Laboratorio: Introducción a la estadística usando R

Jose Ameijeiras Alonso
(basado en el material de Beatriz Pateiro López)

1 Introducción al lenguaje R **2**
1.1 Comandos y conceptos básicos . . . 2
1.2 Objetos en R . . . 3
1.2.1 Vectores . . . 3
1.2.2 Matrices . . . 5
1.2.3 Data frames . . . 7
1.3 Uso de condiciones lógicas para seleccionar subconjuntos de datos . . . 8
1.4 Funciones gráficas y argumentos de funciones gráficas en R . . . 9
1.5 Importar datos en R . . . 11

2 Análisis exploratorio de datos **11**
2.1 Tablas de frecuencias y resúmenes gráficos . . . 12
2.2 Medidas características . . . 12
2.2.1 Medidas de posición . . . 13
2.2.2 Medidas de dispersión . . . 13
2.2.3 Medidas de forma . . . 14
2.3 El diagrama de cajas . . . 14

3 Ejercicios de repaso **16**

Referencias **17**

El entorno R es un conjunto integrado de programas para manejo y análisis de datos, cálculo y creación de gráficos. Se trata de un proyecto de software libre iniciado por Ross Ihaka y Robert Gentleman (R&R) en los años 90. En la actualidad, el *R Development Core Team* es el responsable de su desarrollo y mantenimiento. Además, la comunidad de usuarios que programan en R ha crecido considerablemente, proporcionando bibliotecas o paquetes con funcionalidades adicionales a las proporcionadas por la distribución básica. El objetivo de esta sesión es familiarizar al alumno con el entorno y sus principales utilidades. Al mismo tiempo, revisaremos algunas de las técnicas estadísticas más habituales y su implementación en R.

1 Introducción al lenguaje R

El sistema de R consta de un sistema base y de paquetes adicionales que extienden su funcionalidad (contributed packages). Todo lo necesario para instalar R se puede obtener a través de la página web del proyecto. Este es también el sitio principal para encontrar más información sobre R: documentación general, manuales, tutoriales, etc.

Al ejecutar R, verás que dentro de la ventana (R GUI) aparece una barra de menú, una barra de herramientas y la **consola de R**, en donde escribiremos los comandos de R y obtendremos los resultados del programa. Para comenzar, como R permite realizar cálculos aritméticos, podríamos utilizar el programa como una potente calculadora. Los operadores artiméticos básicos son `+`, `-`, `*`, `/`, `^` para elevar a una potencia, `%%` para el resto de la división entera, `%/%` para el cociente de la división entera.

```
> 7 * (3 + 2)/5
```

La mayor parte del trabajo en R se lleva a cabo mediante **funciones**. Por ejemplo, si queremos calcular $\sqrt{9}$ en R escribimos:

```
> sqrt(9)
```

Esto le dice a R que llame a la función `sqrt`. Entre paréntesis indicamos los argumentos de la función. R nos permite asignar valores a variables y referenciarlas por su nombre. Usaremos el símbolo `<-` para asignar valores a variables. Recuerda que R distingue entre minúsculas y mayúsculas (case sensitive). Por ejemplo,

```
> x <- 3
> x
```

```
## [1] 3
```

1.1 Comandos y conceptos básicos

Describimos aquí algunos aspectos básicos de R.

- **Obteniendo ayuda:** Para obtener ayuda acerca de, por ejemplo, la función `sqrt`, se podrían utilizar los siguientes comandos `?sqrt`, `help(sqrt)`, `help.search("sqrt")`. Los dos primeros son equivalentes y el tercero permite realizar búsquedas más avanzadas.
- **Case sensitivity:** El lenguaje R es "case sensitive", es decir, distingue entre mayúsculas y minúsculas.
- **Funciones:** Para consultar el código de una función en R, ya sea una función propia de R o bien una del usuario, basta con teclear el nombre en la línea de comandos (sin paréntesis ni ningún tipo de argumentos).
- **Workspace:** El comando `getwd()` nos devuelve el directorio de trabajo y el comando `setwd` nos permite modificarlo. Al iniciar el programa R se abre automáticamente un espacio de trabajo, en él se almacenan todos los datos, funciones, etc. usadas durante esa sesión. En cualquier momento se pueden guardar todos los objetos del espacio de trabajo como un archivo con extensión .RData. Si en sesiones posteriores se necesitan se pueden cargar con la función `load`.
- **Librerías:** Al iniciar el programa R se cargan por defecto unas librerías básicas. A veces es necesario cargar otras librerías para realizar ciertos análisis. Esto se hace a través del comando `library`.
- **Listados:** Cuando introducimos datos en R (ya sea directamente a través del teclado o importándolos a partir de un fichero de texto), solemos guardar el resultado en un objeto. Para obtener un listado de los objetos disponibles en el directorio de trabajo, tecleamos `ls()`. Además, el comando `search()` nos da una lista de las librerías cargadas.
- **Borrado:** Podemos eliminar un objeto con la función `remove` o, equivalentemente, `rm`.

1.2 Objetos en R

R dispone de gran variedad de objetos, tales como escalares, vectores (numéricos, carácter, lógicos), factores, matrices y arrays, listas y data frames. El tipo de objeto que utilicemos dependerá de la estructura de datos que manejemos y del análisis que queramos realizar. Describiremos de forma breve algunas características de las estructuras de datos más importantes de R.

1.2.1 Vectores

El vector es la estructura de datos más simple en R. Un vector es un objeto que consiste en un número de elementos, todos ellos del mismo tipo. La forma más básica de definir un vector en R es introducir sus elementos uno a uno. Para ello se usa la función `c`, que concatena los valores introducidos en un mismo objeto. Por ejemplo:

```
> v <- c(4, 5, 23.8, 67)
> v
```

```
## [1]  4.0  5.0 23.8 67.0
```

```
> x <- c(v, v)
> x
```

```
## [1]  4.0  5.0 23.8 67.0  4.0  5.0 23.8 67.0
```

```
> x <- c(3, 5, 12)
> class(x)
```

```
## [1] "numeric"
```

```
> y <- c("Luis", "Pedro", "Laura")
> class(y)
```

```
## [1] "character"
```

Todos los elementos de un vector deben ser del mismo tipo. Si concatenamos elementos de diferentes tipos, el vector tendrá el tipo "menos restrictivo". Por ejemplo, si concatenamos números y caracteres, el vector resultante será de tipo carácter.

Además del comando `c`, existen otras funciones útiles para crear vectores. La función `seq` se utiliza para generar secuencias de números equidistantes. Por ejemplo, si queremos generar una sucesión de números que empiece en a, termine en b con incremento s, utilizamos la sintaxis `seq(from = a, to = b, by = s)`. Es decir, la opción `by` indica la distancia entre dos elementos consecutivos. Así, para generar la secuencia de los números pares de dos a cien basta escribir:

```
> seq(2, 100, by = 2)  # de 2 a 100 de 2 en 2
```

En caso de que el incremento sea 1, también podemos usar la notación `:`.

```
> 1:10
```

Si queremos generar una sucesión de números que empiece en a, termine en b y tenga longitud l, entonces utilizaremos la sintaxis `seq(from = a, to = b, length = l)`. R calcula la distancia para colocar el número de elementos pedido entre el extremos inferior y superior. Por ejemplo,

```
> seq(1, 10, length = 6)  # 6 números equidistantes entre 1 y 10
```

Cuando escribimos `seq(1, 10, length = 7)`, R asume que el primer valor que introducimos corresponde al valor inicial de la secuencia y que el segundo valor que introducimos corresponde al valor final de la secuencia (*positional matching*). Esto significa que no necesitamos especificar el nombre de los argumentos siempre que los introduzcamos en el mismo orden en el que aparecen en la lista de argumentos de la función. Si una función tiene muchos argumentos y no sabemos en que orden están definidos, lo que debemos hacer es especificar el nombre del argumento que estamos pasando. Suele ser suficiente con indicar la parte inicial del nombre del argumento siempre que no se genere confusión con otros argumentos (*partial matching of names*).

1.2.1.1 Indexado de vectores

Para conocer el valor de la componente i-ésima de un vector `x` tendremos que escribir `x[i]`. Por ejemplo:

```
> x[3]  # Tercer elemento de  x
```

Es posible recuperar más de una componente simultáneamente. Por ejemplo:

```
> y[c(1, 3)]  # Primer y tercer elemento de y
> y[1:3]  # Primeros 3 elementos de y
```

Incluso podemos indicar las componentes que no queremos recuperar:

```
> y[-2]  # Todos los elementos de y salvo el segundo
```

Podemos operar con vectores utilizando los operadores habituales (`+`,`-`, `*`, `/`, ...). Además, existen comandos en R que están especialmente pensados para trabajar con vectores. En el siguiente cuadro se muestran algunos de ellos.

Función R	Descripción
`sum(x)`	suma de los elementos de `x`
`cumsum(x)`	vector con la suma acumulada hasta cada componente `x`
`prodsum(x)`	producto de los elementos de `x`
`cumprod(x)`	vector con producto acumulado de `x`
`max(x)`	máximo de `x`
`min(x)`	mínimo de `x`
`length(x)`	longitud de `x`
`sort(x)`	ordena los elementos de `x` de menor a mayor
`mean(x)`	media aritmética de `x`
`union(x,y)`	union de los vectores `x` e `y` sin repetir
`intersect(x,y)`	comunes sin repetir en los vectores `x` e `y`
`setdiff(x,y)`	cuáles de `x` no están en `y`

1.2.2 Matrices

Una matriz es una colección de elementos dispuestos de forma rectangular (en filas y columnas). Como ocurría con los vectores, los elementos de una matriz deben ser todos del mismo tipo. Una matriz para R es esencialmente un vector que tiene dimensiones, donde las filas y las columnas juegan un papel bastante simétrico (cosa que no ocurre con los data frames). La forma habitual de crear matrices en R es con la función `matrix`.

```
> a <- matrix(10:15, nrow = 2, ncol = 3)
> a
```

```
##      [,1] [,2] [,3]
## [1,]   10   12   14
## [2,]   11   13   15
```

```
> class(a)
```

```
## [1] "matrix"
```

```
> dim(a)  # Dimensión de una matriz [nº de filas, nº de columnas]
```

```
## [1] 2 3
```

Las matrices se van rellenado por columnas. Utiliza el argumento `byrow = TRUE` para rellenar una matriz por filas.

```
> b <- matrix(10:15, nrow = 2, ncol = 3, byrow = TRUE)
> b
```

```
##      [,1] [,2] [,3]
## [1,]   10   11   12
## [2,]   13   14   15
```

```
> class(b)
```

```
## [1] "matrix"
```

```
> dim(b)
```

```
## [1] 2 3
```

Otras funciones útiles para definir matrices son `cbind` y `rbind`. La función `cbind` crea una matriz combinando 2 o más vectores por columnas. La función `rbind` crea una matriz combinando 2 o más vectores por filas.

```
> d <- cbind(x, 2 * x)
> d
```

```
##       x
## [1,]  3  6
## [2,]  5 10
## [3,] 12 24
```

1.2.2.1 Indexado de matrices. Como hacíamos con los vectores, haremos referencia a elementos de una matriz utilizando los corchetes. Hacemos referencia a las filas con el primer índice y a las columnas con el segundo índice (ambos separados por una coma).

```
> a[2, 3]  # Elemento de a en la segunda fila, tercera columna
> a[2, ]  # Segunda fila de a
> a[, 3]  # Tercera columna de a
```

Al igual que pasaba con los vectores podemos pedir un conjunto de filas y/o columnas. Por ejemplo,

```
> a[, c(1, 3)]  # Columnas 1 y 3
```

También podemos eliminar columnas utilizando el signo menos. Por ejemplo,

```
> a[, -1]  # Matriz a sin la primera columna
```

R nos permite realizar operaciones aritméticas con matrices utilizando los operadores habituales (`+`, `-`, `*`, `/`, ...). Las operaciones se realizan elemento a elemento. Por ejemplo:

```
> a * b  # Producto elemento a elemento a[i,j]*b[i,j]
```

El producto matricial en R se obtiene con el operador `%*%`. Es decir, dadas dos matrices `A` y `B`, el comando `A%*%B` multiplica las filas de la matriz `A` por las columnas de la matriz `B` (siempre que que el número de columnas de `A` sea igual al número de filas de `B`).

Existen comandos en R que están especialmente pensados para trabajar con matrices. Algunos de ellos se muestran en el siguiente cuadro.

Función R	Descripción
`dim(A)`	dimensión de `A`
`nrow(A)`	número de filas de `A`
`ncol(A)`	número de columnas de `A`
`rownames(A)`	nombre de las filas de `A`
`colnames(A)`	nombre de las columnas de `A`
`t(A)`	traspuesta de la matriz `A`
`crossprod(A,B)`	producto `t(A)%*%B`
`crossprod(A)`	producto `t(A)%*%A`
`solve(A)`	inversa de `A`
`rowMeans(A)`	medias por filas
`rowSums(A)`	sumas por filas
`colMeans(A)`	medias por columnas
`colSums(A)`	sumas por columnas

La función `apply` permite repetir una misma operación en cada una de las filas (o columnas) de una matriz. En muchos casos esto es más rápido que otras alternativas para repetir una operación. Su sintaxis es `apply(X, MARGIN, FUN)`. En esta expresión `X` es una matriz, `MARGIN` puede valer 1 o 2 y `FUN` es un nombre de comando que actúa sobre vectores. Si `MARGIN = 1` se genera un nuevo vector como resultado de aplicar `FUN` a cada fila de `X`. Si `MARGIN = 2`, se genera un nuevo vector como resultado de aplicar `FUN` a cada columna de `X`.

1.2.3 Data frames

Los data frame son la estructura de datos más importante de R. Un data frame es un objeto con filas y columnas. Es más general que una matriz ya que permite combinar en un mismo objeto columnas de distintos tipos (en un data frame podemos tener, por ejemplo, columnas numéricas y columnas de tipo carácter, mientras que en una matriz todos los elementos deben ser del mismo tipo). Trabajaremos como ejemplo con el conjunto de datos conocido como *Fisher's iris data*. Este conjunto nos da la medida en cm. de las variables longitud y anchura de sépalo y longitud y anchura de pétalo para un total de 150 flores de tres especies diferentes de iris (Iris setosa, versicolor y virginica) y están disponibles en R (R nos permite cargar en el directorio de trabajo conjuntos de datos que ya existen en formato R y que forman parte de alguna de las librerías instaladas). Para acceder a los datos en R usaremos:

```
> data(iris)
> head(iris)  # Muestra las primeras filas del conjunto de datos
```

```
##   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
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
> class(iris)
```

```
## [1] "data.frame"
```

Puesto que un data frame puede ser considerado como una generalización de las matrices, muchas de las funciones que operan sobre matrices también se pueden utilizar con data frames.

1.2.3.1 Indexado de data frames

Obtenemos los nombres de las variables de un data frame con la función `names`.

```
> names(iris)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"
## [5] "Species"
```

Podemos acceder a los datos de las variables de un data frame de manera individual con la notación `$`. Por ejemplo,

```
> iris$Sepal.Length
```

Los métodos de indexado que utilizábamos con las matrices también funcionarán con data frames.

```
> iris[3, 5]  # Especie (5ª columna) de la tercera observación
```

También podemos usar el indexado para extraer subconjuntos de un data frame. Por ejemplo, si queremos obtener las anchuras de sépalo y especie de todas las observaciones, escribiremos:

```
> iris[, c("Sepal.Width", "Species")]
```

1.3 Uso de condiciones lógicas para seleccionar subconjuntos de datos

En la práctica, es habitual trabajar con subconjuntos de datos del conjunto original que satisfagan ciertos criterios. Por ejemplo, nos puede interesar conocer las longitudes de pétalo de aquellas flores cuya longitud de sépalo es superior a 5 cm o las anchuras de pétalo de las flores de la especie Virginica. Podemos seleccionar subconjuntos utilizando condiciones lógicas en R. El siguiente cuadro muestra los operadores lógicos en R. El uso de operadores lógicos con vectores es extremadamente útil.

`<`	menor que
`<=`	menor o igual que
`>`	mayor que
`>=`	mayor o igual que
`==`	igual a
`!=`	no igual a

Un operador lógico permite comprobar si se cumple o no una determinada condición. Si se usa en un vector (por ejemplo un conjunto grande de medidas) podremos conseguir el número de casos que cumplen la condición. Por ejemplo,

```
> iris$Sepal.Length >= 5
```

Este comando nos diría qué elementos del vector `iris$Sepal.Length` son mayores o iguales a 5. Para los valores que cumplen la condición devuelve el valor lógico `TRUE` y para el resto `FALSE`. El resultado es un vector de valores de tipo lógico. Si queremos contar cuántos son podemos usar el comando `sum`, que suma los elementos de un vector. Así,

```
> sum(iris$Sepal.Length >= 5)
```

sumaría los elementos que aparecen etiquetados como `TRUE`.

Los operadores lógicos se pueden utilizar para hacer comparaciones entre vectores lógicos.

`&`	"y" lógico
`|`	"o" lógico
`!`	"no" lógico

Por ejemplo, para comprobar si las longitudes de sépalo están comprendidas entre 5 y 6 tendríamos que escribir

```
> (iris$Sepal.Length >= 5) & (iris$Sepal.Length <= 6)
```

Las funciones `any` y `all` son muy útiles para trabajar con vectores lógicos. La función `any` comprueba si algún valor es verdadero mientas que `all` se usa para saber si todos son verdaderos. Por ejemplo

```
> any(iris$Sepal.Length >= 5)
```

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

```
> all(iris$Sepal.Length >= 5)
```

```
## [1] FALSE
```

También podemos combinar la sintaxis de indexado que hemos visto con condiciones lógicas para definir las componentes que queremos recuperar a través de una condición. Por ejemplo

```
> iris$Petal.Length[iris$Sepal.Length >= 5]
```

nos devuelve las longitudes de pétalo de aquellas flores cuya longitud de sépalo es superior a 5 cm. Para obtener las anchuras de pétalo de las flores de la especie Virginica escribimos:

```
> iris$Petal.Width[iris$Species == "virginica"]
```

A estos nuevos vectores les podríamos aplicar cualquier función de las que hemos visto para vectores como, por ejemplo, la media. Obtendríamos así la anchura media de pétalo de las flores de la especie Virginica:

```
> mean(iris$Petal.Width[iris$Species == "virginica"])
```

```
## [1] 2.026
```

1.4 Funciones gráficas y argumentos de funciones gráficas en R

Existen dos tipos básicos de funciones gráficas en R: las conocidas como funciones de primer nivel y las de segundo nivel.

- De primer nivel: Crean las ventanas gráficas y establecen sus coordenadas.
- De segundo nivel: Añaden elementos gráficos en ventanas gráficas ya abiertas y con coordenadas establecidas. Las funciones gráficas de segundo nivel no se pueden ejecutar si antes no se ha usado una función de primer nivel. R dará un error si se intentan usar sin abrir antes una ventana gráfica

La función `plot` es la función genérica de dibujo de primer nivel. Por tanto abrirá un gráfica con un sistema de coordenadas. En su versión más simple dibuja un gráfico de puntos.

```
> plot(iris$Petal.Width, iris$Petal.Length)
```

La función `plot` y el resto de funciones de primer nivel admiten muchos argumentos que nos permiten cambiar la apariencia del gráfico. En el siguiente cuadro se muestran algunos de los que más se utilizan.

`main`	título principal
`xlab`	etiqueta para el eje x
`ylab`	etiqueta para el eje y
`pch`	tipo de punto
`lty`	tipo de línea
`lwd`	ancho de línea
`col`	color
`xlim`	límites del eje x
`ylim`	límites del eje y

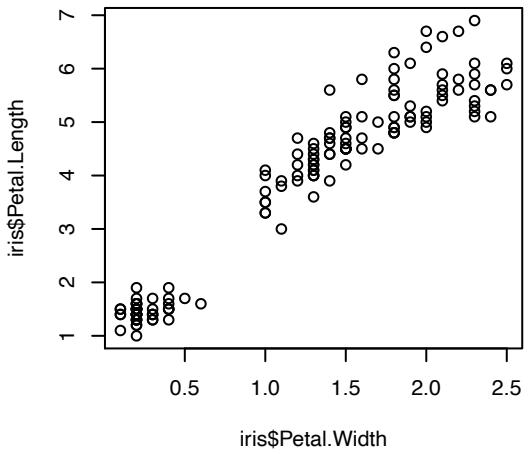


Figure 1: Diagrama de dispersión de la anchura de pétalo frente a logitud de pétalo en flores de iris

Así, podemos cambiar el color y tipo de punto para representar el diagrama de dispersión anterior.

```
> plot(iris$Petal.Width, iris$Petal.Length, col = 2, pch = 19)
```

Otra función de primer nivel es la función `pairs`, que representa matrices de diagramas de dispersión.

```
> pairs(iris[, 1:4], col = iris$Species)
```

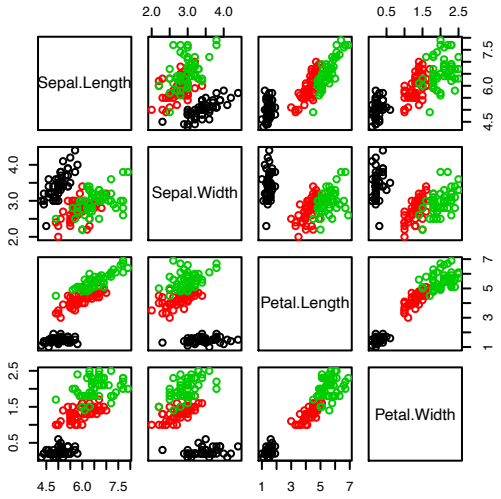


Figure 2: Diagramas de dispersión para datos de iris

Siempre que queramos añadir algún elemento nuevo a un gráfico ya creado (líneas, puntos, texto, etc.), debemos utilizar una función de segundo nivel. A continuación se muestran algunas de las principales funciones de segundo nivel en R.

`lines`	añade una línea a un gráfico
`points`	añade puntos a un gráfico
`abline`	añade una línea recta a un gráfico
`text`	añade texto a un gráfico
`legend`	añade una leyenda a un gráfico

1.5 Importar datos en R

Hasta ahora hemos visto una forma muy sencilla de introducir datos en R: directamente por teclado. Sin embargo esta no es la forma más habitual de introducir datos en R. Frecuentemente querremos que R lea datos de otros programas.

Son varias las funciones de R que nos permite leer datos dispuestos de forma rectangular en un fichero de texto (`read.table`, `read.csv`, ...). Todas ellas tienen como primer argumento:

- `file`: nombre del fichero que contiene los datos que queremos leer.

Además, estas funciones tienen otros argumentos. Algunos de los más importantes son:

- `header`: valor lógico (`TRUE` o `FALSE`) que indica si el fichero contiene en la primera fila los nombres de las variables.
- `sep`: carácter separador de columnas.
- `dec`: carácter utilizado para el punto decimal.

Las principales diferencias entre estos comandos están en qué caracteres se usan por defecto para separar los números de la tabla y cómo se representa la coma decimal de los números.

Función R	`header`	`sep`	`dec`
`read.table`	`FALSE`	`" "`	`"."`
`read.csv`	`TRUE`	`","`	`"."`
`read.csv2`	`TRUE`	`";"`	`","`
`read.delim`	`TRUE`	`"\t"`	`"."`
`read.delim2`	`TRUE`	`"\t"`	`","`

2 Análisis exploratorio de datos

> *It is important to understand what you can do before you learn to measure how well seem to have done it* (Tukey 1977)

El objetivo del análisis exploratorio de datos es identificar las principales características de un conjunto de datos por medio de resúmenes numéricos y gráficos. Estas herramientas nos permitirán conocer características básicas de la distribución de las variables, detectar relaciones entre ellas, reconocer la presencia de valores atípicos, etc.

Trabajaremos de nuevo con el conjunto de datos de iris. Comenzaremos con un análisis individual de cada variable del conjuntos de datos y posteriormente estudiaremos las relaciones entre variables. Debemos distinguir entre las variables cualitativas, que no tienen naturaleza numérica, y las cuantitativas que sí la tienen. Dentro de éstas últimas debemos además distinguir entre las variables de tipo discreto, donde es frecuente que se repitan los valores de la variable, y las continuas, donde no lo es (salvo que sus valores

estén redondeados numéricamente). A continuación se muestra cómo hacer en R los principales resúmenes numéricos y gráficos.

2.1 Tablas de frecuencias y resúmenes gráficos

Comenzaremos con la variable `Species` (variable cualitativa). El siguiente código muestra la tabla de frecuencias correspondiente.

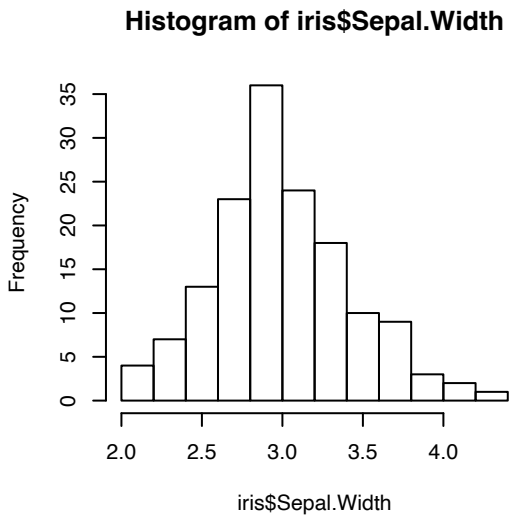
```
> fabs <- table(iris$Species)  # Frecuencia absoluta
> frel <- fabs/sum(fabs)  # Frecuencia relativa
```

Los datos de variables categóricas también se pueden resumir mediante gráficas representativas. Por ejemplo, para dibujar un diagrama de sectores utilizaremos la función `pie`. La función de R que permite crear un gráfico de barras es `barplot`.

```
> pie(fabs)  # Diagrama de sectores
> barplot(fabs)  # Diagrama de barras de frecuencias absolutas
```

El **histograma** en un método gráfico que nos permite resumir la información de una variable continua. Para obtener un histograma de, por ejemplo, la variable `Sepal.Width` se puede escribir:

```
> hist(iris$Sepal.Width)
```



Por defecto R selecciona los intervalos de agrupación con la misma longitud y en el eje vertical se representa la frecuencia absoluta de los datos que caen en cada intervalo. Si se desea representar un histograma de área uno, se debe seleccionar la opción `freq = FALSE`. También podemos introducir nosotros los puntos de corte. Para eso, se usa la opción `breaks`.

2.2 Medidas características

R nos permite calcular fácilmente medidas características. En esta sección veremos como calcular la media, mediana, varianza, desviación típica y otras medidas resumen para variables cuantitativas.

2.2.1 Medidas de posición

R ofrece funciones para calcular medidas de tendencia central. Para calcular la **media** utilizamos la función `mean`.

```
> mean(iris$Sepal.Width)
```

```
## [1] 3.057333
```

La **mediana** se calcula en R con la función `median`.

```
> median(iris$Sepal.Width)
```

```
## [1] 3
```

Los **cuantiles** se calculan en R con la función `quantile`. Por ejemplo:

```
> quantile(iris$Sepal.Width)
```

```
##   0%  25%  50%  75% 100%
##  2.0  2.8  3.0  3.3  4.4
```

Por defecto obtenemos el mínimo, el máximo, y los tres cuartiles. También podemos especificar los cuantiles que deseamos calcular. Por ejemplo, si queremos los deciles:

```
> pdec <- seq(0, 1, by = 0.1)
> quantile(iris$Sepal.Width, pdec)
```

```
##   0%  10%  20%  30%  40%  50%  60%  70%  80%  90% 100%
## 2.00 2.50 2.70 2.80 3.00 3.00 3.10 3.20 3.40 3.61 4.40
```

2.2.2 Medidas de dispersión

Una de las formas más sencillas de medir dispersión es con el **rango**. La función `range` devuelve el menor y mayor valor de un conjunto de datos. La diferencia entre ellos se calcula con la función `diff` como se muestra a continuación.

```
> range(iris$Sepal.Width)  # Mínimo y máximo
```

```
## [1] 2.0 4.4
```

```
> diff(range(iris$Sepal.Width))  # Máximo - mínimo
```

```
## [1] 2.4
```

El **rango intercuartílico** (IQR) es la distancia entre el percentil 75 y el percentil 25. Se puede calcular con la función `IQR` o usando la función `quantile` como se explicó con anterioridad.

```
> IQR(iris$Sepal.Width)
```

```
## [1] 0.5
```

La función `var` calcula la **varianza** de un conjunto de datos a partir de la fórmula:

$$s_c^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2.$$

```
> var(iris$Sepal.Width)
```

```
## [1] 0.1899794
```

La **desviación típica** se calcula como la raíz cuadrada de la varianza. Se puede calcular con la función `sd`.

2.2.3 Medidas de forma

Para obtener medidas de forma, necesitamos la librería moments:

```
> library(moments)
> skewness(iris$Sepal.Width)
```

```
## [1] 0.3157671
```

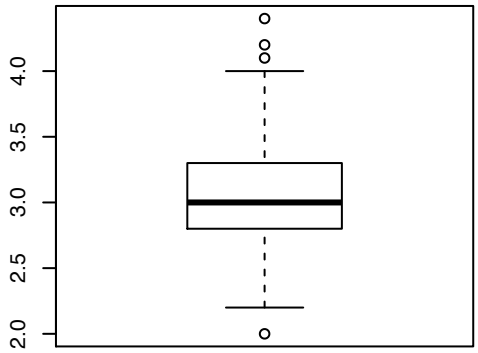
```
> kurtosis(iris$Sepal.Width)
```

```
## [1] 3.180976
```

2.3 El diagrama de cajas

El **diagrama de cajas** (boxplot) es una representación gráfica de variables continuas muy sencilla y útil en la que se visualizan, entre otros, los cuartiles y los valores atípicos. Representamos un boxplot en R con la función `boxplot`.

```
> boxplot(iris$Sepal.Width)
```



El diagrama de caja consta de una caja central que está delimitada por la posición de los cuartiles Q_3 y Q_1. Dentro de esa caja se dibuja la línea que representa la mediana. De los extremos de la caja salen unas líneas que se extienden hasta los puntos $LI = \max\{\min(x_i), Q_1 - 1.5\,(RI)\}$ y $LS = \min\{\max(x_i), Q_3 + 1.5\,(RI)\}$ que representarían el rango razonable hasta el cual se pueden encontrar datos. Los datos que caen fuera del intervalo (LI, LS) se consideran datos atípicos y se representan individualmente.

Cuando trabajamos con datos que se pueden agrupar por grupos, resulta interesante obtener información resumida y medidas características para los distintos grupos. Por ejemplo, supongamos que queremos calcular la longitud media de sépalo para cada especie. Debemos entonces considerar la variable cuantitativa `Sepal.Length` agrupada por `Species`. Para ello, podemos utilizar la función `tapply`. Por ejemplo,

```
> tapply(iris$Sepal.Length, iris$Species, mean)
```

```
##     setosa versicolor  virginica
##      5.006      5.936      6.588
```

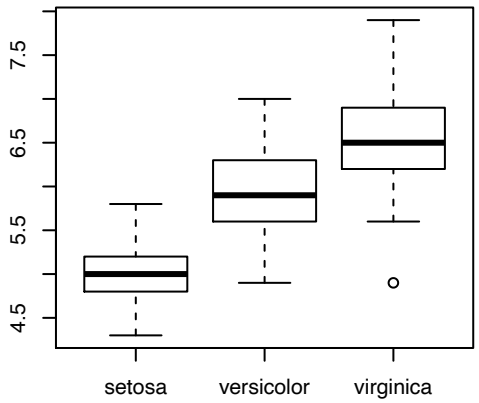
La función `tapply` toma los valores de `Sepal.Length`, los separa en función de `Species` y calcula la media (`mean`) de cada grupo. De la misma manera podemos calcular, por ejemplo, los cuantiles:

```
> tapply(iris$Sepal.Length, iris$Species, quantile)
```

```
## $setosa
##   0%  25%  50%  75% 100%
##  4.3  4.8  5.0  5.2  5.8
##
## $versicolor
##   0%  25%  50%  75% 100%
##  4.9  5.6  5.9  6.3  7.0
##
## $virginica
##    0%   25%   50%   75%  100%
## 4.900 6.225 6.500 6.900 7.900
```

Los boxplot también son útiles para entender diferencias entre grupos. Para obtener un boxplot de la longitud de sépalo para cada especie escribiremos:

```
> boxplot(iris$Sepal.Length ~ iris$Species)
```



3 Ejercicios de repaso

El fichero titanic_es.csv contiene información sobre aproximadamente el 80% de los pasajeros del Titanic[1]. Las variables incluidas en el fichero son:

- clase: clase en la que viajaba el pasajero (primera = Primera clase; segunda = Segunda clase; tercera = Tercera clase).
- sobreviviente: indica si cada pasajero sobrevivió o no al naufragio (0 = No; 1 = Sí).
- sexo: (hombre o mujer).
- edad: edad en años (los menores de un año se representan con un número menor uno).
- tarifa: precio del billete en libras.
- embarque: puerto en el que se realizó el embarque (Cherbourg, Queenstown o Southampton).

Importa los datos en R y almacena el contenido en un objeto de nombre `titanic`. Contesta a las siguientes preguntas utilizando R:

1. ¿Cuántos pasajeros aparecen registrados en el conjunto de datos?
2. ¿Cuántos pasajeros sobrevivieron al naufragio?
3. Calcula el porcentaje de pasajeros que sobrevivió al naufragio.
4. ¿Cuántos pasajeros viajaban en primera clase?
5. ¿Cuántos niños (menores de 12 años) aparecen registrados?
6. Calcula las frecuencias relativas para las distintas clases de pasajeros.

[1] http://www.encyclopedia-titanica.org/

7. Representa con un gráfico adecuado la distribución de las distintas clases de pasajeros.
8. Representa con un gráfico adecuado la distribución de la edad de los pasajeros registrados.
9. ¿Cuál era la edad media de los pasajeros? ¿Y la mediana?
10. ¿Cuál era el precio medio de un pasaje en el Titanic?
11. La frase "Mujeres y niños primero" hace referencia a un protocolo histórico por el que las mujeres y los niños debían ser los primeros en ser salvados en una emergencia. Según los datos, ¿qué porcentaje de mujeres sobrevivió al naufragio? ¿qué porcentaje de hombres sobrevivió al naufragio? ¿qué porcentaje de niños (menores de 12 años) sobrevivió al naufragio?
12. ¿Es la tasa de supervivencia de los hombres que viajaban en primera clase superior a la de los hombres que viajaban en segunda? ¿Es superior a las de los hombres que viajaban en tercera clase?
13. Representa mediante un diagrama de cajas las edades de los pasajeros agrupadas por clase. ¿Qué observas?
14. Calcula la media y la varianza de las edades en cada clase.
15. En el barco se proporcionaba asistencia especial al 5% de los pasajeros de mayor edad. ¿Qué edad debía tener un pasajero para recibir asistencia especial? ¿Qué edad tenía el pasajero más mayor?
16. Dibuja un histograma para el precio del pasaje en primera clase.
17. ¿Cuál es la moda para el puerto de embarque?
18. Los pasajeros que pagaban menos por su billete debían dormir en los camarotes más modestos del barco. En concreto, el 5% de los pasajeros que menos pagaban viajaban en dichos camarotes. ¿Cuál era el precio máximo que se pagaba por uno de estos camarotes?
19. Construye una tabla de contingencia para la distribución de la clase del pasajero por sexo.
20. Representa en un mismo gráfico la distribución de la clase del pasajero por sexo.
21. Calcula los cuartiles para la edad.
22. ¿Pagaban billete los menores de un año?

Referencias

Tukey, J.W. 1977. *Exploratory Data Analysis*. Addison-Wesley Series in Behavioral Science. Addison-Wesley Publishing Company. https://books.google.es/books?id=UT9dAAAAIAAJ.

Laboratorio: Optimización convexa

Jose Ameijeiras Alonso

1 Un problema de optimización convexa **1**

1.1 Representaciones gráficas de superficies en $\mathbb{R}^3$ 1

2 Algoritmo de descenso de gradiente **3**

2.1 Elección del paso 4

2.2 Demostración del algoritmo de descenso de gradiente 4

3 Programación en R **4**

3.1 Funciones 4

3.2 Sentencias condicionales 5

3.3 Bucles 5

En esta práctica revisaremos los principales conceptos relacionados con la optimización convexa y estudiaremos el comportamiento de uno de los métodos más habituales en la resolución de dichos problemas.

1 Un problema de optimización convexa

Considera la función $f : \mathbb{R}^2 \to \mathbb{R}$ dada por:

$$f(x, y) = \frac{1}{2}(x^2 + \gamma y^2) \tag{1}$$

siendo $\gamma > 0$. Queremos programar un metodo iterativo para minimizar la función (el óptimo se alcanza en el punto $(0, 0)$). El problema

$$\min_{(x,y)\in\mathbb{R}^2} f(x, y)$$

es un problema de optimización convexa. ¿Por qué?

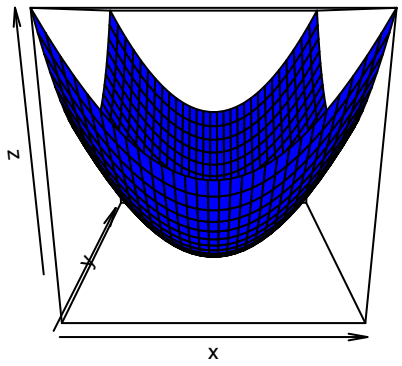
1.1 Representaciones gráficas de superficies en $\mathbb{R}^3$

La función `persp` se utiliza para la representación de superficies en $\mathbb{R}^3$. Podremos por lo tanto utilizarla para representar funciones $f : \mathbb{R}^2 \to \mathbb{R}$ como la que hemos definido anteriormente.

Para su uso, debemos dar como argumentos de entrada dos vectores, `x` e `y`, con los valores que definirán el grid de puntos sobre los que se evaluará la función f que define la superficie a representar.

A continuación, se muestra el código que representa la función dada en (1) para $\gamma = 1$. Necesitaremos definir la función `f` que queremos representar. Puedes consultar al final de este documento algunos aspectos básicos de la programación en R.

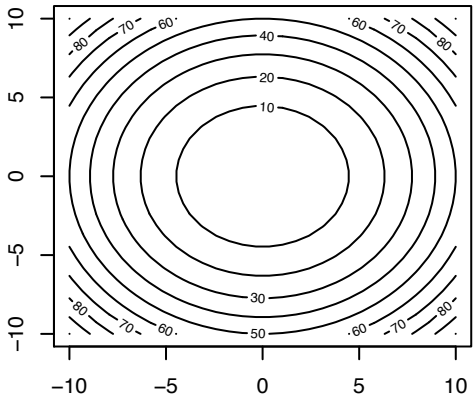
```
> x <- seq(-10, 10, length = 30)
> y <- x
> f <- function(x, y) {0.5 * (x^2 + y^2)}
> z <- outer(x, y, f)
> persp(x, y, z, col = 4)
```



Son muchas las opciones de la función. Por ejemplo, los argumentos `phi` y `theta` se usan para rotar el ángulo de visión de la superfície. Puedes consultarlas todas los argumentos de la función con `help(persp)`.

La función `contour` crea un gráfico de contornos. Su uso es similar al de la función `persp`.

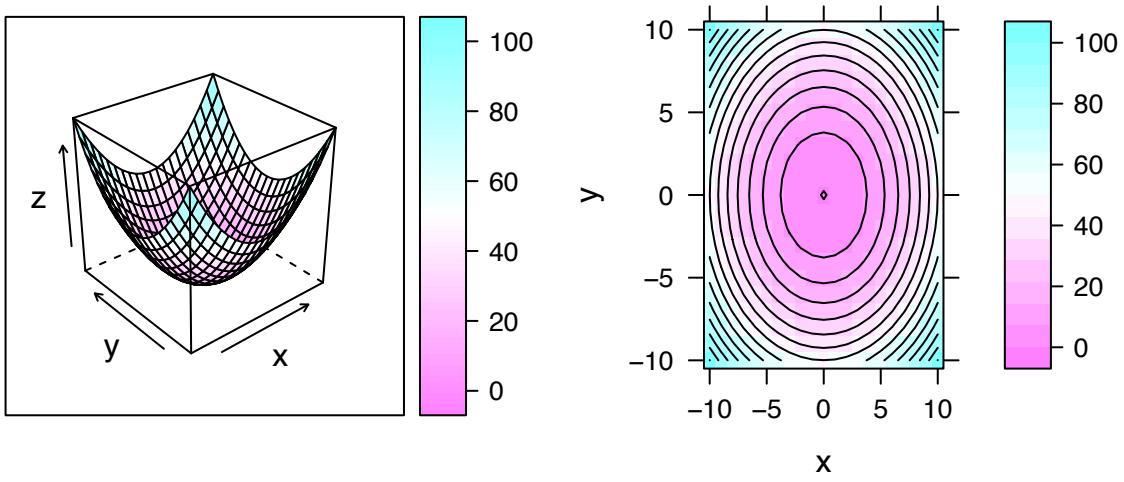
```
> contour(x, y, z)
```



De nuevo, existen muchas opciones para esta función, como por ejemplo, el número de niveles para el trazado de los contornos.

Otra opción para realizar este tipo de representaciones es utilizar la librería `lattice`. Esta librería incluye las funciones `wireframe` y `levelplot`, que proporcionan gráficos similares a los obtenidos con `persp` y `contour`, pero que además nos permiten representar la superfície con un gradiente de color. La llamada a la función es distinta, ya que como argumentos de entrada se requiere una fórmula que relacione `z` con `x` e `y`.

```
> library(lattice)
> g <- expand.grid(x = -10:10, y = -10:10)
> g$z <- 0.5 * (g$x^2 + g$y^2)
> wireframe(z ~ x * y, data = g, drape = TRUE)
> levelplot(z ~ x * y, g, contour = TRUE)
```



2 Algoritmo de descenso de gradiente

Programa el algoritmo de descenso de gradiente para minimizar una funcion objetivo $f : \mathbb{R}^2 \to \mathbb{R}$ del tipo

$$f(x, y) = \frac{1}{2}(x^2 + \gamma y^2)$$

con $\gamma > 0$. Recuerda que el algoritmo es:

Algoritmo: Método de descenso de gradiente
Dado un punto inicial $x \in$ dom(f)
repite
1. $\Delta x = -\nabla f(x)$
2. Line search. Elige un paso $t > 0$
3. Actualiza. $x := x + t\Delta x$
hasta que se cumpla el criterio de parada

Inicia el algoritmo en el punto $x^{(0)} = (\gamma, 1)^t$. Fija un número máximo de iteraciones N y el valor de η para el criterio de parada del algoritmo.

Para cada iteración k, deberás:

- Seleccionar el paso $t^{(k)} > 0$
- Evaluar el gradiente de la función objetivo en $x^{(k-1)}$
- Calcular el nuevo punto $x^{(k)}$

2.1 Elección del paso

1. En primer lugar, programa el algoritmo utilizando un paso t fijo. ¿Cómo afecta la elección de t a la convergencia?

2. Programa el algoritmo utilizando como paso de cada iteración el obtenido por el método de búsqueda exacta (exact line search). En ese caso, recuerda que:

$$t = \arg\min_{s \geq 0} f(x + s\Delta x)$$

   Utilizando este método responde a las siguientes preguntas:

   - ¿Cuántas iteraciones necesita el método para converger cuando $\gamma = 1$?
   - Analiza como es la velocidad de convergencia cuando $1/3 < \gamma < 3$, $\gamma \gg 1$ y $\gamma \ll 1$.

3. Programa el algoritmo utilizando como paso de cada iteración el obtenido por el método backtracking line search. En ese caso, recuerda que:

Algoritmo: Backtracking line search
dada una dirección descendente Δx para f en el valor $x \in \text{dom}(f)$, $\alpha \in (0, 0.5]$, $\beta \in (0, 1)$ $t := 1$ **mientras que** $f(x + t\Delta x) > f(x) + \alpha t \nabla f(x)^t \Delta x, \quad t := \beta t$

   - ¿Cómo afecta la elección de $\alpha \in (0, 0.5]$ y $\beta \in (0, 1)$?

2.2 Demostración del algoritmo de descenso de gradiente

La librería `animation` incluye una función `grad.desc` que nos permite visualizar gráficamente el método de descenso de gradiente con paso t fijo.

A continuación se muestra un ejemplo de ejecución para la función $f(x, y) = \frac{1}{2}(x^2 + 2y^2)$, punto incial $(2, 0)^t$ y $t = 0.2$.

```
> library(animation)
> oopt = ani.options(interval = 0.3, nmax = ifelse(interactive(), 50, 2))
> xx = grad.desc(FUN = function(x, y) 0.5 * (x^2 + 2 * y^2), init = c(2, 1), gamma = 0.2)
> xx$par  # solución
> xx$persp(col = "lightblue", phi = 30)  # Gráfico de superfície
```

3 Programación en R

3.1 Funciones

El lenguaje R permite al usuario definir sus propias funciones. El esquema de definición de una función es muy sencillo.

```
nombre_func <- function(arg_1, arg_2, ...) {
    expr
}
```

Si queremos que por ejemplo `arg_2` tome por defecto el valor `val` bastaría escribir

```
nombre_func <- function(arg_1, arg_2 = val, ...) {
    expr
}
```

Todo argumento que no tenga un valor por defecto será obligatorio. Una función tomará el valor de la expresión, es decir, el valor del último comando ejecutado en `expr` (que no tiene por qué ser el último comando de `expr`, la sentencia `return` puede ser utilizada para devolver un valor por el medio de una expresión y terminar la ejecución de la función).

Las funciones en R pueden ser recursivas; una función puede llamarse a si misma.

En una función en R se distinguen los siguientes tipos de variables:

- Parámetros formales: Son los argumentos de la función, cualquier modificación que se haga sobre los mismos se pierde al salir de la función.
- Variables locales: Aparecen en la función al serles asignado algún valor, desaparecen al salir de la función.
- Variables libres: Son variables que aparecen en una función sin que sean parámetros ni se les haya asignado previamente ningún valor. R busca de dentro de la función hacia fuera (podemos tener funciones anidadas) hasta que encuentra alguna variable que con su nombre. Esto se conoce como "alcance lexicográfico"

3.2 Sentencias condicionales

Como en todos los lenguajes de programación, esto se hace con las sentencias `if`. Estas sentencias en R tienen la siguiente sintaxis:

```
if (expr_1) {expr_2}
else if (expr_3) {expr_4}
else {expr_5}
```

donde `expr_1` y `expr_3` deben devolver un valor lógico. En R también se pueden construir sentencias `if` anidadas

Por ejemplo

```
> if (10>5) {cat("Hola")} else cat ("Adiós")
```

```
## Hola
```

3.3 Bucles

R admite los bucles `for`, `repeat` y `while`, su sintaxis es la siguiente:

- `for`: La sintaxis es : `for (name in expr1) expr2`. Por ejemplo, un bucle que mueva un índice `i` desde 1 hasta `n` se escribiría como: `for (i in 1:n) {expr}`.
- `repeat`: La sintaxis es `repeat expr`. En este caso hay que asegurarse de que en algún momento de `expr` se llega a un `break/return` que nos saca del bucle.
- `while`: La sintaxis es `while(cond) expr`.

Laboratorio: Modelos de regresión lineal con R

Jose Ameijeiras Alonso

1 Ajuste de un modelo de regresión lineal con R **1**

1.1 Estimación de los parámetros del modelo 2

1.2 Contrastes sobre los parámetros del modelo 4

1.3 Predicción 4

2 Ajuste de los parámetros de un modelo de regresión lineal mediante métodos de optimización **4**

2.1 Método de descenso de gradiente 5

2.2 Método de descenso de gradiente estocástico (Stochastic Gradient Descent) 6

En esta práctica repasaremos lo comandos básicos de R para ajustar un modelo de regresion lineal. Además de la estimación de los parámetros del modelo, realizaremos los contrastes estadísticos oportunos para verificar la validez del modelo propuesto. Utilizaremos como ejemplo los datos descritos en la sesión de teoría.

Puesto que el problema de estimación de los parámetros de un modelo de regresión lineal es un problema de optimización convexa sin restricciones, compararemos los valores de los parámetros ajustados con los que obtendríamos aplicando un método iterativo de optimización como el método de descenso de gradiente visto en la sesión anterior. También veremos un método alternativo al método de descenso de gradiente, denominado método de descenso de gradiente estocástico (Stochastic Gradient Descent).

1 Ajuste de un modelo de regresión lineal con R

El fichero Advertising.csv contiene información sobre las ventas de un producto en 200 mercados diferentes, junto con la inversión en publicidad de dichos mercados en distintos medios. En primer lugar, importa los datos y realiza una representación gráfica que te permita visualizar la posible relación entre las ventas y la inversión en publicidad en distintos medios.

Vamos a asumir que la variable Y (`Sales`) depende linealmente de tres variables X_1 (`TV`), X_2 (`Radio`) y X_3 (`Newspaper`), es decir, planteamos el modelo

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \epsilon.$$

Para realizar un ajuste lineal en R, utilizaremos la función `lm`. El argumento principal de la función `lm` es una **fórmula** de R. En nuestro caso escribiremos:

```
> z <- lm(Advertising$Sales ~ Advertising$TV + Advertising$Radio + Advertising$Newspaper)
```

De forma equivalente, podríamos escribir:

```
> z <- lm(Sales ~ TV + Radio + Newspaper, data = Advertising)
```

Observa que no necesitamos especificar en la fórmula el término independiente, ya que éste se incluye por defecto. La función `lm` devuelve un objeto de tipo `lm` con varias componentes.

```
> class(z)
```

```
## [1] "lm"
```

```
> names(z)
```

```
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
```

Podemos resumir los resultados del ajuste con la funcion `summary`

```
> summary(z)
```

```
##
## Call:
## lm(formula = Sales ~ TV + Radio + Newspaper, data = Advertising)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.8277 -0.8908  0.2418  1.1893  2.8292
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.938889   0.311908   9.422   <2e-16 ***
## TV           0.045765   0.001395  32.809   <2e-16 ***
## Radio        0.188530   0.008611  21.893   <2e-16 ***
## Newspaper   -0.001037   0.005871  -0.177     0.86
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.686 on 196 degrees of freedom
## Multiple R-squared:  0.8972, Adjusted R-squared:  0.8956
## F-statistic: 570.3 on 3 and 196 DF,  p-value: < 2.2e-16
```

1.1 Estimación de los parámetros del modelo

Para obtener los coeficientes estimados del modelo, podemos utilizar la función `coef`

```
> coef(z)
```

```
##  (Intercept)           TV        Radio    Newspaper
##  2.938889369  0.045764645  0.188530017 -0.001037493
```

El resultado es equivalente a:

```
> z$coefficients
```

```
##  (Intercept)           TV        Radio    Newspaper
##  2.938889369  0.045764645  0.188530017 -0.001037493
```

Comprueba que los coeficientes ajustados por el método de mínimos cuadrados se obtienen como[1]:

$$\hat{\boldsymbol{\beta}} = \left(\mathbf{X}^t\mathbf{X}\right)^{-1}\mathbf{X}^t\mathbf{y}$$

donde, en nuestro ejemplo,

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad \mathbf{X} = \begin{pmatrix} 1 & x_{11} & x_{12} & x_{13} \\ 1 & x_{21} & x_{22} & x_{23} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & x_{n3} \end{pmatrix}.$$

Para obtener los valores ajustados por el modelo de regresión lineal, $\hat{y}_i$, usaremos la función `fitted`

```
> fitted(z)
```

Podemos obtener la misma información escribiendo `z$fitted.values`. Los residuos del modelo, $\hat{\epsilon}_i$, serán por lo tanto

```
> Advertising$Sales - z$fitted.values
```

Equivalentemente,

```
> residuals(z)
```

En el modelo de regresión lineal múltiple, el RSE (residual standard error) se calcula como:

$$RSE = \sqrt{\frac{RSS}{n-p-1}}.$$

Se trata de una estimación de la desviación típica del error. Calcula el RSE a partir de los residuos del modelo y comprueba que obtienes el mismo valor que se muestra al usar la función `summary`.

Podremos obtener intervalos de confianza para los coeficientes del modelo con la función `confint`. Si no se especifica el argumento `level`, los intervalos se calculan con una confianza $1-\alpha = 0.95$.

```
> confint(z, level = 0.9)
```

```
##                    5 %        95 %
## (Intercept)  2.42340953 3.454369213
## TV           0.04345935 0.048069943
## Radio        0.17429853 0.202761502
## Newspaper   -0.01074031 0.008665319
```

[1] La función `solve` calcula la inversa de una matriz y la función `t` calcula la traspuesta

1.2 Contrastes sobre los parámetros del modelo

Analiza los resultados de los contrastes sobre los parámetros del modelo que devuelve la función `summary`. Observa que de los resultados se desprende (test F) que al menos una de las variables predictoras es útil para predecir la respuesta Y. Además, si nos fijamos en los contrastes individuales, parece que la variable `Newspaper` no es significativa. Ajusta un modelo lineal que explique la variable `Sales` como función de `TV` y `Radio` y analiza si se observa una pérdida importante en el ajuste del modelo (coeficiente R^2). Como verás, parece que el modelo que explica la variable `Sales` como función de `TV` y `Radio` es más razonable.

```
> z2 <- lm(Sales ~ TV + Radio, data = Advertising)
```

1.3 Predicción

Una vez fijado el modelo que explica la variable `Sales` como función de `TV` y `Radio`, podemos hacer predicciones como se muestra a continuación. Por ejemplo, si queremos predecir las ventas en un comercio que invierte 20000$ en publicidad en radio y 100000$ en publicidad en TV, escribiremos:

```
> newdata = data.frame(TV = 100, Radio = 20)
> predict(z2, newdata)
```

```
##        1
## 11.25647
```

A continuación, se muestra el intervalo de confianza para la venta media en mercados con una inversión de 20000$ en publicidad en radio y 100000$ en publicidad en TV. En segundo lugar se muestra el intervalo de predicción para la venta en un mercado que invierte 20000$ en publicidad en radio y 100000$ en publicidad en TV. Observa que el intervalo para la predicción es mayor.

```
> newdata = data.frame(TV = 100, Radio = 20)
> predict(z2, newdata, interval = "confidence")
```

```
##        fit      lwr      upr
## 1 11.25647 10.98525 11.52768
```

```
> predict(z2, newdata, interval = "predict")
```

```
##        fit      lwr      upr
## 1 11.25647 7.929616 14.58332
```

2 Ajuste de los parámetros de un modelo de regresión lineal mediante métodos de optimización

Consideremos de nuevo el modelo de regresión lineal

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p + \epsilon.$$

Recordamos que, dada una muestra de entrenamiento $(y_1, x_{11}, \ldots, x_{1p}), \ldots, (y_n, x_{n1}, \ldots, x_{np})$, los parametros estimados del modelo se obtienen minimizando la función:

$$RSS(\beta_0, \beta_1, \ldots, \beta_p) = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_{i1} - \ldots - \beta_p x_{ip})^2. \quad (1)$$

El problema de estimación de los parámetros se puede ver, por lo tanto, como un problema de optimización convexa sin restricciones. Podremos aproximar su solución mediante un método de optimización como, por ejemplo, el método de descenso de gradiente.

2.1 Método de descenso de gradiente

Para simplificar el problema, consideraremos un modelo de regresión lineal simple:

$$Y = \beta_0 + \beta_1 X_1 + \epsilon.$$

En primer lugar, vamos a simular una muestra con $n = 100$ observaciones, (y_i, x_i), de un modelo de regresión lineal simple con parámetros β_0 y β_1. Para ello, generamos en primer lugar los valores x_i a partir de una distribución uniforme. A continuación generamos los errores del modelo, ϵ_i, a partir de una distribución normal de media 0 y varianza σ^2. Los valores y_i se calcularán entonces como

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i.$$

En R, haremos:

```
> n <- 100
> x <- runif(n, min = 0, max = 5)  # x_i: n puntos aleatorios en el itervalo [min,max]
> beta0 <- 2  # Parámetro beta0 del modelo
> beta1 <- 5  # Parámetro beta1 del modelo
> epsilon <- rnorm(n, sd = 1)  # error (con desviación típica sd=1)
> y <- beta0 + beta1 * x + epsilon  # y_i
```

Puedes hacer un diagrama de dispersión de la muestra de entrenamiento generada y ver el efecto de diferentes valores del parámetro `sd`. Calcula el valor de los parámetros estimados con la función `lm`.

Veremos ahora como aproximar los parámetros del modelo mediante el método de descenso de gradiente (también conocido como batch gradient descent). Para ello debemos minimizar la función:

$$J(\beta_0, \beta_1) = \frac{1}{2} \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2.$$

Observa que este problema es equivalente a (1). El factor $1/2$ es únicamente para simplificar la notación del método de descenso de gradiente.

Comprueba que el método de gradiente para minimizar la función $J(\beta_0, \beta_1)$ se puede escribir:

Algoritmo: Método de descenso de gradiente para $J(\beta_0, \beta_1)$

Dado $(\hat{\beta}_0, \hat{\beta}_1)$ y $t > 0$

repite

$\hat{\beta}_0 = \hat{\beta}_0 + t \sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)$

$\hat{\beta}_1 = \hat{\beta}_1 + t \sum_{i=1}^{n} x_i (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)$

hasta que se cumpla el criterio de parada

Prográmalo en R y compara los resultados obtenidos con los que devuelve la función `lm`. Recuerda que el método de descenso de gradiente converge (siempre que el paso t no sea demasiado grande).

2.2 Método de descenso de gradiente estocástico (Stochastic Gradient Descent)

Observa que en el método de descenso de gradiente, para cada actualización del valor de los parámetros estimados, se necesita la muestra de entrenamiento al completo. Existe una alternativa al método de descenso de gradiente que también proporciona buenos resultados. El algoritmo para el caso particular del modelo de regresión lineal simple es el siguiente:

Algoritmo: Método de descenso de gradiente estocástico para $J(\beta_0, \beta_1)$

Dado $(\hat{\beta}_0, \hat{\beta}_1)$ y $t > 0$
repite
    repite para cada $i = 1, \ldots, n$
    $\hat{\beta}_0 = \hat{\beta}_0 + t(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)$
    $\hat{\beta}_1 = \hat{\beta}_1 + t x_i (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)$
hasta que se cumpla el criterio de parada

Observa que con este método se actualiza el valor de los parámetros con cada observación de la muestra de entrenamiento. Mientras que el método de descenso de gradiente (batch) necesita leer todos los datos para hacer una única iteración (lo cual puede ser costoso si n es muy grande), el método de descenso de gradiente estocástico puede empezar a iterar desde la primera observación. Programa este método y compara los resultados con los obtenidos por el método de descenso de gradiente. Puedes representar gráficamente las iteraciones en un gráfico de contorno (curvas de nivel).

Laboratorio: Modelos de regresión lineal con R (II)

Jose Ameijeiras Alonso

Hemos visto que uno de los modelos más sencillos y habituales para describir y predecir el comportamiento de una variable Y a partir de variables predictoras $X = (X_1, \ldots, X_p)$, es el modelo de regresión lineal:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p + \epsilon,$$

donde ϵ es el error, que se supone de media cero. El modelo lineal presenta ventajas desde el punto de vista de la estimación y la interpretación, sin embargo, existen situaciones en las que los estimadores de los parámetros del modelo (obtenidos mediante el método de mínimos cuadrados) podrían no resultar adecuados, dando lugar por ejemplo a una baja precisión en las predicciones. Una de las situaciones en las que surge este problema es cuando los predictores están fuertemente correlacionados. Otra situación que afecta a la precisión de la predicción del modelo lineal y también a la falta de interpretabilidad del mismo, se da cuando el número de variables explicativas p es mayor que el número de observaciones n. La regresión Ridge y Lasso son dos modelos de regresión regularizada. Este tipo de métodos nos permiten mejorar la precisión de predicción del modelo lineal en situaciones como las citadas anteriormente. En esta práctica revisaremos como ajustar con R modelos de regresion lineal regularizada.

1 Problemas en el ajuste del modelo de regresión lineal

Como acabamos de comentar, existen situaciones en las que los estimadores de los parámetros del modelo de regresión lineal (obtenidos mediante el método de mínimos cuadrados) podrían no resultar adecuados. Analizaremos en esta sección dos situaciones habituales que provocan inestabilidad en los parámetros estimados.

1.1 Efecto de la multicolinealidad en la estimación del modelo de regresión lineal por el método de mínimos cuadrados

En primer lugar analizaremos mediante un pequeño estudio de simulación, como la multicolinealidad afecta de forma importante a la varianza de los estimadores del modelo de regresión lineal.

Para ello, generaremos muestras de entrenamiento del modelo

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \epsilon$$

siendo X_1, X_2 variables con distintos niveles de correlación.

Dadas dos variables aleatorias X_1 y X_2, podemos medir la relación lineal que hay entre ambas variables mediante el coeficiente de correlación, definido por

$$\rho = \frac{\mathrm{Cov}(X_1, X_2)}{\sigma_{X_1}\sigma_{X_2}},$$

donde $\mathrm{Cov}(X_1, X_2)$ denota la covarianza entre X_1 y X_2 y σ_{X_1}, σ_{X_2} denota la desviación típica de X_1 y X_2, respectivamente. El coeficiente de correlación entre dos variables satisface $-1 \leq \rho \leq 1$. Diremos que dos variables son incorreladas si $\rho = 0$.

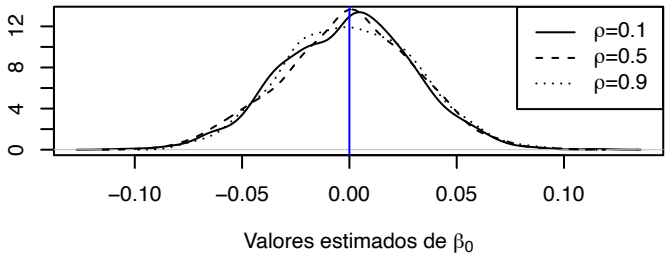
Se pueden simular fácilmente observaciones de dos variables con una determinada correlación ρ en R. Para ello, en primer lugar generamos dos secuencias de números aleatorios incorrelados, X_1 y X_1^* , a partir de una distribución normal (usando la función `rnorm`). A continuación se calcula $X_2 = \rho X_1 + \sqrt{1-\rho^2} X_1^*$. Aplica el procedimiento descrito para generar con R observaciones de dos variables con distintas correlaciones ρ. Comprueba, con la función `cor`, que la correlación muestral de las secuencias X_1 y X_2 obtenidas se aproxima a la correlación ρ elegida.

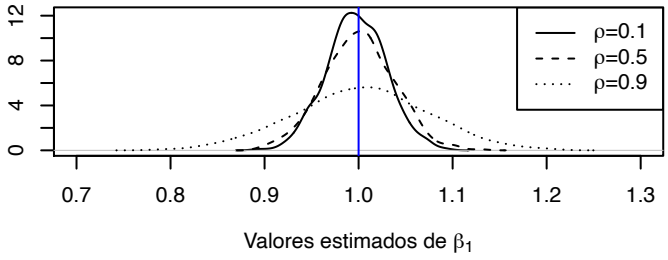
Para analizar el efecto de la correlación entre variables en la estimación de los parámetros de un modelo de regresión lineal, realiza el siguiente ejercicio.

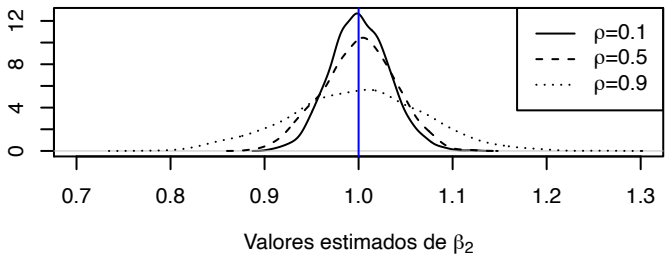
1. Genera una muestra de tamaño $n = 1000$ de dos variables X_1 y X_2 con correlación $\rho = 0.1$.
2. Genera $B = 1000$ muestras del modelo

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \epsilon$$

   con $\beta_0 = 0$, $\beta_1 = 1$ y $\beta_2 = 1$.
3. Ajusta un modelo de regresión lineal a cada una de las muestras y guarda los parámetros estimados del modelo.
4. Repite el mismo procedimiento para niveles de correlación $\rho = 0.5$ y $\rho = 0.9$.
5. Para cada uno de los parámetros del modelo, representa en un mismo gráfico la distribución de los valores estimados para los distintos niveles de correlación elegidos. Puedes usar la función `density`.







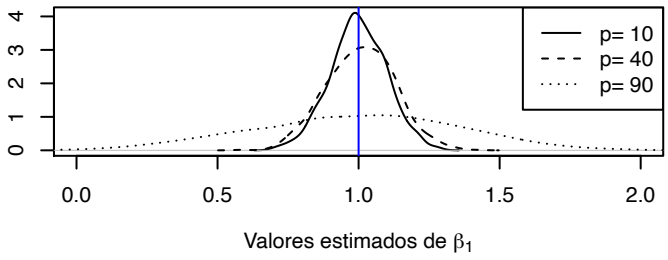
1.2 Efecto de la alta dimensión en la estimación del modelo de regresión lineal por el método de mínimos cuadrados

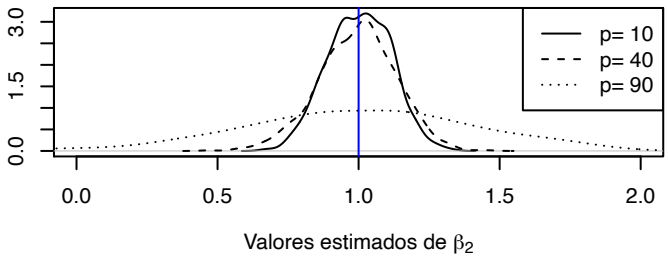
Otra de las situaciones en las que los estimadores de los coeficientes del modelo de regresión lineal son inestables se da cuando el número de variables explicativas p es elevado con respecto al número de observaciones n. Para analizar el efecto de la alta dimesión en la estimación de los parámetros de un modelo de regresión lineal, realiza el siguiente ejercicio.

1. Genera una muestra de tamaño $n = 100$ de $p = 90$ variables incorreladas.
2. Genera $B = 1000$ muestras del modelo

$$Y = \beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p + \epsilon$$

   con $\beta_0 = 0$, $\beta_i = 1$, para $i = 1, \ldots, p$.
3. Ajusta un modelo de regresión lineal a cada una de las muestras y guarda las estimaciones de los parámetros $\hat{\beta}_1$ y $\hat{\beta}_2$.
4. Repite el mismo procedimiento para distintos valores de p, por ejemplo, $p = 40$ y $p = 10$.
5. Para $\hat{\beta}_1$ y $\hat{\beta}_2$, representa en un mismo gráfico la distribución de los valores estimados de para los distintos valores de p elegidos. Puedes usar la función `density`.




p= 10
p= 40
p= 90
0.0
1.5
3.0
0.0
0.5
1.0
1.5
2.0
Valores estimados de β_2

Laboratorio: Modelos de regresión lineal con R (III)

Jose Ameijeiras Alonso

1 Regresión lineal con regularización **1**

1.1 Regresión Ridge 1

1.2 Regresión Lasso (least absolute shrinkage and selection operator) 3

1.3 Selección del parámetro de penalización 4

1 Regresión lineal con regularización

1.1 Regresión Ridge

Son varios los paquetes disponibles en R para el ajuste del modelo de regresión lineal mediante técnicas de regularización. Por ejemplo, para el caso de la regresión Ridge podremos usar, entre otros, los paquetes `MASS` y `glmnet`.

Trabajaremos con los datos discutidos en la sesión de teoría correspondientes al estudio sobre cáncer de próstata de Stamey et al. (1989). Los datos se encuentran disponibles en la librería `Brq`.

```
> library(Brq)  # Prostate data set
> data(Prostate)
```

Los datos aparecen recogidos en un `data.frame` con 9 variables y 97 observaciones. El objetivo del estudio es determinar qué variables influyen en la presencia de un antígeno prostático específico (variable `lpsa`) para detectar el cáncer de próstata. Disponemos así como variables explicativas de medidas como el volumen del tumor (`lcavol`), el log-peso de la próstata (`lweight`), la edad (`age`), la cantidad de hiperplasia prostática benigna (`lbph`), el grado de infiltración del tumor en la vesícula seminal (`svi`), el grado de penetración capsular (`lcp`) y el score de Gleason (`gleason` y `pgg45`).

Proponemos un modelo de regresión lineal para explicar la variable `lpsa` en función del resto de marcadores clínicos ($p = 8$).

$$Y = \beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p + \epsilon$$

En primer lugar, selecciona las observaciones correspondientes a la muestra de entrenamiento.

Ajusta un modelo de regresión lineal que explique la variable `lpsa` en función del resto de variables del modelo, utilizando el método de estimación de mínimos cuadrados (OLS). Recuerda que estimador por el método de mínimos cuadrados de los parámetros $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_p)^t$ se obtienen resolviendo el problema de optimización:

$$\underset{\boldsymbol{\beta}}{\text{Minimizar}} \quad \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_{i1} - \ldots - \beta_p x_{ip})^2$$

A continuación se muestran los valores de los parámetros estimados.

	OLS
intercept	0.6694
lcavol	0.5870
lweight	0.4545
age	-0.0196
lbph	0.1071
svi	0.7662
lcp	-0.1055
gleason	0.0451
pgg45	0.0045

Ajustaremos ahora el mismo modelo de regresión lineal, mediante el procedimiento de estimación Ridge. Recuerda que en ese caso, los parámetros $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_p)^t$ se obtienen resolviendo el problema de optimización

$$\underset{\boldsymbol{\beta}}{\text{Minimizar}} \quad \sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_{i1} - \ldots - \beta_p x_{ip})^2 + \lambda \sum_{j=1}^{p} \beta_j^2.$$

La solución del problema depende del valor de λ seleccionado. En particular, para $\lambda = 0$ el problema se reduce al estimador ordinario de mínimos cuadrados. Para realizar el ajuste mediante el procedimiento de estimación Ridge con R, usamos la función `lm.ridge` del paquete `MASS`. Observa que la sintaxis es similar a la de la función `lm`, añadiendo el argumento `lambda` con el valor de la penalización λ que queramos considerar.

```
> library(MASS)
> rr <- lm.ridge(y ~ x, lambda = 0)
```

En cuanto a los parámetros estimados, debemos distinguir entre el resultado almacenado en `rr$coef` y los valores que nos devuelve `coef(rr)`. Puedes observar que el resultado de `coef(rr)` coincide con el obtenido por el método de mínimos cuadrados. La diferencia con respecto a los valores almacenados en `rr$coef` es que éstos no están en la escala original. Son los coeficientes que se obtienen de ajustar el modelo de regresión a las variables predictoras estandarizadas. Es decir, coincidirían con los resultados de ajustar el modelo de regresión lineal por el método de mínimos cuadrados a los datos que se obtienen tras restar a cada columna de la matriz `x` su media y dividir entre la desviación típica. Para comprobarlo, puedes utilizar la función `scale`, que estandariza una matriz de datos por columnas.

A continuación realizamos el ajuste mediante el procedimiento de estimación Ridge, usando diferentes valores del parámetro λ. Recuerda que el incremento del valor de λ implica la contracción del vector $(\hat{\beta}_1, \ldots, \hat{\beta}_p)^t$.

```
> lam <- seq(0, 10, by = 0.01)
> rr <- lm.ridge(y ~ x, lambda = lam)
```

Puedes observar que ahora `coef(rr)` nos devuelve una matriz donde las filas recogen los parámetros estimados para cada valor de λ.

Otro de los paquetes de R que nos permiten realizar un ajuste lineal mediante el procedimiento de estimación Ridge es el paquete `glmnet`. El paquete `glmnet` es uno de los más utilizados en el contexto de regresión regularizada, ya que implementa en una única función (la función `glmnet`) distintos tipos de penalización. Veremos como es el uso de la función `glmnet` para regresión Ridge.

```
> library(glmnet)
> rr_glmnet <- glmnet(x, y, alpha = 0, lambda = 0)
```

El argumento `alpha` nos permite especificar el tipo de regularización que deseamos usar (`alpha=0` para Ridge). De nuevo, el argumento `lambda` corresponde al parámetro de penalización. Puedes observar con `coef(rr_glmnet)` que, como era de esperar, volvemos a obtener los mismos coeficientes estimados que con el ajuste por mínimos cuadrados. Usando diferentes valores del parámetro λ:

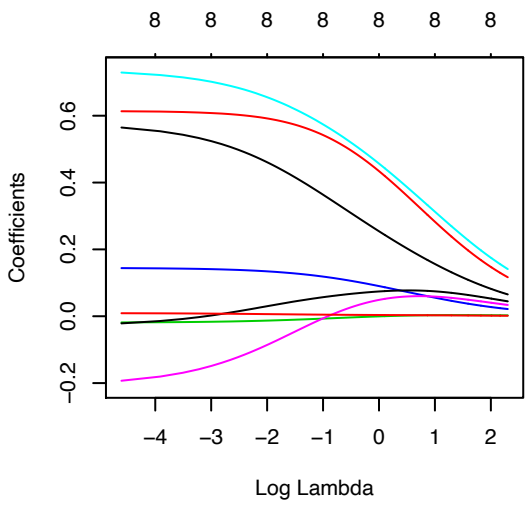
```
> lam <- seq(0, 10, by = 0.01)
> rr_glmnet <- glmnet(x, y, alpha = 0, lambda = lam)
```

Ahora, `coef(rr_glmnet)` nos devuelve una matriz donde las columnas recogen los parámetros estimados para cada valor de λ (las estimaciones se devuelven en la escala original). Fíjate que `coef(rr_glmnet)[,j]` son las estimaciones para el valor de λ almacenado en `rr_glmnet$lambda[j]`. Además, es importante mencionar que el problema de optimización que resuelve la función `glmnet` para Ridge es:

$$\underset{\boldsymbol{\beta}}{\text{Minimizar}} \quad \frac{1}{2n}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_{i1} - \ldots - \beta_p x_{ip})^2 + \frac{\lambda}{2}\sum_{j=1}^{p}\beta_j^2.$$

Observa que no coincide exactamente con la formulación original de problema y esto hace que un mismo valor de $\lambda \neq 0$ no devuelva los mismos estimadores con la función `lm.ridge` y `glmnet`. Aún así el papel de λ como parámetro de penalización es el mismo, como se observa en la siguiente gráfica, que muestra las estimaciones de los parámetros a medida que aumenta λ.

```
> plot(rr_glmnet, xvar = "lambda")
```



1.2 Regresión Lasso (least absolute shrinkage and selection operator)

Al igual que ocurre con la regularización Ridge, son varios los paquetes de R que implementan el método de reularización Lasso. Nosotros realizaremos el ajuste con la función `glmnet` que, como hemos dicho, implementa distintos tipos de penalización dependiendo del valor del argumento `alpha` que seleccionemos. Antes de realizar el ajuste, recordamos que en este caso los parámetros $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_p)^t$ se obtienen resolviendo el problema de optimización

$$\underset{\boldsymbol{\beta}}{\text{Minimizar}} \quad \sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_{i1} - \ldots - \beta_p x_{ip})^2 + \lambda\sum_{j=1}^{p} \mid \beta_j \mid .$$

Ajustamos el modelo con la función `glmnet`

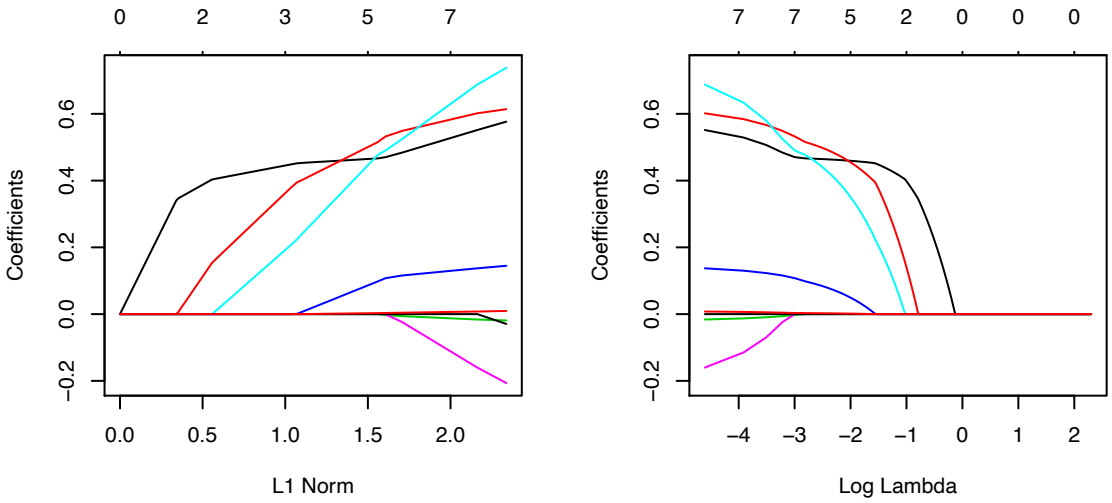
```
> library(glmnet)
> lasso <- glmnet(x, y, alpha = 1, lambda = 0)
```

El argumento `alpha` nos permite especificar el tipo de regularización que deseamos usar (`alpha=1` para Lasso). Al igual que antes, el argumento `lambda` corresponde al parámetro de penalización y, de nuevo, `coef(lasso)` nos devuelve los coeficientes estimados que coinciden con los obtenidos por el ajuste por mínimos cuadrados. Usando diferentes valores del parámetro λ:

```
> lam <- seq(0, 10, by = 0.01)
> lasso <- glmnet(x, y, alpha = 1, lambda = lam)
```

Ahora, `coef(lasso)` nos devuelve una matriz donde las columnas recogen los parámetros estimados para cada valor de λ (las estimaciones se devuelven en la escala original)[1]. Representamos las estimaciones obtenidas en función de la norma L_1 del vector de parámetros estimado y en función del logaritmo de λ:

```
> plot(lasso)
> plot(lasso, xvar = "lambda")
```



1.3 Selección del parámetro de penalización

Tanto el método de regularización Ridge como el Lasso, requieren un método para la selección del parámetro de penalización λ adecuado. Uno de los métodos más utilizados para este propósito es la *validación cruzada*. El proceso fija un rango de posibles valores λ y ajusta el modelo para cada λ utilizando una parte de la muestra de entrenamiento (un subconjunto de observaciones se apartó previamente). A continuación se analiza la capacidad predictiva del modelo sobre el subconjunto de observaciones apartado (muestra de validación). Para ello es habitual calcular el error cuadrático medio (MSE) en la muestra de validación, es decir, $\frac{1}{n_v}\sum_{i=1}^{n_v}(y_i - \hat{y}_i)^2$, donde n_v es el tamaño de la muestra de validación, y_i son las observaciones de la muestra de validación e $\hat{y}_i$ son las predicciones dadas por el modelo ajustado. El λ óptimo es aquel para el cual el MSE sea menor.

[1] La función objetivo que minimiza `glmnet` para Lasso es $\frac{1}{2n}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_{i1} - \ldots - \beta_p x_{ip})^2 + \lambda\sum_{j=1}^{p} \mid \beta_j \mid$.

Como este método depende en gran medida de qué observaciones se incluyen en la muestra para ajustar los modelos y cuáles en la muestra de validación, se han desarrollado otros métodos de validación cruzada.

Por ejemplo, la validación cruzada de k iteraciones (*k-fold cross-validation*) divide la muestra de entrenamiento en k subconjuntos de tamaño similar. En la primera etapa, se utiliza el primer subconjunto como muestra de validación, se ajusta el modelo en los $k-1$ subconjuntos restantes y se calcula MSE_1, el error cuadrático medio en la muestra de validación. A continuación se utiliza el segundo subconjunto como muestra de validación, se ajusta el modelo en los $k-1$ subconjuntos restantes y se calcula MSE_2, el error cuadrático medio en la muestra de validación. Por último, una vez repetido el mismo proceso para los k subconjuntos, se calcula la media aritmética de los MSE_i, con $i = 1, \ldots, k$. El error obtenido se denomina error de validación cruzada y el λ óptimo es aquel para el cual el error de validación cruzada sea menor. Un caso particular de validación cruzada de k iteraciones es $k = n$ (*Leave-one-out cross-validation*), donde las muestras de validación en cada iteración están formadas por una única observación.

La función `cv.glmnet` realiza la validación cruzada de k iteraciones para `glmnet` y nos devuelve el valor óptimo de λ según este criterio. Veamos como se ejecuta en primer lugar para la regularización Ridge.

```
> lam <- seq(0, 10, by = 0.01)
> cvout <- cv.glmnet(x, y, alpha = 0, lambda = lam)
```

La función `cv.glmnet` usa por defecto $k = 10$. El valor λ que minimiza el error de validación cruzada se encuentra almacenado en `cvout$lambda.min` y los parámetros obtenidos al ajustar el modelo de regresión con regularización Ridge para ese valor de λ se obtienen directamente con la función `coef` como se muestra a continuación

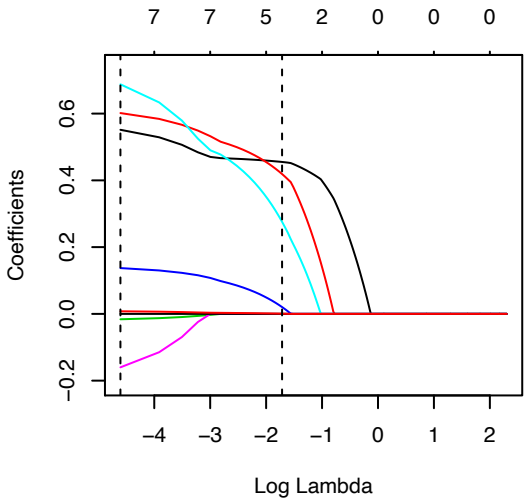
```
> cvout$lambda.min
> coef(cvout, s = "lambda.min")
```

La función también devuelve el valor `cvout$lambda.1se`, que representa un valor de λ para un modelo más simple que el de `cvout$lambda.min`, pero cuyo error está a una desviación estándar del mejor.

Del mismo modo, si queremos seleccionar el parámetro λ en el ajuste Lasso, haremos:

```
> lam <- seq(0, 10, by = 0.01)
> cvoutl <- cv.glmnet(x, y, alpha = 1, lambda = lam)
> cvoutl$lambda.min
> coef(cvoutl, s = "lambda.min")
> cvoutl$lambda.1se
> coef(cvoutl, s = "lambda.1se")
```

Mostramos a continuación una gráfica de los parámetros estimados con Lasso junto con el valor de λ seleccionado por validación cruzada (el valor óptimo `cvoutl$lambda.min` y `cvoutl$lambda.1se`)



En resumen, si seleccionamos como valores de λ los dados por `cvout$lambda.1se` (Ridge) y `cvoutl$lambda.1se` (Lasso), los parámetros estimados con mínimos cuadrados, Ridge y Lasso son los que se muestran en la siguiente tabla:

	OLS	Ridge	Lasso
Intercept	0.6694	0.3470	1.2882
lcavol	0.5870	0.2803	0.4596
lweight	0.4545	0.3183	0.1369
age	-0.0196	-0.0022	0
lbph	0.1071	0.0555	0
svi	0.7662	0.4671	0.3216
lcp	-0.1055	0.0762	0
gleason	0.0451	0.0848	0
pgg45	0.0045	0.0027	0

Laboratorio: Modelos lineales de clasificación con R (I)

Jose Ameijeiras Alonso

En esta sesión práctica revisaremos el problema de clasificación y veremos como ajustar modelos lineales de clasificación con R. Recordamos que en un problema de clasificación se dispone de un conjunto de observaciones que pueden venir de dos o más poblaciones o clases distintas. El objetivo es clasificar una nueva observación a partir de un conjunto de variables predictoras $X = (X_1, \ldots, X_p)$. Para ello contamos con la información de la muestra de entrenamiento, que consiste en observaciones de las variables predictoras junto con la clasificación correspondiente a cada observación.

En la primera parte de esta práctica comentaremos brevemente como ajustar con R un modelo de regresión logística.

1 Ajuste de un modelo de regresión logística con R

En primer lugar veremos como ajustar un modelo de regresión logística con R.

```
> library(ISLR)
> data(Default)
```

Este conjunto de datos contiene información simulada de 10000 clientes de una entidad bancaria. El objetivo es predecir cuando un cliente incurrirá en impago de crédito de la tarjeta. Para ello podemos utilizar la información correspondiente al saldo medio mensual del cliente.

Si representamos los datos, parece razonable pensar que el saldo medio mensual puede influir en la probabilidad de que un cliente incurra en impago de crédito de la tarjeta. Veremos como ajustar un modelo de regresión logística para estudiar esa posible relación. Es decir, supondremos:

$$p(X) = \mathbb{P}(Y = 1/X) = \frac{1}{1 + \mathrm{e}^{-(\beta_0 + \beta_1 X)}}$$

donde X representa el saldo mensual del cliente e

$$Y = \begin{cases} 1, & \text{si el cliente incurre en impago,} \\ 0, & \text{si el cliente no incurre en impago.} \end{cases}$$

El modelo de regresión logística se ajusta en R utilizando el comando `glm` (general linear models). Los modelos lineales generalizados son una extensión de los modelos lineales que permiten que la variable dependiente tenga una distribución no normal. La formulación de modelos lineales generalizados permite unificar en un mismo modelo métodos como la regresión lineal y la regresión logística, sin más que especificar la función link o familia de distribución de los errores correspondiente a cada caso.

Por ejemplo, la regresión logística se puede formular como un modelo lineal generalizado en el que la distribución de los errores es binomial (`family="binomial"`)

```
> fit <- glm(default ~ balance, data = Default, family = "binomial")
```

Al igual que se hacía en el análisis de regresión lineal, podremos utilizar las funciones `coef`, `summary`, `residuals`, etc. para obtener información relacionada con el ajuste del modelo. También se puede usar la función `predict` para obtener las predicciones del modelo. Si queremos obtener las predicciones para $\mathbb{P}(Y = 1/X)$, debemos añadir el argumento `type="response"`. Así:

```
> plot(Default$balance, predict(fit, type = "response"))
```

representa gráficamente las predicciones del modelo para los valores de X de la muestra de entrenamiento. Clasificaremos en el grupo de impago a aquellos clientes para los cuales la predicción obtenida para $\mathbb{P}(Y = 1/X)$ sea superior a 0.5.

Laboratorio: Modelos lineales de clasificación con R (II)

Jose Ameijeiras Alonso

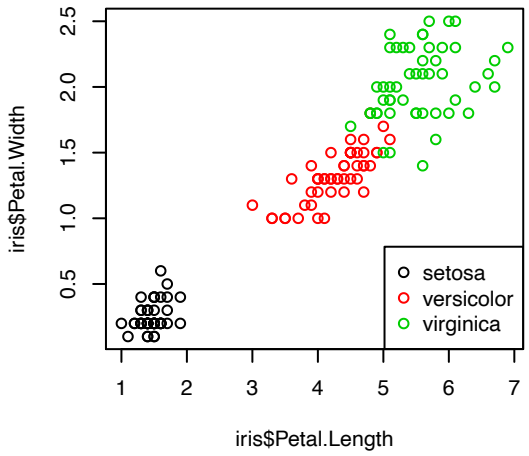
En esta sesión práctica revisaremos el problema de clasificación y veremos como ajustar modelos lineales de clasificación con R. Recordamos que en un problema de clasificación se dispone de un conjunto de observaciones que pueden venir de dos o más poblaciones o clases distintas. El objetivo es clasificar una nueva observación a partir de un conjunto de variables predictoras $X = (X_1, \ldots, X_p)$. Para ello contamos con la información de la muestra de entrenamiento, que consiste en observaciones de las variables predictoras junto con la clasificación correspondiente a cada observación. En esta práctica se llevará a cabo una clasificación por k vecinos más próximos y un Análisis Lineal/Cuadrático Discriminante.

1 k-vecinos más próximos con R

Ilustraremos el problema de clasificación con el conjunto clásico de datos de Iris. Este conjunto nos da la medida en cm. de las variables longitud y anchura de sépalo y longitud y anchura de pétalo para un total de 150 flores de tres especies diferentes de iris (iris setosa, versicolor y virginica).

Para comenzar nos centraremos en las variables longitud y anchura de pétalo. Ya sabemos como representar gráficamente los datos correspondientes a estas variables.

```
> data(iris)
> plot(iris$Petal.Length, iris$Petal.Width, col = iris$Species)
> legend("bottomright", levels(iris$Species), pch = 1, col = 1:3)
```



Para llevar a cabo k-vecinos más próximos en R utilizaremos la función `knn`, que pertence a la librería `class`.

Utilizando la distancia Euclidea clasificaremos a una nueva observación en función de los k puntos que están más cerca de él. Podríamos ver que $k = 4$ puntos quedan más cerca del punto que tiene un `Petal.length=4` y un `Petal.width=1`. Si añadimos el argumento `prob=TRUE` también nos devolverá la probabilidad de pertenece a cada grupo.

```
> library(class)
> matexp <- cbind(iris$Petal.Length, iris$Petal.Width)
> knn.pred <- knn(train=matexp, test = c(4,1), cl = iris$Species,  k=4,  prob = T)
```

En este caso, vemos que a esa nueva planta la clasificaría como `versicolor` con una probabilidad 1 (sus 4 vecinos más próximos también son versicolor).

2 Análisis Lineal Discriminante con R

Para llevar a cabo un Análisis Lineal Discriminante en R utilizaremos la función `lda`, que pertence a la librería `MASS`.

```
> library(MASS)
> lda.fit <- lda(Species ~ Petal.Length + Petal.Width, data = iris)
```

Recuerda que el Análisis Lineal Discriminante es un modelo generativo, es decir, calcula la probabilidad a posteriori $\mathbb{P}(Y = k/X = x)$ mediante el teorema de Bayes:

$$\mathbb{P}(Y = k/X = x) = \frac{\mathbb{P}(X = x/Y = k)\mathbb{P}(Y = k)}{\mathbb{P}(X = x)} = \frac{f_k(x)\pi_k}{\sum_{j=1}^{K} f_j(x)\pi_j}$$

En la expresión anterior π_k denota la probabilidad a priori de que una observación provenga de la clase k, es decir, $\pi_k = \mathbb{P}(Y = k)$. Por otro lado, $f_k(x) = \mathbb{P}(X = x/Y = k)$ representa la densidad de probabilidad de X en la clase k.

En Análisis Lineal Discriminante se asume además que las funciones de densidad de probabilidad de cada clase $f_k(x)$ son distribuciones Normales de media μ_k y con la misma matriz de covarianzas Σ para $k = 1, \ldots, K$. Una vez calculadas las probabilidades a posteriori, la regla de clasificación consiste en asignar cada observación a la clase para la cual $\mathbb{P}(Y = k/X = x)$ es mayor.

En la práctica, para llevar a cabo la clasificación, tendremos que estimar las probabilidades a priori π_k, así como los parámetros de la densidad de probabilidad Normal correspondiente a cada clase.

La salida de la función `lda` nos muestra, entre otras cosas, las estimaciones de π_k. En este caso,

```
> lda.fit$prior
```

```
##     setosa versicolor  virginica
##  0.3333333  0.3333333  0.3333333
```

Es decir, como en este caso hay 50 observaciones dentro de cada especie, se tiene que 1/3 de las observaciones pertenecen a la especie setosa, 1/3 de las observaciones pertenecen a la especie versicolor y 1/3 de las observaciones pertenecen a la especie virginica ($\hat{\pi}_k = 1/3, k = 1, 2, 3$).

También se muestran en la salida la longitud y anchura de pétalo media de cada especie (estimaciones de μ_k):

```
> lda.fit$means
```

```
##            Petal.Length Petal.Width
## setosa            1.462       0.246
## versicolor        4.260       1.326
## virginica         5.552       2.026
```

La predicción se lleva a cabo como es habitual con la función `predict`. A continuación evaluamos la función `predict` en la muestra de entrenamiento

```
> lda.pred <- predict(lda.fit)
```

Observa que en `lda.pred$class` se indica la especie asignada a cada observación por la regla de clasificación. Por otro lado, en `lda.pred$posterior` se muestra, para cada observación, el valor estimado para la probabilidad a posteriori de cada especie.

A continuación se muestra un resumen del resultado de la clasificación en la muestra de entrenamiento. Observa que todas las flores de la especie setosa han sido correctamente clasificadas, se han cometido 4 errores de clasificación en las de la especie versicolor y 2 errores de clasificación en las de la especie virginica. En resumen tenemos una tasa de error de 0.04.

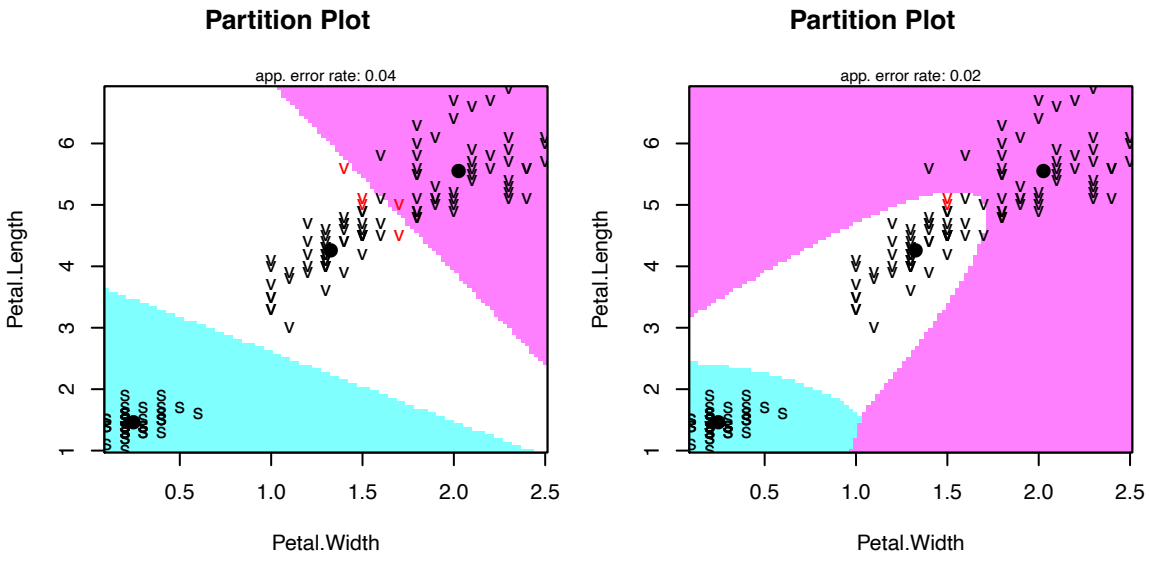
```
> table(lda.pred$class, iris$Species)
```

```
##
##              setosa versicolor virginica
##   setosa         50          0         0
##   versicolor      0         48         4
##   virginica       0          2        46
```

Debemos recordar que la regla de decisión que determina el Análisis Lineal Discriminante se basa en la suposición de la normalidad de las observaciones y en la de que las matrices de covarianzas en las clases son iguales. Si mantenemos que la densidad de probabilidad de cada clase es normal pero no podemos asumir la igualdad de las matrices de covarianzas, entonces la regla de decisión deja de dar lugar a un modelo de clasificación lineal. Estaremos en ese caso ante un Análisis Cuadrático Discriminante (QDA). Para llevar a cabo un Análisis Cuadrático Discriminante en R utilizaremos la función `qda`, que pertence a la librería `MASS`.

Por último, como en el ejemplo que hemos utilizado a lo largo de esta sección teníamos únicamente dos variables predictoras, podríamos visualizar las regiones determinadas por la regla de decisión. Para ello usaremos la librería `klaR`, que también nos permite llevar a cabo un Análisis Lineal Discriminante y ofrece más herramientas de visualización. Visualizamos al mismo tiempo los resultados de un Análisis Cuadrático Discriminante

```
> library(klaR)
> partimat(Species ~ Petal.Length + Petal.Width, data = iris, method = "lda")
> partimat(Species ~ Petal.Length + Petal.Width, data = iris, method = "qda")
```



En el gráfico se observa que las fronteras de las tres zonas de clasificación con LDA están delimitadas por rectas (es un modelo de clasificación lineal). También podemos ver señaladas en rojo las 4 observaciones de la muestra de entrenamiento que han resultado mal clasificadas con LDA.

Repite los métodos de clasificación, k-vecinos más próximos, el Análisis Lineal Discriminante y el Análisis Cuadrático Discriminante para los datos de iris utilizando todas las variables predictoras disponibles en el conjunto de datos y analiza los resultados obtenidos.

Laboratorio: Análisis de Componentes Principales con R

Jose Ameijeiras Alonso

1 Análisis de componentes principales con `princomp` 1

2 Análisis de componentes principales a través de la descomposición espectral de la matriz de covarianzas 5

3 El biplot 6

4 Las componentes principales y los cambios de escala 7

En esta sesión práctica veremos como llevar a cabo un Análisis de Componentes Principales con R. El análisis de componentes principales se concibe como una técnica de reducción de la dimensión, pues permite pasar de una gran cantidad de variables interrelacionadas a unas pocas componentes principales. El método consiste en buscar combinaciones lineales de las variables originales que representen lo mejor posible a la variabilidad presente en los datos. De este modo, con unas pocas combinaciones lineales, que serán las componentes principales, sería suficiente para entender la información contenida en los datos. Al mismo tiempo, la forma en que se construyen las componentes, y su relación con unas u otras variables originales, sirven para entender la estructura de correlación inherente a los datos. Por último, las componentes principales, que forman un vector aleatorio de dimensión menor, pueden ser empleadas en análisis estadísticos posteriores, como por ejemplo en regresión.

1 Análisis de componentes principales con `princomp`

Trabajaremos a lo largo de este laboratorio con el siguiente ejemplo. Se ha examinado a 25 alumnos, aspirantes a ingresar en el Máster Interuniversitario en Big Data, de 5 materias diferentes: Programación (cuyo resultado se almacena en la variable `prog`), Ingeniería de Computadores (`ingcom`), Ingeniería del Software (`ingsof`), Sistemas de la Información (`sist`) y Estadística (`estad`). Las puntuaciones obtenidas se encuentran en el fichero `aspirantes.txt` y se muestran a continuación.

```
> dat <- read.table("aspirantes.txt", header = TRUE)  # Lectura de los datos
> dat
```

```
##    prog ingcom ingsof sist estad
## 1    36     58     43   36    37
## 2    62     54     50   46    52
## 3    31     42     41   40    29
## 4    76     78     69   66    81
## 5    46     56     52   56    40
## 6    12     42     38   38    28
## 7    39     46     51   54    41
## 8    30     51     54   52    32
## 9    22     32     43   28    22
## 10    9     40     47   30    24
## 11   32     49     54   37    52
## 12   40     62     51   40    49
## 13   64     75     70   66    63
## 14   36     38     58   62    62
```

```
## 15   24     46     44   55    49
## 16   50     50     54   52    51
## 17   42     42     52   38    50
## 18    2     35     32   22    16
## 19   56     53     42   40    32
## 20   59     72     70   66    62
## 21   28     50     50   42    63
## 22   19     46     49   40    30
## 23   36     56     56   54    52
## 24   54     57     59   62    58
## 25   14     35     38   29    20
```

El objetivo de este estudio es obtener un ranking global de alumnos para la entrada en el máster, a través de una puntuación global, extraída como cierta combinación lineal de las calificaciones en las cinco materias examinadas.

Vamos a realizar entonces un Análisis de Componentes Principales con R. El comando básico de R que ejecuta el análisis de componentes principales es `princomp`. También se pueden obtener resultados similares con el comando `prcomp`. La diferencia principal entre uno y otro es que princomp diagonaliza la matriz S, mientras que `prcomp` diagonaliza la matriz S_c. Esto modifica los autovalores en la proporción $n/(n-1)$, pero no supone ningún cambio en los autovectores. Hemos optado por el comando `princomp`. A continuación se muestran las instrucciones que permiten realizar un análisis de componentes principales para el ejemplo, utilizando R. En primer lugar, como salida del objeto se muestran las desviaciones típicas de las componentes, que son las raíces cuadradas de los autovalores de S.

```
> test.pca <- princomp(dat)
> test.pca
```

```
## Call:
## princomp(x = dat)
##
## Standard deviations:
##    Comp.1    Comp.2    Comp.3    Comp.4    Comp.5
## 28.489680  9.035471  6.600955  6.133582  3.723358
##
##  5  variables and  25 observations.
```

Si hacemos un `summary` del objeto, obtenemos además la proporción de varianza explicada y sus valores acumulados.

```
> summary(test.pca)
```

```
## Importance of components:
##                            Comp.1     Comp.2     Comp.3     Comp.4
## Standard deviation     28.4896795 9.03547104 6.60095491 6.13358179
## Proportion of Variance  0.8212222 0.08260135 0.04408584 0.03806395
## Cumulative Proportion   0.8212222 0.90382353 0.94790936 0.98597332
##                            Comp.5
## Standard deviation     3.72335754
## Proportion of Variance 0.01402668
## Cumulative Proportion  1.00000000
```

Además de esta información, el objeto `test.pca` almacena otra información relevante.

```
> names(test.pca)
```

```
## [1] "sdev"     "loadings" "center"   "scale"    "n.obs"    "scores"
## [7] "call"
```

Como hemos visto, podemos obtener las desviaciones típicas de las componentes. De este modo podemos obtener las varianzas de las componentes, que son los autovalores.

```
> test.pca$sdev
```

```
##    Comp.1    Comp.2    Comp.3    Comp.4    Comp.5
## 28.489680  9.035471  6.600955  6.133582  3.723358
```

```
> test.pca$sdev^2
```

```
##    Comp.1    Comp.2    Comp.3    Comp.4    Comp.5
## 811.66184  81.63974  43.57261  37.62083  13.86339
```

Recordamos que si extraemos una componente principal, obtenemos la variable aleatoria unidimensional

$$z_1 = {v_1}' x.$$

Si en lugar de recoger todo el vector aleatorio, sólo aportamos z_1 reduciendo la información a una variable unidimensional, junto con la simplificación se produce una reducción de variabilidad, que pasa a ser

$$\mathrm{Var}(z_1) = \lambda_1.$$

Decimos que el cociente

$$\frac{\lambda_1}{\lambda_1 + \cdots + \lambda_d}$$

es la proporción de variabilidad explicada por la primera componente principal.

Si en lugar de una única componente principal extraemos r componentes resulta

$$\frac{\lambda_1 + \cdots + \lambda_r}{\lambda_1 + \cdots + \lambda_r + \lambda_{r+1} + \cdots + \lambda_d}$$

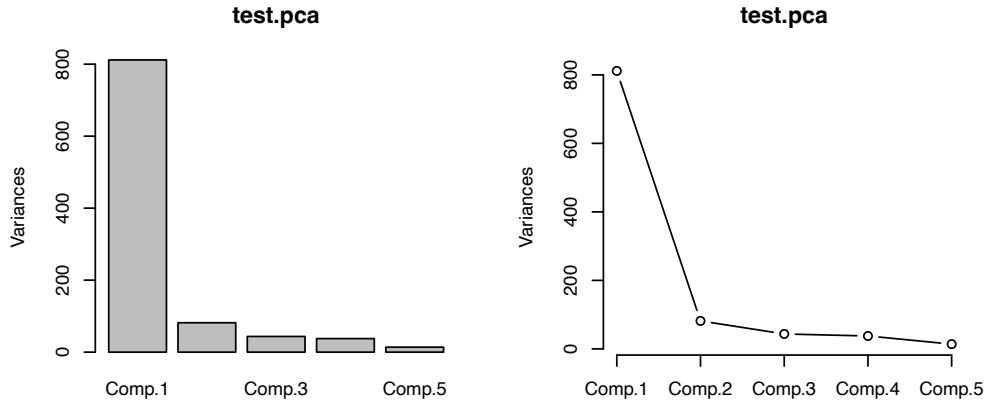
es la proporción de variabilidad explicada por las r primeras componentes principales.

Debemos decidir entre la simplificación que supone la reducción de la dimensión y la pérdida de información resultante de la variabilidad no explicada. Como criterios para tomar esta decisión, se suelen emplear los siguientes:

- Criterio de la varianza explicada. Consiste en retener el número de componentes que conjuntamente expliquen una proporción de varianza establecida, habitualmente un 90% o 95% del total.
- Gráfico de sedimentación (*screeplot*). Representar en un gráfico los valores propios $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$ en orden decreciente, y buscar un "codo" en el gráfico, entendiendo por codo un punto a partir del cual los valores propios son claramente más pequeños que los anteriores, y muy similares entre sí.
- Retener un número preestablecido de componentes principales. Por ejemplo, es costumbre retener dos componentes, pues se pueden representar fácilmente en el plano.

Podemos hacer un gráfico de las varianzas de las componentes con `screeplot`.

```
> screeplot(test.pca)  # Representación de los autovalores (gráfico de sedimentación)
> screeplot(test.pca, type = "lines")  # El mismo gráfico con líneas
```



Además, podemos obtener los autovectores asociados (columnas de la matriz `test.pca$loadings`).

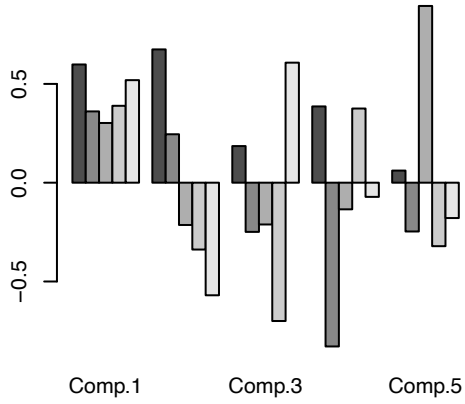
```
> test.pca$loadings
```

```
##
## Loadings:
##        Comp.1 Comp.2 Comp.3 Comp.4 Comp.5
## prog    0.598  0.675  0.185  0.386
## ingcom  0.361  0.245 -0.249 -0.829 -0.247
## ingsof  0.302 -0.214 -0.211 -0.135  0.894
## sist    0.389 -0.338 -0.700  0.375 -0.321
## estad   0.519 -0.570  0.607        -0.179
##
##                Comp.1 Comp.2 Comp.3 Comp.4 Comp.5
## SS loadings       1.0    1.0    1.0    1.0    1.0
## Proportion Var    0.2    0.2    0.2    0.2    0.2
## Cumulative Var    0.2    0.4    0.6    0.8    1.0
```

Observamos que la primera componente principal es la variable con máxima varianza y tiene todos sus coeficientes positivos. Se puede interpretar como una componente de *tamaño* que determina la "calificación general de los estudiantes". La primera componente ordena las estudiantes según su tamaño (calificación general), del más pequeño al más grande. La segunda componente tiene coeficientes positivos y negativos y se interpreta como uan componente de *forma* que ordena a los estudiantes contrastando los buenos en Programación e Ingeniería de Computadores frente a los buenos en el resto de materias.

Representamos los coeficientes de las componentes principales mediante el siguiente gráfico.

```
> barplot(loadings(test.pca), beside = TRUE)
```



Podemos obtener también el número de observaciones, el vector de medias y los escalados aplicados a cada variable.

```
> test.pca$n.obs
```

```
## [1] 25
```

```
> test.pca$center
```

```
##   prog ingcom ingsof   sist  estad
##  36.76  50.60  50.68  46.04  43.80
```

```
> test.pca$scale
```

```
##   prog ingcom ingsof   sist  estad
##      1      1      1      1      1
```

Por último, obtenemos los `scores`, que son el resultado de XP, siendo P la matriz que tiene como columnas los autovectores de S.

```
> test.pca$scores
```

```
##            Comp.1      Comp.2      Comp.3      Comp.4     Comp.5
##  [1,]  -7.5403215  10.2167650   2.5374713  -8.6708997 -4.3011164
##  [2,]  20.3610372  13.3460340   8.9820585   6.4124949 -1.3548711
##  [3,] -19.5031539   6.5552439  -1.6414327   5.0042015 -2.2980498
##  [4,]  65.9652730   1.3136646   5.1988159  -5.2093505 -1.0457668
##  [5,]   9.7780565   6.0680143  -9.1921582   2.9249303 -2.1069944
##  [6,] -33.0739529  -4.3722312  -3.7345190  -2.6038323 -5.3246839
##  [7,]   1.4212177  -0.7833317  -5.7800406   7.8225646 -0.4967347
##  [8,]  -6.7011638  -0.4667798 -13.3936497  -0.3040531  2.6525120
##  [9,] -36.1916160   5.6543609   2.9062814   5.5458471  6.5171961
## [10,] -38.0586178  -3.8266815  -2.5248244  -6.0338259  6.3212284
## [11,]  -1.6837296  -5.9262762  10.1237093  -4.9410716  4.5102357
## [12,]   6.4961788   3.9922267   5.0805842 -10.8805481 -1.3208797
## [13,]  48.6656614   2.5248899  -7.4227003  -6.1976282  3.0745497
## [14,]  12.8648106 -20.9375616   1.3328311  13.8459667  1.2286291
## [15,]  -5.1279619 -14.2990082  -2.9194088   2.7778854 -9.4299722
## [16,]  14.7627376   1.9542134   2.1019732   6.8802981  0.7262472
```

```
## [17,]   0.5206666   0.3328352  12.2272438   5.5083908  5.1001830
## [18,] -55.8465116   0.7024956   1.3349377  -4.9981377 -2.2873123
## [19,]   1.2809636  24.1913016   1.8618184   5.1875656 -3.1248129
## [20,]  44.0731110  -1.0133086  -8.2094067  -5.5695807  3.6879677
## [21,]   2.7282968 -15.4819784  13.1612877  -5.6870830 -3.1343155
## [22,] -22.3031608  -2.8413762  -5.9443642  -4.0929718  2.9545412
## [23,]  10.4526967  -7.6936163  -3.2010012  -3.0864389 -0.6469043
## [24,]  28.7147101  -2.0746379  -2.7048913   5.2002364 -0.7510021
## [25,] -42.0552279   2.8647426  -0.1806153   1.1650401  0.8501260
```

2 Análisis de componentes principales a través de la descomposición espectral de la matriz de covarianzas

Comprobamos que, efectivamente, los resultados prporcionados por la función `princomp` coindicen con los obtenidos a partir de la descomposición espectral de la matriz de covarianzas. En primer lugar, calculamos el número de observaciones y el vector de medias.

```
> n <- nrow(dat)
> apply(dat, 2, mean)
```

```
##   prog ingcom ingsof   sist  estad
##  36.76  50.60  50.68  46.04  43.80
```

A continuación, calculamos la matriz de covarianzas muestral, sus autovalores y autovectores y comprobamos que coinciden con los resultados de `test.pca$sdev^2` y `test.pca$loadings`

```
> S <- cov(dat) * (n - 1)/n
> auto <- eigen(S)
> lambda <- auto$values
> lambda
```

```
## [1] 811.66184  81.63974  43.57261  37.62083  13.86339
```

```
> v <- auto$vectors
> v
```

```
##            [,1]       [,2]       [,3]        [,4]        [,5]
## [1,] -0.5982782  0.6745404  0.1852556 -0.38597894  0.06131111
## [2,] -0.3607532  0.2450733 -0.2490064  0.82871854 -0.24701742
## [3,] -0.3021774 -0.2140882 -0.2114109  0.13484564  0.89441442
## [4,] -0.3890403 -0.3384022 -0.6999921 -0.37537871 -0.32129949
## [5,] -0.5188995 -0.5697232  0.6074477  0.07178665 -0.17892129
```

A continuación, calculamos la proporción de varianza explicada y sus valores acumulados.

```
> lambda/sum(lambda)
```

```
## [1] 0.82122218 0.08260135 0.04408584 0.03806395 0.01402668
```

```
> cumsum(lambda/sum(lambda))
```

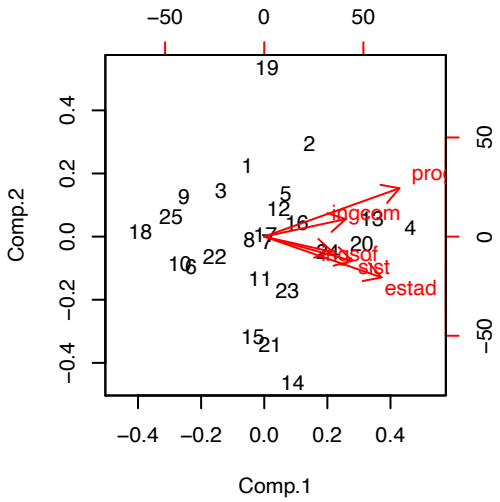
```
## [1] 0.8212222 0.9038235 0.9479094 0.9859733 1.0000000
```

3 El biplot

El biplot es una representación gráfica simultánea de los individuos (mediante puntos) y las variables (mediante flechas), en un mismo sistema de coordenadas bidimensional construido en base a las dos primeras componentes principales. Permite interpretar el significado de las componentes (la primera en el eje horizontal

y la segunda en el eje vertical) en base a las direcciones de las flechas. A su vez, se valoran como parecidos los individuos cuyos puntos están próximos en el biplot. De igual modo tendrán correlación positiva las variables con flechas semejantes. Asimismo, los individuos que se encuentran en la dirección de cierta flecha tendrán observaciones altas en la variable representada por la flecha.

```
> biplot(test.pca)
```



4 Las componentes principales y los cambios de escala

Un problema importante del análisis de componentes principales es que sus resultados dependen de la escala de medida de las variables originales. Así, si se cambia de escala una variable (dejando las demás fijas) las componentes principales se modifican, cambiando incluso la posible interpretación de las mismas. En concreto, si se aumenta la escala de una variable, ésta verá incrementada su varianza y su aportación proporcional a la varianza total, atrayendo de este modo a la primera componente principal, que se asemejará a esta variable.

Claro está que si se realiza el mismo cambio de escala en todas las variables, entonces los resultados del análisis de componentes principales se mantendrán idénticos.

Este problema se puede solventar de dos maneras:

- Midiendo todas las variables en la misma escala (siempre que sean de la misma naturaleza).
- Aplicando el análisis de componentes principales a las variables estandarizadas. Esto último equivale a trabajar con la matriz de correlaciones, en lugar de la matriz de covarianzas.

El archivo `decatlon.txt` contiene los resultados de 33 participantes en la prueba combinada de atletismo durante una competición. Las filas corresponden a los participantes y las columnas a los resultados en las diez pruebas (cuatro carreras, tres lanzamientos y tres saltos). Las variables correspondientes son: 100 metros (X100), salto de longitud (long), lanzamiento de peso (poid), salto de altura (haut), 400 metros (X400), 110-metros vallas (X110), lanzamiento de disco (disq), salto con pértiga (perc), lanzamiento de jabalina (jave) y 1500 metros (X1500).

1. Realiza un Análisis de Componentes Principales con los datos de decatlón. Justifica el uso de la matriz de covarianzas o de la matriz de correlaciones muestrales para llevar a cabo el análisis.
2. Haz una interpretación de las dos primeras componentes principales. ¿Cuál es la proporción de variabilidad explicada por las dos primeras componentes principales?
3. Realiza el biplot correspondiente y comenta la gráfica obtenida.