 3, frame = FALSE, xlab="", ylab = "")
title('f(t)')
plot(c(-0.25, 0, 1, 1, 1.25), c(0, 0, 1, 0, 0),
     type = "l", lwd = 3, frame = FALSE, xlab="", ylab = "")
title('t f(t)')
```

Reglas sobre los valores esperados

El valor esperado es una operador lineal.

Si a y b no son aleatorias y X y Y son dos variables aleatorias entonces:

- $E[aX + b] = aE[X] + b$
- $E[X + Y] = E[X] + E[Y]$

Ejemplo, si lanza una moneda X y simula una variable aleatoria uniforme Y, ¿Cuál es el valor esperado de su suma?

$$E[X + Y] = E[X] + E[Y] = ,5 + ,5 = 1$$

Otro ejemplo, si se lanza un dado dos veces. ¿Cuál es el valor esperado del promedio?

Sean X_1 y X_2 los resultados de los dos lanzamientos.

$$E[(X_1 + X_2)/2] = \frac{1}{2}(E[X_1] + E[X_2]) = \frac{1}{2}(3{,}5 + 3{,}5) = 3{,}5$$

Ejemplo,

1. Sea X_i para $i = 1, \ldots, n$ sea una colección de variables aleatorias, cada una de una distribución con media μ. 2. Calcule el valor esperado del promedio muestral de X_i.

$$to$$

$$E\left[\frac{1}{n}\sum_{i=1}^{n} X_i\right] =$$

$$\frac{1}{n}E\left[\sum_{i=1}^{n} X_i\right]$$

$$=$$

$$\frac{1}{n}\sum_{i=1}^{n} E\left[X_i\right]$$

$$=$$

$$\frac{1}{n}\sum_{i=1}^{n} \mu = \mu.$$

=

$$= \frac{1}{n}E\left[\sum_{i=1}^{n} X_i\right]$$

$$= \frac{1}{n}\sum_{i=1}^{n} E\left[X_i\right]$$

$$\frac{1}{n}\sum_{i=1}^{n} \mu = \mu.$$

Importante:

- En consecuencia, el valor esperado de la **media muestral** es la media de la población que se está intentando estimar.
- Cuando el valor esperado de un estimador es lo que trata de estimar, se dice que el estimador es **no sesgado**.

La varianza

- La varianza de una variable aleatoria es una medida de *dispersión*
- Si X es una variable aleatoria con media μ, la varianza de X está definida como:

$$Var(X) = E[(X - \mu)^2]$$

La distancia esperada (al cuadrado) alrededor de la media.

- La densidades con una mayor varianza están mas dispersas que las densidades con una menor varianza.

Forma computacional conveniente

$$Var(X) = E[X^2] - E[X]^2$$

- Si a es una constante entonces $Var(aX) = a^2 Var(X)$
- La raíz cuadrada de la varianza es denominada **desviación estándar**
- La desviación estándar tiene las mismas unidades que X

Ejemplo, ¿Cuál es la varianza muestral del resultado de lanzar una moneda?

- $E[X] = 3{,}5$
- $E[X^2] = 1^2 \times \frac{1}{6} + 2^2 \times \frac{1}{6} + 3^2 \times \frac{1}{6} + 4^2 \times \frac{1}{6} + 5^2 \times \frac{1}{6} + 6^2 \times \frac{1}{6} = 15{,}17$
- $Var(X) = E[X^2] - E[X]^2 \approx 2{,}92$

Ejemplo, ¿Cuál es la varianza muestral del resultado del lanzamiento de una moneda con con probabilidad de obtener cara igual a p?

- $E[X] = 0 \times (1 - p) + 1 \times p = p$
- $E[X^2] = E[X] = p$
- $Var(X) = E[X^2] - E[X]^2 = p - p^2 = p(1 - p)$

Interpretando las varianzas

- La desigualdad de Chebyshev's es útil para interpretar las varianzas
- Esta desigualdad establece que:

$$P(|X - \mu| \geq k\sigma) \leq \frac{1}{k^2}$$

Por ejemplo, la probabilidad que una variable aleatoria caiga más allá de k desviaciones estándar desde la media es menor que $1/k^2$

$$to$$

$$2\sigma \rightarrow$$
$$25\,\%$$
$$3\sigma \rightarrow$$
$$11\,\%$$
$$4\sigma \rightarrow$$
$$6\,\%$$

$25\,\%3\sigma$
$11\,\%4\sigma$
$6\,\%$

- Note que esto es solamente una cota, la probabilidad verdadera puede ser un poco menor.

Ejemplo:

- Se dice que los CIs están distribuidos con una media de 100 y una desviación estándar de 15
- ¿Cuál es la probabilidad que una persona seleccionada aleatoriamente tenga un CI superior a 160 o inferior a 40?
- Por lo tanto se quiere conocer la probabilidad que una persona tenga mas de 4 desviaciones estándar desde la media.
- La desigualdad de Chebyshev sugiere que no será superior a $6\,\%$
- Con frecuencia se cita que las distribuciones de los CIs tienen forma de campana, en este caso este límite es muy conservador.
- La probabilidad que obtener un valor aleatorio más allá de 4 desviaciones estándar desde la media está por el orden de 10^{-5} (una milésima de uno por ciento)

Distribución de Variables aleatorias

Simulación de probabilidad

Errores de decisión, significación y confianza

CAPÍTULO 6

Fundamentos de Inferencia Estadística

Cuando se cuenta con distribuciones de probabilidad que involucran parámetros y eventos (variables aleatorias), se pueden calcular las probabilidades de ocurrencia de eventos determinados (valores dados de una variable aleatoria).

La Inferencia Estadística consiste en el proceso inverso: usar muestras para predecir con exactitud como es la población, es decir, cuales parámetros caracterizan sus distribuciones de probabilidad, y tener una idea de cuan confiable serán estas predicciones.

Ejemplos de motivación

¿Quién ganará la elección?

¿Es un tratamiento efectivo?

Lleva a las consideraciones:

- ¿Es la muestra representativa?
- ¿Hay variables conocidas o no, observadas o no, que contaminan los resultados?
- ¿Hay un sesgo sistemático?
- ¿Existe aleatoriedad implícita o explícita?
- ¿Se intenta estimar un modelo mecanicista?

Objetivos

- Estimar y cuantificar la incertidumbre de una estimación.
- Determinar si el estimado de la población es una valor de referencia.
- Deducir una relación mecanicista cuando las cantidades se miden con ruido.
- Valorar el impacto de una política.

Herramientas de Ejemplo

Aleatorización: Balancear las variables no observadas que puedan sesgar la inferencia.

Muestreo aleatorio: Obtención de datos representativos.

Muestreo de modelos: Modelado del proceso de muestreo, usualmente iid.

Prueba de hipótesis: Toma de decisiones bajo incertidumbre.

Intervalos de confianza: Cuantificar la incertidumbre de la estimación.

Modelos de probabilidad: Relación entre los datos y la población de interés.

Diseño de estudios: Con el fin de minimizar el sesgo y la variabilidad.

Bootstrapping no paramétrico: Realizar inferencias con un modelo básico basado en los datos.

Permutación, aleatorización y prueba de intercambiabilidad: Realizar inferencias basadas en la permutación de los datos.

Enfoques de probabilidad

Probabilidad de frecuencia: proporción en la ocurrencia de eventos. Inferencia de frecuencia -> qué niveles de tolerancia me indica la proporción

de eventos ocurridos.

Probabilidad bayesiana: cálculo probabilístico de creencias, suponiendo que las creencias siguen ciertas reglas. Inferencia bayesiana

Repaso de diversos contrastes estadísticos

Teniendo en cuenta que el curso no es un curso de aprendizaje de estadística, sino más bien de aplicación de técnicas estadísticas, que se suponen en general conocidas, utilizando un determinado paquete estadístico, no vamos a explicar aquí los fundamentos de los contrastes, sino tan sólo vamos a dar algunas funciones útiles para realizar contrastes en R.

Existe en R diversas funciones para realizar contrastes estadísticos que se encuentran incorporados al lenguaje. Por esa razón, no es necesario cargar ningún paquete para utilizarlas.

R cuenta con bastantes funciones para el contraste de hipótesis, pero sólo nos vamos a fijar en algunas que implementan contrastes muy comúnmente utilizados. Las funciones que vamos a ver son las siguientes:

Test	Descripción
binom.test	Test exacto sobre el parámetro de una binomial
cor.test	Test de asociación entre muestras apareadas
wilcox.test	Test de suma de rangos de Wilcoxon para una y dos muestras
prop.test	Test de igualdad de proporciones
chisq.test	Test de la chi-cuadrado para datos de conteo
fisher.test	Test exacto de Fisher para datos de conteo
ks.test	Test de Kolmogorov-Smirnov para ajuste de datos a distribuciones dadas
shapiro.test	Test de Shapiro para comprobar ajuste de datos a una distribución normal
oneway.test	Test para comprobar la igualdad de medias entre varios grupos de datos
var.test	Test para comprobar la igualdad de varianzas entre dos grupos de datos

Se van a poner ejemplos de cada contraste, para ver qué tipo de problemas resuelven y tener una base para su aplicación.

binom.test

Lleva a cabo un contraste exacto (no aproximado) sobre el valor de la probabilidad de éxito en un experimento de Bernoulli. Por ejemplo, tiramos una moneda al aire 600 veces, y nos salen 300 caras. Y queremos contrastar la hipótesis de que la moneda está equilibrada, esto es, la probabilidad de que p=0.5. Para ello diremos cuántas veces salió cara y cuántas veces salió cruz:

```
> binom.test(c(300,300), p=0.5)

    Exact binomial test

data:  c(300, 300)
number of successes = 300, number of trials = 600,
p-value = 1
alternative hypothesis: true probability of success is not equal to 0.5
95 percent confidence interval:
 0.4592437 0.5407563
sample estimates:
probability of success
                   0.5
```

Como el p-valor ha salido 1 (es lo más alto posible), no se rechaza la hipótesis nula. Esto es, se da por bueno que la moneda estaba equilibrada. Se observa que también se obtiene un intervalo de confianza al 95 % para el verdadero valor de la probabilidad de obtener cara. Dicho intervalo es (0.459, 0.540) en nuestro ejemplo.

Otro ejemplo: tiramos la moneda al aire y obtenemos 30 caras y 100 cruces:

```
> binom.test(c(30,100), p=0.5)

    Exact binomial test

data:  c(30, 100)
number of successes = 30, number of trials = 130,
p-value = 5.421e-10
alternative hypothesis: true probability of success is not equal to 0.5
95 percent confidence interval:
 0.1614375 0.3127614
sample estimates:
probability of success
             0.2307692
```

En este caso, el p-valor es prácticamente cero, lo que quiere decir que se debe rechazar la hipótesis de que la moneda está equilibrada. Como el intervalo de confianza resulta ser (0.16, 0.31), se concluye que la probabilidad de obtener cara para la moneda utilizada está probablemente dentro de dicho intervalo.

prop.test

Con esta función se contrasta si las proporciones en varios grupos son las mismas, o bien que dichas proporciones equivalen a unas determinadas. Para empezar, se tira una moneda al aire 100 veces, y se pregunta si la proporción de caras puede ser considerada el 50 % :

Como el p-valor es alto, 0.76, no se rechaza la hipótesis nula de que p=0.5. En este caso, el test funciona como en el caso de binom.test.

Se supone ahora que se quiere saber si dos muestras han salido de la misma población. Por ejemplo, se quiere saber si las mujeres estudian un grupo de carreras diferentes en la misma proporcion. Para ello se mira la lista de matriculados en cuatro carreras y se comprueba si las proporciones de mujeres en las mismas coinciden:

```
> estudiantes <- c(100,200,50,500)
> names(estudiantes) = c("quimica","derecho","matematicas","economia")
> estudiantes
    quimica     derecho matematicas    economia
        100         200          50         500
> estudiantes.mujeres = c(60,120,10,200)
> prop.test(estudiantes.mujeres, estudiantes)

    4-sample test for equality of proportions without
    continuity correction

data:  estudiantes.mujeres out of estudiantes
X-squared = 44.5373, df = 3, p-value = 1.16e-09
alternative hypothesis: two.sided
sample estimates:
prop 1 prop 2 prop 3 prop 4
   0.6    0.6    0.2    0.4
```

La conclusion es que no. Las mujeres, en este ejemplo, se matriculan de las carreras en distinta proporcion. Es aparente que se matriculan menos en Matematicas, en este ejemplo, pero el uso del contraste nos asegura con un alto grado de seguridad que esas diferencias observadas no son debidas al azar, sino que son estructurales.

chisq.test

Este contraste se utiliza para tablas de contingencia. El caso habitual será disponer de una tabla de contingencia que cruza dos variables, contando el número de coincidencias según las categorías de ambas variables, y se trata de ver si las variables son o no son independientes.

Para poner un ejemplo, se supone que se tiene una muestra de personas en las que hemos observado dos variables: el color de ojos, que puede ser "claro" y "oscuro", y el color de pelo, que puede ser "rubio" ó "moreno". Entonces se hace una tabla de contingencia para cruzar dichas características, y se aplica el test para ver si el "color de ojos" y el "color de pelo" son variables independientes:

Se supone que los datos han sido recogidos previamente en una variable llamada 'datos':

```
> color.ojos <- sample(c("claro", "oscuro"), 50, replace = TRUE, prob = c(0.3, 0.7))
> color.pelo <- sample(c("rubio", "moreno"), 50, replace = TRUE, prob = c(0.6, 0.4))
> datos <- table(color.pelo, color.ojos)
> datos
          color.ojos
color.pelo claro oscuro
    moreno     4     16
    rubio     14     16
> chisq.test(datos)

    Pearson's Chi-squared test with Yates'
    continuity correction

data:  datos
X-squared = 2.6367, df = 1, p-value = 0.1044
```

Como el p-valor es relativamente alto (por encima de 0.05), se concluye que no se rechaza la hipótesis nula, que es que hay independencia

> entre las variables observadas: el color de ojos y el color de pelo no guardan correlación en estos datos.

cor.test / wilcox.test

Son dos tests para ver si hay asociación entre muestras de datos. Por ejemplo, una serie de alumnos pueden ser sometidos a dos test de inteligencia distintos. Si hay 100 alumnos, tendríamos dos vectores de longitud 100: el primero sería la puntuación alcanzada por los alumnos en el primer test, y el segundo la puntuación de dichos alumnos para el segundo test. Los contrastes cor.test y wilcox.test servirían para ver si hay correlación entre ambos tests.

Como se advierte al principio del tema, se supone que el fundamento del contraste se conoce. De todos modos, se recuerda que el test de Wilcoxon es no paramétrico, y tiene más en cuenta el orden correlativo de los datos que el valor de los datos mismos. Para apreciar las diferencias, pongamos unos ejemplos:

Primero se genera una secuencia de datos, y a partir de la misma, otras dos, relacionadas de alguna manera, y veremos qué dicen los tests al respecto:

```
> x <- seq(-5, 5)
> x
 [1] -5 -4 -3 -2 -1  0  1  2  3  4  5
> y <- x^2
> y
 [1] 25 16  9  4  1  0  1  4  9 16 25
```

Se ve que entre x e y hay una relación, aunque sea no lineal

```
> z <- x^3
> z
 [1] -125  -64  -27   -8   -1    0    1    8   27   64  125
```

Ahora se ve que entre x y z también hay relación

```
> cor.test(x,y)

        Pearson's product-moment correlation

data:  x and y
t = 0, df = 9, p-value = 1
alternative hypothesis: true correlation is not equal to 0
sample estimates:
cor
  0
```

La relación entre x e y no es detectada por el test cor.test: el p-valor ha salido 1.

```
> wilcox.test(x,y)

        Wilcoxon rank sum test with continuity correction

data:  x and y
W = 17.5, p-value = 0.005106
alternative hypothesis: true mu is not equal to 0

Warning message:
Cannot compute exact p-value with ties in: wilcox.test.default(x, y)
```

La relación entre x e y sí ha sido detectada por wilcox.test.

```
> cor.test(x,z)

        Pearson's product-moment correlation

data:  x and z
```

```
t = 7.1257, df = 9, p-value = 5.51e-05
alternative hypothesis: true correlation is not equal to 0
sample estimates:
     cor
0.921649
```

En este caso, cor.test sí detecta la relación, al mantenerse el orden de los datos.

fisher.test

Este test tiene el mismo fin que el test chisq.test, pero otro fundamento, y es útil para muestras pequeñas, caso para el cual chisq.test no es adecuado.

El ejemplo que trae la librería es muy adecuado. Cierta dama fina (inglesa, por supuesto), era aficionada a tomar te con leche (¡cómo no!), y presumía de ser capaz de distinguir cuando su criada le echaba primero la leche y luego el té, o bien echaba primero el té y después la leche. Tal habilidad era mirada con incredulidad por sus allegados. Para salir de dudas, Fisher le dio a probar a la dama una serie de tazas de té, en las que unas veces se había echado primero la leche y otras primero el té.

Fisher apuntó los aciertos de la dama y con la tabla de contingencia así construida, veamos lo que pasó....

```
> TeaTasting <-
  matrix(c(3, 1, 1, 3),
         nrow = 2,
         dimnames = list(Guess = c("Milk", "Tea"),
                         Truth = c("Milk", "Tea")))
> fisher.test(TeaTasting, alternative = "greater")

         Fisher's Exact Test for Count Data

data:  TeaTasting
p-value = 0.2429
alternative hypothesis: true odds ratio is greater than 1
95 percent confidence interval:
 0.3135693       Inf
sample estimates:
odds ratio
  6.408309
```

Dado el p-valor obtenido, el test no mostró evidencia suficiente a favor de la habilidad de la dama. Sin embargo, vemos que de 4 veces que se le dio primero la leche, acertó 3, y lo mismo para el té.

Otro ejemplo, con los datos de color de pelo y ojos que hemos utilizado más arriba:

```
> fisher.test(datos)

    Fisher's Exact Test for Count Data

data:  datos
p-value = 0.07425
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
 0.05741048 1.20877599
sample estimates:
odds ratio
 0.2929304
```

En este caso, no es concluyente (pero casi), como no se fue al utilizar la función chisq.test. A pesar de todo, para el tamaño de la muestra, que es pequeño, este test es más fiable.

ks.test

Se utilizará este contraste para comprobar si dos conjuntos de datos siguen la misma distribución, o bien para ver si un determinado conjunto de datos se ajusta a una distribución determinada. Las muestras no necesitan ser del mismo tamaño.

Por ejemplo, se generan 50 datos de una distribución normal, y 30 de una uniforme, a ver si el test es capaz de advertir que las distribuciones de origen no son las mismas:

```
> x <- rnorm(50)
> y <- runif(30)
> ks.test(x, y)

    Two-sample Kolmogorov-Smirnov test

data:  x and y
D = 0.58, p-value = 2.381e-06
alternative hypothesis: two-sided
```

Como el p-valor es prácticamente cero, se rechaza la hipótesis de que los datos vienen de la misma distribución.

shapiro.test

Es como ks.test, pero está especializado en la distribución normal. Por ejemplo, con los mismos datos x generados en el ejemplo anterior, se comprueba si el test detecta que provienen de una distribución normal:

```
> shapiro.test(x)

    Shapiro-Wilk normality test

data:  x
W = 0.9851, p-value = 0.7767
```

En efecto, el p-valor es grande (0.78), y no se rechaza la hipótesis nula de que los datos x provienen de una normal.

oneway.test

Este es el contraste de igualdad de medias que se estudia en los modelos ANOVA. Se supone que hay varios grupos de datos, que cada grupo sigue una distribución normal (no necesariamente con la misma varianza), y se trata de ver si las medias son todas iguales o no. Este test aparece mucho en procesos de calidad, de diseño de experimentos, en los que se fabrica un producto de varias maneras distintas, y se trata de ver si los procedimientos conducen al mismo resultado o no.

Para poner un ejemplo, se consideran los datos de fabricación de camisas de trabajo sacados del libro de Diseño y Analisis de Experimentos de Montgomery, del grupo Editorial Iberoamericana. Se trata de fabricar camisas de trabajo que tengan la mayor resistencia posible a la rotura. Para ellos se fabrican 5 tipos de camisas, donde en cada grupo se da una proporción determinada de algodón en la composición de la camisa. Las camisas del grupo 1 tienen un 15 % de algodón, las del grupo 2 un 20 %, las del grupo 3 un 25 %, las del grupo 4 un 30 %, y las del grupo 5 un 35 % de algodón. Se fabrican 5 camisas de cada tipo, se rompen y se mide su resistencia. Se trata de determinar si la resistencia media de las camisas de cada grupo es la misma o no. Vamos allá:

```
> algodon = scan()      # introducimos los datos por teclado
1: 7
2: 7
3: 15
4: 11
```

```
5: 9
6: 12
7: 17
8: 12
9: 18
10: 18
11: 14
12: 18
13: 18
14: 19
15: 19
16: 19
17: 25
18: 22
19: 19
20: 23
21: 7
22: 10
23: 11
24: 15
25: 11
26:
Read 25 items
```

Ahora se introducen el resto de los datos:

```
> porcentaje.algodon = c(15,15,15,15,15,20,20,20,20,20,25,25,25,25,25,30,30,
30,30,30,35,35,35,35,35)
> datos.algodon = cbind(algodon,porcentaje.algodon)
> datos.algodon
      algodon porcentaje.algodon
 [1,]       7                 15
 [2,]       7                 15
 [3,]      15                 15
 [4,]      11                 15
 [5,]       9                 15
 [6,]      12                 20
 [7,]      17                 20
 [8,]      12                 20
 [9,]      18                 20
[10,]      18                 20
[11,]      14                 25
[12,]      18                 25
[13,]      18                 25
[14,]      19                 25
[15,]      19                 25
[16,]      19                 30
[17,]      25                 30
[18,]      22                 30
[19,]      19                 30
[20,]      23                 30
[21,]       7                 35
[22,]      10                 35
[23,]      11                 35
[24,]      15                 35
[25,]      11                 35
```

Hasta aquí se tienen los datos de resistencia de las 25 camisas, junto con la proporción de algodón que hay en cada una. Ahora se aplica el test de igualdad de medias. En la función se especifica como parámetro algodon ~ porcen-

taje.algodon lo que significa que estudiaremos la variable algodon (su resistencia), dividiendola en grupos según el porcentaje de algodón presente.

```
> oneway.test(algodon ~ porcentaje.algodon,data=datos.algodon)

        One-way analysis of means (not assuming equal variances)

data:  algodon and porcentaje.algodon
F = 12.4507, num df = 4.000, denom df = 9.916, p-value = 0.0006987
```

Como se ve, el p-valor es prácticamente cero. Se rechaza, pues, la hipótesis de que todos los grupos son similares. Esto es, variar el porcentaje de algodón tiene consecuencias en la resistencia de la camisa. De hecho, se puede hacer un diagrama de caja (un boxplot) por grupos, para ver cómo evoluciona la resistencia en función del porcentaje de algodón:

```
> boxplot(algodon ~ porcentaje.algodon)
```

var.test

Este es un contraste para comprobar la igualdad de varianzas entre dos grupos de datos. Nos puede ser útil en multitud de situaciones, por ejemplo, cuando se tienen dos máquinas que fabrican lo mismo (por ejemplo, rodamientos), y se quiere saber si lo hacen con la misma variabilidad. Para poner un ejemplo, se generan datos de dos distribuciones normales con distinta varianza y veamos si el test detecta esta situación:

```
> x <- rnorm(50, mean = 0, sd = 2)
> y <- rnorm(30, mean = 1, sd = 1)
> var.test(x, y)

        F test to compare two variances

data:  x and y
F = 5.3488, num df = 49, denom df = 29, p-value = 7.546e-06
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
  2.687373 10.063368
sample estimates:
ratio of variances
          5.348825
```

Como se ve, el p-valor sale prácticamente cero: se rechaza la hipótesis de que los conjuntos de datos x e y provienen de distribuciones con la misma varianza.

CAPÍTULO 7

Inferencia para datos numéricos y categóricos

Datos apareados

Bootstrapping

Comparación entre dos medias

Inferencia con la distribución t

Inferencia para una proporción simple

Diferencia entre dos proporciones

Inferencia de proporciones por simulación

Comparación entre 3 o más proporciones

CAPÍTULO 8

Análisis Exploratorio de Datos

Representaciones visuales de los datos

El análisis exploratorio de datos a través de gráficos cumple con los siguientes principios:

- Principio 1: muestra comparaciones
- Principio 2: muestra causalidad, mecanismos y explicación
- Principio 3: muestra datos multivariantes
- Principio 4: integra múltiples nosod de evidencia
- Principio 5: describe y documenta la evidencia
- Principio 6: es el reino del contenido

¿Por qué se utilizan gráficos en análisis de datos?

- Para entender las propiedades de los datos.
- Para encontrar patrones en los datos.
- Para sugerir estrategias de modelado.
- Para análisis de "depuración".
- Para comunicar resultados.

Sin embargo, los **gráficos exploratorios** se usan sólo para **comunicar resultados**.

Características de los gráfcos exploratorios

- Se hacen rápido.
- Se hacen en gran cantidad.
- El propósito es el entendimiento personal.
- Los ejes y las leyendas generalmente son limpiadas (a posteriori).
- El color y el tamaño se utilizan generalmente para obtener información.

Ejemplo:

La contaminación ambiental en Estados Unidos

- La Agencia de Protección Medioambiental de Estados Unidos (EPA por sus siglas en inglés) establece estándares nacionales ambientales de la calidad del aire para la contaminación del aire exterior
  - Estándares Nacionales Ambientales de Calidad del Aire
- Para contaminción de partícula fina (PM2.5), la "media anual, premediada durante 3 años" no puede exceder $12\mu g/m^3$
- Los datos de PM2.5 diarios están disponibles desde la página web del EPA
  - Sistema de Calidad del Aire de EPA
- **Pregunta**: ¿Hay lugares en Estados Unidos que superen ese estandar nacional de contaminación de partícula fina?

Datos

Promedio anual PM2.5 acumulado desde el 2008 al 2010

```
pollution <- read.csv("data/avgpm25.csv",
                      colClasses = c("numeric", "character",
                                     "factor", "numeric", "numeric"))
                      head(pollution)
```

```
##        pm25   fips   region   longitude   latitude
##  1    9.771  01003     east      -87.75      30.59
##  2    9.994  01027     east      -85.84      33.27
##  3   10.689  01033     east      -87.73      34.73
##  4   11.337  01049     east      -85.80      34.46
##  5   12.120  01055     east      -86.03      34.02
##  6   10.828  01069     east      -85.35      31.19
```

¿Alguna localidad excede el estándar de $12\mu g/m^3$?

Resúmenes simples de datos con una dimensión

- Resumen de cinco números
- Boxplots
- Histogramas
- Gráfico de densidad
- Gráfico de barras

Sumario de cinco números

```
summary(pollution$pm25)
```

```
##  Min.  1st Qu.    Median   Mean   3rd Qu.    Max.
##  3.38     8.55     10.00   9.84    11.40    18.40
```

Boxplot

```
boxplot(pollution$pm25, col = "blue")
```

Histograma

```
hist(pollution$pm25, col = "green")
```

```
hist(pollution$pm25, col = "green")
rug(pollution$pm25
```

```
hist(pollution$pm25, col = "green", breaks = 100)
rug(pollution$pm25)
```

Características externas

```
boxplot(pollution$pm25, col = "blue")
abline(h = 12)
```

```
hist(pollution$pm25, col = "green")
abline(v = 12, lwd = 2)
abline(v = median(pollution$pm25), col = "magenta", lwd = 4)
```

Gráfico de barras

```
barplot(table(pollution$region), col = "wheat",
        main = "Number of Counties in Each Region")
```

Resumenes simples de datos con dos dimensiones

- Puntos 1-D múltiples/superpuestos (`Latice` y `ggplot2`)
- Diagramas de dispersión
- Diagramas de dispersión suavizados

Más de 2 dimensiones

- Gráficos 2-D múltiples/superpuestos; coplots
- Uso de color, tamaño, y sobertura para añadir dimensiones
- Graficos Spinning
- Gráficos reales 3D (no muy útiles)

Boxplots múltiples

```
boxplot(pm25 ~ region, data = pollution, col = "red")
```

Múltiples histogramas

```
par(mfrow = c(2, 1), mar = c(4, 4, 2, 1))
hist(subset(pollution, region == "east")$pm25, col = "green")
hist(subset(pollution, region == "west")$pm25, col = "green")
```

Diagrama de dispersión

```
with(pollution, plot(latitude, pm25))
abline(h = 12, lwd = 2, lty = 2)
```

Usando color

```
with(pollution, plot(latitude, pm25, col = region))
abline(h = 12, lwd = 2, lty = 2)
```

Diagramas de dispresión múltiple

```
par(mfrow = c(1, 2), mar = c(5, 4, 2, 1))
with(subset(pollution, region == "west"), plot(latitude, pm25, main = "West"))
with(subset(pollution, region == "east"), plot(latitude, pm25, main = "East"))
```

Resumen

- Los gráficos exploratorios son "rápidos y sucios"
- Permiten resumir los datos (a menudo de forma gráfica) y destacar cualquier característica común
- Explora preguntas básicas e hipótesis (y quizás las descarte)
- Sugiere estrategias de modelado para los siguientes pasos

`lattice`

Utilizando `lattice`:

- Los gráficos se crean con un simple llamado a la función (`xyplot`, `bwplot`, etc.)
- Son muy útiles para tipos de gráficos condicionados: observar como y cambia con x a través de niveles de z
- Elementos como márgenes/espaciado se establecen automáticamente ya que todo el gráfico se especifica al principio.
- Es muy útil para colocar muchos, muchos gráficos juntos en una pantalla
- Muchas veces resulta poco útil para especificar un gráfico completo en una única llamada de función
- Los comentarios y anotaciones en el gráfico no son intuitivos
- El uso de las funciones del panel y los subscripts se dificulta y requiere una preparación intensa.
- No se puede añadir al gráfico una vez que se ha creado.

ggplot2

- Divide la diferencia entre `base` y `lattice`
- Trabaja de modo automático con espaciado, texto, títulos, y también permite hacer anotaciones a través de `adding`
- Tiene una similaridad superficial con el `lattice` pero en general es mucho más intuitivo y sencillo de utlizar.
- El modo por defecto tiene muchas opciones disponibles (¡que pueden configurarse!)
- Es una implementación de *Grammar of Graphics* de Leland Wilkinson
- Fue escrita por Hadley Wickham (mientras era estudiante en la Universidad de Iowa)
- Está disponible desde CRAN a través de `install.packages()` (http://ggplot2.org)
- Es pensar los gráficos en términos de "verbo", "sustantivo", "adjetivo"

Elementos Básicos

qplot()

- Trabaja muy parecido a la función `plot` en sistemas gráficos base
- Busca datos en un conjunto de datos, tal como `lattice`, o el entorno inmediato
- Los gráficos se hacen con *estética* (tamaño, color, forma) y *geometría* (puntos, líneas)
- Aquellos elementos que son importates para indicar subconjuntos de datos deben ser **etiquetados** (si tienen propiedades distintas)
- `qplot()` oculta lo que ocurre en el interior, lo cual está bien para la mayoría de las operaciones
- `ggplot()` es la función central y es mucho más flexible para hacer cosas que no se pueden hacer con `qplot()`

Ejemplo

```
library(ggplot2)
str(mpg)
```

```
'data.frame':   234 obs. of  11 variables:
 $ manufacturer: Factor w/ 15 levels "audi","chevrolet",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ model       : Factor w/ 38 levels "4runner 4wd",..: 2 2 2 2 2 2 2 3 3 3 ...
 $ displ       : num  1.8 1.8 2 2 2.8 2.8 3.1 1.8 1.8 2 ...
 $ year        : int  1999 1999 2008 2008 1999 1999 2008 1999 1999 2008 ...
 $ cyl         : int  4 4 4 4 6 6 6 4 4 4 ...
 $ trans       : Factor w/ 10 levels "auto(av)","auto(l3)",..: 4 9 10 1 4 9 1 9 4 10 ...
 $ drv         : Factor w/ 3 levels "4","f","r": 2 2 2 2 2 2 2 1 1 1 ...
 $ cty         : int  18 21 20 21 16 18 18 18 16 20 ...
 $ hwy         : int  29 29 31 30 26 26 27 26 25 28 ...
 $ fl          : Factor w/ 5 levels "c","d","e","p",..: 4 4 4 4 4 4 4 4 4 4 ...
 $ class       : Factor w/ 7 levels "2seater","compact",..: 2 2 2 2 2 2 2 2 2 2 ...
```

`ggplot2` "Hello, world!"

```
qplot(displ, hwy, data = mpg)
```

Modificando la estética

```
qplot(displ, hwy, data = mpg, color = drv)
```

Añadiendo geometría

```
qplot(displ, hwy, data = mpg, geom = c("point", "smooth"))
```

Histogramas

```
qplot(hwy, data = mpg, fill = drv)
```

Facetas

```
qplot(displ, hwy, data = mpg, facets = . ~ drv)
qplot(hwy, data = mpg, facets = drv ~ ., binwidth = 2)
```

Creación de gráficos analíticos

Resúmenes exploratorios de datos

Visualización multidimensional exploratoria

El análisis multivariante está relacionado con conjuntos de datos que involucran más de una variable para cada unidad de observación. En general tales conjuntos de datos vienen representados por matrices de datos X con n filas y p columnas, donde las filas representan los casos, y las columnas las variables. En cuanto al análisis de tales matrices, unas veces estaremos interesados en estudiar las filas (casos), otras veces las relaciones entre las variables (las columnas) y algunas veces en estudiar simultáneamente ambas cosas.

Graficos sin hacer transformaciones

El primer comando que tenemos disponible para la tarea de explorar datos multivariantes es:

```
> pairs()
```

Vamos a ver cómo se puede utilizar para el conjunto de datos iris, que contiene una base de 4 medidas tomadas sobre una muestra de 150 flores ornamentales (lirios). Las medidas tomadas son la longitud y anchura de pétalos y sépalos. En el problema original, se trataba de clasificar las flores en tres especies predeterminadas, utilizando las cuatro medidas mencionadas.

```
> data(iris) # Se carga el conjunto de datos iris
> iris[1:4,] # Se muestran 4 registros
Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
> pairs(iris[,1:4]) # Se muestran los gráficos de dispersion de las variables iris
```

El equivalente (mejorado) de este gráfico con el paquete `ggplot2` se realizaba con la función `plot.matrix()`. Esta función ha quedado obsoleta como indica el siguiente mensaje:

```
> library(ggplot2)
> plotmatrix(iris[, 1:4])
Error: Please use GGally::ggpairs. (Defunct; last used in version 0.9.2)
```

De manera que, tal como nos indican, se instala el paquete `GGally` y se utiliza la función `ggpairs()`

```
> library(GGally)
> ggpairs(iris[, 1:4])
```

Ahora se van a dibujar los datos etiquetados por especie. Asi se podra ver si las flores de una misma especie aparecen agrupadas. Las flores que aparezcan entre dos grupos corresponderán a flores difíciles de clasificar claramente en uno de los grupos.

Debido a que las etiquetas son bastante largas ("setosa", "virginica", "versicolor"), se sustituyen por letras, para facilitar la visualización. Se usará la "s" para setosa, la "v" para virginica, y la "c" para versicolor:

```
> iris.especies = c(rep("s",50),rep("c",50),rep("v",50))
```

Para dibujar incluyendo las etiquetas, podemos proceder asi:

```
> pairs(iris[,1:4], panel=function(x,y,...) text(x,y,iris.especies))
```

El comando anterior dibujará las coordenadas de los datos, 2 a 2, etiquetando con la clase de cada flor.

Un modo alternativo de proceder es generarnos nosotros mismos gráficos similares. Podríamos hacer (entre otras posibilidades), algo así como:

```
> par(mfrow=c(1,2))
> plot(iris[,1], iris[,2])
> points(iris[iris.especies=="s", 1], iris[iris.especies=="s", 2], col=2)
> points(iris[iris.especies=="c", 1], iris[iris.especies=="c", 2], col=3)
> points(iris[iris.especies=="v", 1], iris[iris.especies=="v", 2], col=4)
> plot(iris[,1], iris[, 3])
> points(iris[iris.especies=="s", 1], iris[iris.especies=="s", 3], col=2)
> points(iris[iris.especies=="c", 1], iris[iris.especies=="c", 3], col=3)
> points(iris[iris.especies=="v", 1], iris[iris.especies=="v", 3], col=4)
```

La primera orden hace que la pantalla de dibujo quede dividida como una matrix 1 x 2, esto es, una matriz con una fila y dos columnas, por tanto, dos graficos en uno de esa manera. La segunda orden dibuja en el primer gráfico las coordenadas 1 y 2 de los datos. La tercera orden y siguientes van repintando los puntos que corresponden a cada especie. Obsérvese que al dar cada orden lo que sucede. El segundo bloque de ordenes pinta la coordenada 1 frente a la dos y hace lo mismo.

Graficos transformando los datos

Cuando los datos tienen más de 3 dimensiones, dibujar las coordenadas 2 a 2 puede ser interesante, pero cuando hay muchas, puede ser más interesante hacer una proyección de los datos usando algún tipo de transformación lineal o no linea, que nos permita resumir en un gráfico de dos dimensiones las características más importantes de los mismos en cuanto a variabilidad, en cuanto a conservación de distancias en la proyección, o en cuanto a la representación gráfica de los clusters hallados en el conjunto de datos.

A continuación pasamos a exponer las diversas técnicas, junto con los comandos en R que las implementan.

Análisis de componentes principales

Esta técnica funciona construyendo automáticamente un conjunto de combinaciones lineales de las variables del conjunto de datos, con la máxima variabilidad posible. Cada componente principal viene determinada por un vector propio de la matriz de covarianzas o correlaciones de los datos, esto es, un vector propio de la matriz $X^T X$.

Geométricamente, el primer vector propio de la matriz corresponde a la dirección de máxima variabilidad en el conjunto de datos, que es la dirección sobre la que se proyectará. Por ejemplo, si tenemos datos sobre una elipse alargada, y hacemos un análisis de componentes principales con una sola componente, la coordenada que representará nuestro conjunto de datos corresponderá a la proyección de los datos sobre el eje principal de la elipse.

Se puede encontrar información más detallada de la teoría en cualquier libro de análisis multivariante, y en la red, podéis acudir a los siguientes links, por ejemplo:

http://cran.r-project.org/doc/vignettes/HSAUR/Ch_principal_components_analysis.pdf (Análisis de componentes principales, Proyecto R).

http://csnet.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf (Lindsay I. Smith, Universidad de Otago, Nueva Zelanda).

Para usar componentes principales hay que cargar una librería de las varias que hay con R para análisis multivariante (mva, multiv y otras). Por ejemplo:

```
> data(iris)   # Se cargan los datos de las flores
> iris.pca = prcomp(iris[,1:4])  #  Se reliza el análisis de componentes principales
> attributes(iris.pca)        # Se observa qué devuelve el análisis
$names
[1] "sdev"     "rotation" "x"

$class
[1] "prcomp"
```

Ahora dibujamos las dos primeras componentes principales y etiquetamos:

```
> plot(iris.pca$x[,1:2],type="n")
> text(iris.pca$x[iris$Species=="setosa",1:2],col=2,"s")
> text(iris.pca$x[iris$Species=="virginica",1:2],col=3,"v")
> text(iris.pca$x[iris$Species=="versicolor",1:2],col=4,"c")
```

La estructura `iris.pca$x` contiene las coordenadas de los puntos, luego al pedir que dibuja las `1:2` estamos pidiendo que dibuje las dos primeras componentes. El `type="n"` hace que dibuje pero que no se vea, esto es, marca las dos primeras coordenadas. Las demás órdenes dibujan los puntos por clases con un color distinto para cada clase.

Como se puede apreciar en el gráfico, hemos conseguido representar un conjunto de dimensión 4 por un conjunto de dimensión 2, y si se compara con los gráficos ya hechos de los datos iris, puede verse que la estructura se ha respetado bastante fielmente. Puede concluirse que hay 3 grupos, uno bien separado y otros dos más juntos. Y en efecto, así es.

La técnicas de componentes principales se pueden emplear para cualquier conjunto de datos del cual se quiera tener una idea de la estructura. También puede utilizarse para codificar información, en el sentido de sustituir la codificación original de ciertos datos por otra que ocupe menos espacio. Así por ejemplo, si se dispone de imágenes de fotos (en blanco y negro, representadas por niveles de gris), puede utilizarse la técnica de componentes principales para reducir la dimensión de las imágenes. Se puede encontrar información más detallada a este respecto en:

http://www.willamette.edu/~gorr/classes/cs449/Unsupervised/pca.html (G. B. Orr, Willamette University, USA).

Para saber cúantas componentes son suficientes para representar fidedignamente un conjunto de datos (por ejemplo, si el conjunto tiene dimensión 10, deberemos usar 2, 3, ¿cuántas componentes?) se miran los valores propios asociados a la descomposición comentada más arriba. Tales valores se guardan en "sdev", de manera que en el ejemplo del iris:

```
> iris.pca$sdev
[1] 2.0562689 0.4926162 0.2796596 0.1543862
```

Se observa que los valores se dan de mayor a menor. Tales valores corresponden a variabilidades a lo largo de los ejes de proyección (esto es, las componentes principales). Para hacernos una idea de su importancia relativa, iremos calculando la suma acumulada, dividida por la suma total para que resulte 1:

```
> cumsum(iris.pca$sdev^2)/sum(iris.pca$sdev^2)
[1] 0.9246187 0.9776852 0.9947878 1.0000000
```

Como seve, con 1 componente se recoge aproximadamente el 92 % de la variabilidad de los datos, lo que no es poco a efectos de visualización. También se ve que con 3 componentes ya se explica el 99 % de la variabilidad. Esto es, una componente no aporta casi nada de información a efectos de explicar la estructura de los datos. La razón de tomar los cuadrados es porque la varianza es el cuadrado de la desviación típica.

Escalado multidimensional

La idea del escalado multidimensional es la misma que la del análisis de componentes principales: Se trata de representar los datos en baja dimensión (usualmente 2), pero en este caso, en vez de utilizar las coordenadas de los datos para llevar a cabo la transformación, se usará la información proporcionada por las distancias entre los datos.

Esta manera de proceder tiene mucho sentido para datos cuyas coordenadas no se conocen, pero de los cuales conocemos las distancias entre ellos. Por ejemplo, en un mapa de carreteras vienen las distancias en kilometros entre diversas ciudades españolas. Esta técnica permite reconstruir las coordenadas de los datos, conociendo las distancias entre los mismos.

Podéis consultar ejemplos y la teoría más detallada en:

http://www.analytictech.com/networks/mds.htm (Analytic Technologies, USA)

Otro caso en el que el escalado multidimensional resulta útil se da cuando la información viene dada por matrices de preferencias. Este es el caso cuando se pide a un grupo de alumnos que asignen preferencias al resto de sus compañeros.

Como primer ejemplo, consideraremos el que viene con la función `cmdscale()`.

```
> data(eurodist)     # Se cargan los datos en memoria
> attributes(eurodist)
$Size
[1] 21

$Labels
[1] "Athens"          "Barcelona"       "Brussels"        "Calais"
[5] "Cherbourg"       "Cologne"         "Copenhagen"      "Geneva"
[9] "Gibralta"        "Hamburg"         "Hook of Holland" "Lisbon"
[13] "Lyons"           "Madrid"          "Marseilles"      "Milan"
[17] "Munich"          "Paris"           "Rome"            "Stockholm"
[21] "Vienna"

$class
[1] "dist"

> eurodist.mds = cmdscale(eurodist)

> x = eurodist.mds[,1]
> y = eurodist.mds[,2]
> plot(x, y, type="n", xlab="", ylab="")

> eurodist2 = as.matrix(eurodist)
> text(x, y, dimnames(eurodist2)[[1]], cex=0.5)
```

Asi pues, `eurodist` es un objeto que guarda los nombres de las distancias, la matriz de distancias entre las ciudades, asi como el numero de objetos, que se podría recuperar escribiendo:

```
> attr(eurodist,"Size")
[1] 21
```

El proceso de escalado se realiza con la función cmdscale y después se dibuja el mapa de las ciudades con sus nombres. Podéis repetir el proceso con algunas ciudades españolas. El mapa que obtendréis puede aparecer rotado o invertido, dado que el método garantiza la conservación de las distancias, pero eso no garantiza que la orientación en el plano sea la correcta.

A continuación vamos a dibujar un mapa de preferencias entre un grupo de 8 personas. Cada persona asigna 1 a las personas de su preferencia. Dicho mapa reflejará grupitos de amigos. Obsérvese que en la diagonal hay unos, asumiendo que uno es amigo de sí mismo.

Los datos se pueden encontrar aquí:

amigos.txt

La matriz de datos presenta el siguiente aspecto:

```
X        paco pepe lola juan carlos luis ana pedro
paco        1    1    0    0      0    0   0     1
pepe        1    1    0    0      0    1   0     1
lola        0    0    1    1      0    0   1     0
juan        0    0    1    1      0    0   1     1
carlos      0    1    0    0      1    0   0     1
luis        1    0    1    0      1    1   0     0
ana         1    1    0    0      0    0   1     0
pedro       1    1    0    0      0    0   0     1
```

Para leer dicha matriz de un archivo, se puede leer directamente, como hago yo a continuación, o de otra forma que prefiráis. El directorio que pongo yo en la orden scan es el mío. En vuestro caso, podrá variar.

```
> amigos.txt = scan("datos//amigos.txt")
Read 64 items
> amigos.txt = structure(amigos.txt,dim=c(8,8),byrow=T)
> amigos.txt = t(amigos.txt)
> amigos.txt
[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    1    1    0    0    0    0    0    1
[2,]    1    1    0    0    0    1    0    1
[3,]    0    0    1    1    0    0    1    0
[4,]    0    0    1    1    0    0    1    1
[5,]    0    1    0    0    1    0    0    1
[6,]    1    0    1    0    1    1    0    0
[7,]    1    1    0    0    0    0    1    0
[8,]    1    1    0    0    0    0    0    1
attr(,"byrow")
[1] TRUE
> rownames(amigos.txt)<-c("paco","pepe","lola","juan","carlos","luis","ana","pedro")
> colnames(amigos.txt)<-c("paco","pepe","lola","juan","carlos","luis","ana","pedro")
> amigos.txt
> amigos.mds <- cmdscale(amigos.txt)
> plot(amigos.mds[,1],amigos.mds[,2],type="n")
> text(amigos.mds[,1],amigos.mds[,2],labels=rownames(amigos.txt))
```

El comando rownames asigna nombres a las filas, y colnames hace lo propio con las columnas. El gráfico final muestra una imagen bidimensional de las preferencias de esas personas

Análisis de cluster

El *análisis de cluster*, denominado en español a veces *análisis de conglomerados*, es una técnica no supervisada para descubrir la estructura de datos de un conjunto. Entra en el apartado de técnicas gráficas para datos multivariantes por el siguiente motivo: Cuando unos datos son de alta dimensión, es bastante dificil descubrir cuántos grupos hay en el conjunto de datos, y qué dato pertenece a cada grupo, en su caso. El análisis de cluster engloba una serie de técnicas para dividir el conjunto de datos en una serie de grupos, y frecuentemente se apoya en métodos de visualización propios para la visualización de tales grupos.

La información que se usa para llevar a cabo la mayoría de los procedimientos de análisis de cluster es la proporcionada por las distancias entre los datos, al igual que en el caso del escalado. Aunque hay varias maneras de llevar a cabo la construcción de los grupos, hay dos grandes grupos de técnicas: las técnicas jerárquicas y las técnicas no jerárquicas.

Las técnicas jerárquicas proceden por niveles a la hora de construir los grupos. En un caso se procede de abajo hacia arriba, considerando tantos grupos como datos hay en el conjunto, y agrupando sucesivamente hasta que solo queda un grupo. La visualización del procedimiento utilizando una herramienta gráfica denominada dendrograma, ayudará a descubrir la estructura de grupos en los datos. Para ir agrupando los datos, primero se buscan los datos más cercanos entre sí. En el segundo nivel se agrupan los grupos más cercanos entre si, y asi sucesivamente. El otro caso frecuente es partir de un sólo grupo e ir partiendolo en grupos más pequeños hasta llegar al nivel de los datos individuales. Para todas estas técnicas jerárquicas hay librerías en R. Aquí Se utilizará la función `hclust()`.

Para ilustrar el funcionamiento, vamos a utilizar una base de animales descritos por algunas características. Los datos están en:

```
animal.txt
animal.des
anima2.des
```

Se leen los datos:

```
> animal = scan(""datos//Animal.txt")
Read 208 items
> animal = structure(animal,dim=c(13,16),byrow=T)
```

Hasta aqui se tiene la matrix animal como una matriz de 16 animales por 13 características. Ahora se leen los nombres de los animales y de las características:

```
animal.nombres <-
c("paloma","gallina","pato","ganso","lechuza","halcon",
  "aguila","zorro","perro","lobo","gato","tigre","leon",
  "caballo","cebra","vaca")

> animal.nombres
[1] "paloma"  "gallina" "pato"    "ganso"   "lechuza" "halcon"  "aguila"  "zorro"
[9] "perro"   "lobo"    "gato"    "tigre"   "leon"    "caballo" "cebra"   "vaca"

animal.carac <-
c("peque","mediano","grande","bipedo","cuadrupedo","pelo","pezunas",
  "melena","plumas","caza","corre","vuela","nada")
```

Se asignan nombres a filas y columnas:

```
> dimnames(animal) <- list(animal.carac,animal.nombres)
```

Y por ultimo, echamos un vistazo a una parte de la matriz:

```
> animal[1:5,1:10]
        peque mediano grande bipedo cuadrupedo pelo pezuñas melena plumas caza
paloma      1       0      0      1          0    0       0      0      1    0
gallina     1       0      0      1          0    0       0      0      1    0
pato        1       0      0      1          0    0       0      0      1    0
ganso       1       0      0      1          0    0       0      0      1    0
lechuza     1       0      0      1          0    0       0      0      1    1
```

Se puede revisar toda la matriz, y ver que las descripciones son correctas, aunque algo simples, por supuesto.

Ahora el objetivo en un análisis de cluster sería visualizar de alguna manera la estructura de los datos, y descubrir qué grupos de animales pueden existir.

Con este fin se calcula la matriz de distancias entre los animales, descritos por sus características, se realiza el análisis de cluster sobre los grupos usando la función hclust, y después se visualizan los grupos utilizando el dendrograma:

```
> animal.dist <- dist(animal)
> animal.hclust <- hclust(animal.dist)
> plot(animal.hclust)
```

Como se puede ver al repetir estos comandos, salen dos grandes grupos, uno con las aves, y otro con los mamiferos. Dentro de las aves, se verá un grupo formado por las rapaces y otro por las demás, y dentro de los mamíferos, una distinción entre mamíferos grandes y menos grandes, distinguiendo los cazadores de los que no lo son. ¿Qué tal queda?

Esta técnica aplicada a datos menos evidentes, puede ayudarnos a descubrir grupos interesantes en los datos bajo estudio.

CAPÍTULO 9

Modelos de Regresión

Conceptos de regresión basados en R

El análisis de regresión se utiliza para explicar una determinada variable, digamos Y, en función de una variable X, o bien en función de varias variables $X_1, X_2, ..., X_k$. En el primer caso se tratará de regresión univariante, y en el segundo caso, de regresión multivariante. El modelo de explicación en ambos casos es lineal, esto es, se asume que la dependencia entre Y y las variable explicativa X adopta la forma:

$$Y = a + bX + error$$

O, en el caso multivariante:

$$Y = a + b_1X_1 + b_2X_2 + ... + b_kX_k + error$$

El término de error aparece porque cada vez que observamos una X, no siempre observaremos la misma Y.

Por ejemplo si X es la estatura de una persona, e Y el peso, cada vez que observemos una estatura, no siempre obtendremos el mismo peso en Y.

Los que hayan estudiado estadística, conocen el modelo perfectamente. Los que no, simplemente deben saber que el modelo es útil para predecir relaciones lineales, y para un estudio más profundo, cualquier libro de estadística con un capítulo de regresión sirve.

En caso de querer leer un buen libro on-line con muchos ejemplos programados en R, se puede acudir a:

http://cran.r-project.org/doc/contrib/Faraway-PRA.pdf (Proyecto R, Julian Faraway, Bath University, Reino Unido).

Regresión lineal simple

A continuación vamos a poner un ejemplo de regresión simple, realizado en R.

El archivo `sueldos.txt` contiene la antigüedad (en años) y el sueldo (en millones) de una serie de directivos de una empresa. Se trata de determinar si existe relación entre el sueldo de un directivo de dicha empresa y su antigüedad en la misma, y si es así, de estimarla.

En primer lugar, cargamos los datos y ponemos nombres a las variables:

```
> sueldos <- scan("datos//sueldos.txt") # variará según donde se guarden los datos
Read 40 items
> sueldos <- structure(sueldos, dim=c(2,20), byrow=T)
> sueldos <- t(sueldos)
> colnames(sueldos) = c("antiguedad","sueldo")
```

```
> sueldos
      antiguedad sueldo
 [1,]          0   4.61
 [2,]          2   6.97
 [3,]          2   6.36
 [4,]          2   6.61
 [5,]          1   3.61
 [6,]          5  10.15
 [7,]          0   4.00
 [8,]          3   8.63
 [9,]          3   9.34
[10,]          0   3.86
[11,]          5  12.62
[12,]          4   9.42
[13,]          3   7.63
[14,]          4   9.97
[15,]          2   6.33
[16,]          0   3.19
[17,]          1   5.62
[18,]          2   7.98
[19,]          4  10.49
[20,]          4   8.54
attr(,"byrow")
[1] TRUE
```

El primer paso será observar el aspecto de los datos, para ver si es correcto intentar ajustar una recta a los mismos:

```
> plot(sueldos)
```

Una vez hecho eso, para ajustar la recta, se usa la función *lm()*. `lm` significa "linear model", y su uso es muy sencillo:

Se convierten los sueldos a data frame para usar con comodidad los nombres de las variables:

```
> sueldos <- data.frame(sueldos)  # Convierte la matriz sueldos a data frame
> attach(sueldos) # Permite que las variables de "sueldos" se puedan llamar directamente
> sueldos.lm <- lm(sueldo ~ antiguedad)  # Realiza la regresión
```

Si queremos dibujar la recta de regresión haremos:

```
> abline(sueldos.lm)
```

Si queremos ver qué se guarda exactamente en el objeto de regresión `sueldos.lm` :

```
> attributes(sueldos.lm)
$names
[1] "coefficients"  "residuals"     "effects"       "rank"
[5] "fitted.values" "assign"        "qr"            "df.residual"
[9] "xlevels"       "call"          "terms"         "model"

$class
[1] "lm"
```

Los que hayan estudiado regresión con anterioridad reconocerán varios de los nombres de la estructura anterior. Para explicar los más importantes, se procede a visualizar un sumario del proceso de regresión:

```
> summary(sueldos.lm)

Call:
lm(formula = sueldo ~ antiguedad)
```

```
Residuals:
Min      1Q  Median      3Q     Max
-1.6538 -0.4717  0.1455  0.4444  1.3334

Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept)   3.7581     0.3324   11.31 1.31e-09 ***
antiguedad    1.5057     0.1164   12.93 1.50e-10 ***
---
Signif. codes:  0  `***'  0.001  `**'  0.01  `*'  0.05  `.'  0.1  ` '  1

Residual standard error: 0.8441 on 18 degrees of freedom
Multiple R-Squared: 0.9028,     Adjusted R-squared: 0.8974
F-statistic: 167.2 on 1 and 18 degrees of freedom, p-value: 1.502e-010
```

Lo primero es localizar la ecuación de la recta de regresión. Esta viene dada por los coeficientes, que en este caso son 3.7581 y 1.5057, lo que quiere decir que la recta de regresión viene dada por:

$$Sueldo = 1{,}5057 + 3{,}7581 Antiguedad$$

Naturalmente para cada dato concreto se comete un error. Dichos errores son los residuos. Si se quiere ver para los datos de nuestro problema, se escribe:

```
> sueldos.lm$residuals
1          2          3          4          5          6          7
0.8518934  0.2004948 -0.4095052 -0.1595052 -1.6538059 -1.1366032  0.2418934
8          9         10         11         12         13         14
0.3547954  1.0647954  0.1018934  1.3333968 -0.3609039 -0.6452046  0.1890961
15         16         17         18         19         20
-0.4395052 -0.5681066  0.3561941  1.2104948  0.7090961 -1.2409039
```

Si se quiere dibujar:

```
> plot(sueldos.lm$residuals)
> abline(h=0)
```

Dibujamos una línea horizontal en $y = 0$ porque los residuos cumplen la propiedad de estar centrados alrededor de dicha linea. Si en el gráfico se observa algún dato que se aleja mucho por arriba o por abajo, eso quiere decir que para ese dato, el modelo de regresión no predijo bien, dado que su residuo es elevado. Eso quiere decir que tal dato no es bien explicado por el modelo, y así tenemos una forma de detectar tal situación.

Por ejemplo, algo así podría pasar para el sueldo del director, tal vez.

Los valores predichos para los datos observados son los fitted values:

```
> sueldos.lm$fitted.values
1         2         3         4         5         6         7         8
3.758107  6.769505  6.769505  6.769505  5.263806 11.286603  3.758107  8.275205
9        10        11        12        13        14        15        16
8.275205  3.758107 11.286603  9.780904  8.275205  9.780904  6.769505  3.758107
17        18        19        20
5.263806  6.769505  9.780904  9.780904
```

La lista anterior muestra el sueldo predicho para cada individuo por el modelo de regresión ajustado.

Hay otras características de interés en el modelo, de naturaleza estadística. Por ejemplo, el valor `R-Squared` mide la variabilidad de los datos explicada por el modelo. En el ejemplo anterior es aproximadamente 0.90 lo que quiere decir que más del 90 % de la variabilidad de los datos fue recogida por el modelo: esto es, es un buen modelo.

En cuanto al uso del modelo para predecir, si se quiere predecir el sueldo de un directivo recién entrado en la empresa (0 años de antigüedad), se escribiría:

```
> predict.lm(sueldos.lm, data.frame(antiguedad=0))
[1] 3.758107
```

Luego la predicción es 3.75 u.m. Si se quiere predecir los sueldos esperables de directivos que lleven año y medio, 2 y 3.5 años, se escribiría:

```
> predict.lm(sueldos.lm, data.frame(antiguedad=c(1.5,2,3.5)))
       1        2        3
6.016656 6.769505 9.028054
```

Regresión lineal múltiple

Funciona de forma similar al modelo de regresión lineal simple, con la diferencia de que lo que se estima es un plano de regresión.

Vamos a cargar unos datos de R de nombre `cars`:

```
> data(mtcars)
> attach(mtcars)
```

Mostramos los primeros registros:

```
> mtcars[1:5,]
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
```

Vamos a explicar el consumo `mpg` en función de la potencia `hp` y del peso `wt`:

```
> cars.lm = lm(mpg ~ hp + wt)
```

Observemos que mpg es la variable que se explica, y el signo más $+$ indica sólo yuxtaposición, esto es, que las variables que explican con hp y wt:

Se observa el resultado:

```
> summary(cars.lm)

Call:
lm(formula = mpg ~ hp + wt)

Residuals:
Min      1Q Median      3Q     Max
-3.941 -1.600 -0.182  1.050  5.854

Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) 37.22727    1.59879  23.285  < 2e-16 ***
hp          -0.03177    0.00903  -3.519  0.00145 **
wt          -3.87783    0.63273  -6.129 1.12e-06 ***
---
Signif. codes:  0  `***'  0.001  `**'  0.01  `*'  0.05  `.'  0.1  ` '  1

Residual standard error: 2.593 on 29 degrees of freedom
```

```
Multiple R-Squared: 0.8268,     Adjusted R-squared: 0.8148
F-statistic: 69.21 on 2 and 29 degrees of freedom,      p-value: 9.109e-012
```

Por tanto, el modelo es:

$$millasrecorridasporgaln = 37{,}22 - 0{,}03potencia - 3{,}87peso$$

Esto es, cuanto más potente es el carro, menos millas recorre (de ahí el signo negativo de su coeficiente), y cuanto más pesa, menos millas recorre.

El `R-squared` es del 83 %, lo que quiere decir que esas dos variables explican bastante bien el consumo.

Se pueden dibujar los residuos para ver si hay algún carro que se comporta de modo muy distinto a los demás:

```
> plot(cars.lm$residuals)
> abline(h=0)
```

Si se quiere predecir las millas recorridas por galón por un carro con 150 caballos y peso 2.5:

```
> predict.lm(cars.lm, data.frame(hp=150, wt=2.5))
[1] 22.76675
```

Relación entre dos variables numéricas

Regresión lineal con un predictor único

Inferencias mediante regresión lineal

Regresión con múltiples predictores

Selección de modelos

CAPÍTULO 10

Investigación Reproducible

Estructura y organización de un análisis de datos

Archivos de un análisis de datos

- Datos
  - Datos "crudos"
  - Datos procesados
- Figuras
  - Figuras exploratorias
  - Figuras definitivas
- Código R
  - Scripts crudo / no utilizados
  - Scripts definitivos
  - Archivos R Markdown
- Texto
  - Archivos LEÁME (README)
  - Texto de análisis / reporte

Datos crudos

- Deben estar almacenados en la carpeta de análisis
- Si se acceden desde la web, deben incluir el URL, la descripción y las fechas de acceso en el LEÁME

Datos procesados

- Los datos procesados deben tener nombres tales que sea fácil saber con cual script fueron generados.
- En el LEÁME debe indicarse la relación entre el script de procesamiento y los datos procesados.
- Los datos procesados deben ser ordenados

Figuras exploratorias

- Las figuras hechas durante los análisis, que no son necesariamente parte del reporte final.
- No necesitan ser "bonitas".

Figuras finales

- Usualmente un pequeño conjunto de las figuras originales
- Ejes/colores configurados para hacer la figura clara
- Posiblemente múltiples paneles

Scripts crudos

- Pueden estar menos comentados (¡pero los comentarios le ayudarán!)
- Pueden haber múltiples versiones
- Pueden incluir análisis que serán descartados

Scripts definitivos

- Claramente comentados
  - Pequeños comentarios con generosidad - qué, cuándo, porqué, cómo
  - Los bloques de comentarios mas grandes para secciones completas
- Incluyen detalles de procesamiento
- Solamente análisis que aparecen en el escrito final

Archivos R markdown

- Los archivos R markdown pueden usarse para generar reportes reproducibles
- Integra texto y código R
- Son fáciles de crear en RStudio

Archivos LEÁME

- No es necesario si utiliza R markdown
- Debe incluir instrucciones paso a paso para el análisis
- Aquí hay un ejemplo https://github.com/jtleek/swfdr/blob/master/README

Texto del documento

- Debe incluir: título, introducción (motivación), métodos (estadísticos que utilizó), resultados (incluyendo medidas de incertidumbre), y conclusiones (incluyendo problemas potenciales).
- Debe contar una historia
- *No debe incluir todos los análisis que realizó*
- Deben incluirse las referencias para los métodos estadísticos

Recursos adicionales

- Información sobre un estudio no reproducible que ocasionó que pacientes de cáncer no fueran atendidos adecuadamente The Duke Saga Starter Set
- Investigación reproducible y bioestadística
- Buenas prácticas para el manejo de proyectos de análisis estadístico
- Plantilla de proyecto un conjunto de archivos pre-organizados para analisis de datos.

Generación de documentos utilizando R markdown

¿Qué es Markdown?

- Creado por John Gruber y Aaron Swartz.
- Una versión simplificada de los lenguajes de "marcado"
- Permite enfocarse en escribir en lugar de formatear
- Elementos de formato simples/mínimos
- Se pueden convertir fácilmente a HTML (y otros formatos) utilizando las herramientas existentes.
- Complete information is available at http://daringfireball.net/projects/markdown/
- Some background information at http://daringfireball.net/2004/03/dive_into_markdown

¿Qué es R Markdown?

- R markdown es la integración de código R con markdown
- Permite crear documentos que contienen código R "vivo"
- El código R es evaluado como parte del procesamiento de markdown
- Los resultado del código R se insertan en el documento markdown
- Un herramienta fundamental de la **programación estadística literal**

¿Qué es R Markdown? (cont.)

- R markdown puede convertirse en markdown estándar utilizando el paquete knitr.
- Markdown puede convertirse a HTML utilizando el paquete markdown de R.
- Cualquier editor básico de txto puede utilizarse para crear un dcumento markdown; no se requieren heramientas especiales de edición.
- El flujo de trabajo R markdown –> markdown –> HTML se puede gestionar con

  facilidad usando RStudio.
- Se pueden generar presentaciones con R markdown usando el paquete slidify

Compilación de estos documentos utilizando knitr

Programación estadística literaria con knitr

Problemas, problemas

- Los autores deben realizar un esfuerzo considerable para poner datos / resultados en la web
- Los lectores deben descargar datos / resultados individualmente y reconstruir los datos que van con secciones que código, etc
- Autores / lectores deben interactuar de forma manual con los sitios web
- No existe un documento único para integrar el análisis de datos con representaciones textuales; es decir, datos, código, y texto no están vinculados

Programación estadística literaria

- La idea original proviene de Don Knuth

- Un artículo es un flujo de texto y código
- El código del análisis se divide en texto y trozos de código "chunks"
- El código de la presentación da formato a los resultados (tablas, figuras, etc)
- El texto del artículo explica lo que está sucediendo
- Los programas literarios se *tejen* para producir documentos legibles y se *enredan* para producir documentos de lectura mecánica.
- La programación literaria es un concepto general. se necesita:
  - Un lenguaje de documentación
  - Un lenguaje de programación
- El sistema Sweave original desarrollado por Friedrich Leisch utiliza LaTeX y R
- knitr soporta varios lenguajes de documentación

¿Cómo hacer que mi trabajo sea reproducible?

- Decidirse a hacerlo (lo ideal desde el principio)
- Llevar un registro de las cosas, tal vez con un sistema de control de versiones para rastrear instantáneas / cambios
- Utilice de software cuyo funcionamiento se puede codificar
- No guarde la salida
- Guarde los datos en formatos no propietarios

Beneficios la literate Programming

- Texto y código, todo en un solo lugar, siguiendo un orden lógico
- Los datos, resultados actualizados automáticamente para reflejar los cambios externos
- Código es "vivo"-automático - "prueba de regresión" mientras se construye el documento

Debilidades

- El texto y código en un solo lugar; puede hacer que los documentos sean difíciles de leer, sobre todo si hay una gran cantidad de código.
- Puede ralentizar sustancialmente el procesamiento de documentos (aunque hay herramientas para resolverlo)

¿Qué es knitr?

- Un paquete R escrito por Yihui Xie (mientras que él era un estudiante de postgrado en Iowa State)
- Disponible en CRAN
- Soporta RMarkdown, LaTeX y HTML como lenguajes de documentación
- Puede exportar a PDF, HTML
- Integrada en RStudio para su conveniencia

Requisitos

- Una versión reciente de R
- Un editor de texto (el que viene con RStudio está bien)
- Algunos paquetes de apoyo también disponibles en CRAN
- Algún conocimiento de Markdown, LaTeX o HTML

- Usaremos Markdown aquí

¿Para que es bueno knitr?

- Manuales
- Documentos técnicos de longitud media/corta
- Tutoriales
- Informes (En especial si se generan periódicamente)
- Preprocesamiento de datos documentos/resúmenes

¿Para que NO es bueno?

- Artículos de investigación muy largos
- Cálculos complejos que consumen mucho tiempo
- Documentos que requieren un formato preciso

Mi primer documento knitr

Nuevo archivo -> R markdown Dentro del archivo insertar "trozos de R" (R chunks) Para procesar hacer clic en "Knit HTML"

Se podría hacer lo anterior por línea de comandos

```
library(knitr)
setwd(<working directory>)
knit2html("document.Rmd")
browseURL("document.html")
```

Salida HTML

Se puede observar la conversión a HTML del texto a Markdown Y el código y el resultado del código R convertido. knitr produce Markdown a partir de los trozos de R

Notas

- knitr llenará un nuevo documento con el texto de relleno; elimínelo
- Los trozos de código empiezan con:

  ```
  ```{r} y terminan con```
  ```

- Todo el código R va en entre estos marcadores
- Los trozos de código pueden tener nombres, lo cual es útil cuando empezamos a hacer gráficos:

  ```
  ```{r primer trozo}
  ## El código R va aquí
  ```
  ```

Por defecto, el código en un trozo de código se reproduce, al igual que los resultados de la computación (si hay resultados para imprimir)

Flujo de procesamiento con el documento en knitr

- Usted escribe el documento RMarkdown (.Rmd)
- knitr produce un documento Markdown (.md)
- knitr convierte el documento Markdown a HTML (por defecto)
  ```

- .RMD -> .md -> Html
- No debe modificar (o guardar) los documentos .md o html hasta que haya terminado

En Resumen

- Programación estadística literaria puede ser una forma útil para poner texto, código, datos y salidas en un solo documento
- knitr es una poderosa herramienta para la integración de código y de texto en un formato de documento sencillo

Investigación Reproducible

SI: Comience con buena ciencia

- Basura entra, basura sale
- Una pregunta coherente y enfocada simplifica muchos los problemas
- Trabajar con buenos colaboradores refuerza las buenas prácticas
- Algo que es interesante para usted será (esperemos) motivar buenos hábitos

NO: haga las cosas a mano

- Editar hojas de cálculo de datos para "limpiarlas"
  - Eliminación de los valores atípicos
  - Control de calidad
  - Validar
- Edición de tablas o figuras (por ejemplo: redondeo, formato)
- Descarga de datos de un sitio web (hacer clic en enlaces en un navegador web)
- Mover datos en su equipo; separar / reformatear archivos datos
- "Sólo vamos a hacer esto una vez ...."

Las cosas hechas a mano deben estar documentadas con precisión (esto es más difícil de lo que parece)

NO: Point And Click

- Muchos paquetes de análisis estadísticos/procesamiento de datos tienen interfaces gráficas de usuario (GUI)
- Las GUIs son convenientes/intuitivas, pero las acciones que se realizan con una interfaz gráfica de usuario puede ser difícil para que otros reproduzcan
- Algunos GUIs producen un archivo de registro o script que incluye comandos equivalentes; se pueden guardar para revisarse posteriormente
- **En general, tenga cuidado con el software de análisis de datos que es altamente** *interactivo*; facilidad de uso a veces puede conducir a análisis no-reproducible
- Otro software interactivo, como editores de texto, por lo general están bien

SI: Enseñar a la computadora

- Si algo hay que hacer como parte de su análisis/investigación, trate de enseñar a su equipo a hacerlo (incluso si sólo tiene que hacerlo una vez)
- Con el fin de dar instrucciones a su computadora, es necesario anotar exactamente lo que quiere decir que hacer y cómo se debe hacer
- Enseñar a un equipo casi garantiza que sea replicable "reproducible"

Por ejemplo, se puede hacer a mano:

1. Ir al repositorio de UCI Machine Learning en http://archive.ics.uci.edu/ml/

2. Descargar el **'Conjunto de datos de intercambio de bicicletas'_** haciendo clic en el enlace a la carpeta de datos, a continuación, hacer clic en el enlace al archivo zip del conjunto de datos, y elegir "Guardar enlace como ... " y después guardarla en una carpeta de su computadora

O puede enseñar a su computadora a hacer lo mismo con el código de R:

```
download.file("http://archive.ics.uci.edu/ml/machine-learning-databases/00275/
               Bike-Sharing-Dataset.zip", "ProjectData/Bike-Sharing-Dataset.zip")
```

Note que:

- Se especifica la URL completa del archivo de base de datos (sin hacer clic a través de un serie de enlaces)
- Se especifica el nombre del archivo guardado en el equipo local
- Se escribe el directorio en el que se guardó el archivo especificado ("ProjectData")
- Código siempre se puede ejecutar en R (siempre y cuando el enlace esté disponible)

SI: Utilice un sistema de control de versiones

- Haga las cosas más despacio
- Añadir los cambios en pequeños partes (no sólo realice commits masivos)
- Seguimiento de pasos; permite volver a versiones antiguas
- Software como GitHub / BitBucket / SourceForge le facilitará publicar los resultados

SI: Lleve registro de su entorno de trabajo

- Si trabaja en un proyecto complejo que involucra muchas herramientas / conjuntos de datos, el software y el entorno computacional informática puede ser fundamental para la reproducción de sus análisis
- **Arquitectura computacional**: CPU (Intel, AMD, ARM), GPUs,
- **Sistema operativo**: Windows, Mac OS, Linux / Unix
- **Cadena de herramientas**: Compiladores, interpretes, línea de comandos, lenguajes de programación (C, Perl, Python, etc.), bases de datos, software de análisis de datos
- **Software de soporte / Infraestructura: Bibliotecas, paquetes de R,** dependencias
- **Dependencias externas: sitios web, repositorios de datos,** bases de datos remotas, repositorios de software
- **Números de versión**: Lo ideal, para todo (si está disponible)

—

DO: Keep Track of Your Software Environment

```
> sessionInfo()
R version 3.1.1 (2014-07-10)
Platform: x86_64-pc-linux-gnu (64-bit)

locale:
 [1] LC_CTYPE=es_VE.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=es_VE.UTF-8        LC_COLLATE=es_VE.UTF-8
 [5] LC_MONETARY=es_VE.UTF-8    LC_MESSAGES=es_VE.UTF-8
 [7] LC_PAPER=es_VE.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=es_VE.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats     graphics  grDevices utils     datasets
[6] methods   base

other attached packages:
[1] stringr_0.6.2   openxlsx_2.0.15

loaded via a namespace (and not attached):
[1] Rcpp_0.11.3 tools_3.1.1
```

NO: Guarde la salida

- Evite guardar la salida de análisis de datos (tablas, figuras, resúmenes, datos procesados, etc), excepto tal vez temporalmente con fines de eficiencia.
- Si un archivo de salida "extraviado" no se puede conectar fácilmente con los medios con los que fue creado, entonces no es reproducible.
- Guarde código + datos que generaron la salida, en lugar de la misma salida
- Los archivos intermedios están bien siempre y cuando exista una documentación clara de cómo fueron creados

SI: Establezca su semilla

- Los generadores de números aleatorios generan números pseudo-aleatorios basados en un semilla inicial (normalmente un número o conjunto de números)
  - En R se puede utilizar la función `set.seed()` para establecer la semilla y para especificar el generador de números aleatorios a usar
- El ajuste de la semilla permite que el flujo de números aleatorios sea exactamente reproducible
- Cada vez que se genera números aleatorios para un propósito no trivial, **configure siempre la semilla**

SI: Piense en la secuencia completa

- El análisis de datos es un proceso largo; no es solamente tablas / figuras / reportes
- datos crudos -> datos procesados -> análisis -> reportes
- Como obtuvo el resultado es tan importante como el resultado.

- Cuanto más replicable se puede hacer la secuencia de análisis de los datos es mejor para todos

En resumen

- ¿Estamos haciendo buena ciencia?
- **¿Alguna parte de este análisis se hace a mano?**
    - Si es así, ¿esas partes están documentadas *con precisión*?
    - ¿La realidad coincide con la documentación?
- ¿Hemos enseñado a la computadora para hacer tanto como sea posible (es decir, codificado)?
- ¿Estamos utilizando un sistema de control de versiones?
- ¿Hemos documentado nuestro entorno de software?
- ¿Hemos guardado cualquier salida que no podemos reconstruir a partir de datos originales + código?
- ¿Cuánto se puede retroceder en la secuencia del análisis antes que los resultados no sean reproducibles (automáticamente)?

CAPÍTULO 11

Índices y tablas

- *genindex*
- *search*