Prácticas de Aprendizaje Automático

Parte 1: Introducción a R

Acid S., email: acid@decsai.ugr.es

Curso 2017-18

1. El entorno R. Motivos para su utilización
2. Distribución e instalación de R
3. Primeros pasos
4. Pidiendo ayuda en R
5. Salvar e iniciar sesiones anteriores
6. Instalación y uso de paquetes (librerías) de R
7. Objetos de R y atributos
8. Vectores y and Listas
9. Matrices
10. Dataframe
11. Datos perdidos
12. Indexando o extrayendo subconjuntos
13. Eliminando datos perdidos

Índice

El entorno R. Motivos para su utilización

- R es, un lenguaje que permite implementar técnicas de aprendizaje de ML y estadísticas.
- Es, a la vez:
  - un entorno interactivo para el análisis estadístico y gráfico,
  - un lenguaje de programación interpretado de alto nivel, con funciones orientadas a objetos y, de sintaxis simple e intuitiva.
- El diseño de R se ha inspirado en S (Becker, Chambers and Wilks) y en Scheme (Sussman).
- Software de libre distribución, bajo GNU General Public License.

Motivos para su utilización

- La estructura y facilidad de uso de R nos permite implementar nuestras propias funciones y rutinas según se vaya necesitando.
- Puede utilizarse para realizar gráficos de alta calidad de enorme utilidad en los trabajos de investigación.
- Uno de los mejores softwares integrados actuales y resulta que es nuestro por nada, *es gratuito.*
- Cada vez más gente presenta sus resultados en el contexto de R. Basta mirar cualquier disciplina y ver quien está utilizando R.
- La distribución de R viene acompañada de un numeroso conjunto de funciones (librería base). Sin embargo, existen de libre disposición numerosas librerías específicas con las últimas técnicas disponibles (y además con explicación de su uso).

Índice

Para instalar R hay que bajar un fichero ejecutable de la página web del proyecto R

url

`http://www.r-project.org/`

En esta página debemos hacer clic en download CRAN, escoger a continuación uno de los servidores (CRAN Mirrors en SPAIN)
En fechas recientes, la versión disponible de R, R-3.3.1 para Windows (32/64 bit)... actualmente la R-3.4.3
Puede seguirse con más detalle en el cuaderno de prácticas...

Para un desarrollo más cómodo de las prácticas y de los trabajos a entregar, vamos a instalar también otro software **Rstudio** y *RMarkdown*.

url

`https://www.rstudio.com/products/rstudio/download`

Puede seguirse con más detalle en el cuaderno de prácticas...

Índice

En el prompt de R podemos escribir expresiones matemáticas que se evalúan como en una calculadora.

```
> 2*pi*1.5            # existe pi ya definida
[1] 9.424778
```

Cuando se introduce una expresión, se evalúa y, devuelve resultado o bien se utilizan en la parte derecha de la **asignación**, mediante el uso de $<-$.

```
> perimetro <- 2*pi*1.5
> mensaje <- "Esto es una cadena"
> print(perimetro)                  # muestra el valor de pe
[1] 9.424778
> mensaje                 # auto impresion de mensaje
[1] "Esto es una cadena"
```

En la asignación hay parte izquierda y derecha, la derecha se evalúa primero y el resultado se almacena en una variable declarada implícitamente. *perimetro* es un vector, cuyo primer valor es 9.424778.

Algunas funciones

Cualquier función matemática que se nos ocurra está en R y podemos aplicarla sobre cualquier número, algunos ejemplos:

```
x <- 1.8

log(x),   exp(x),  log(x,n),  log10(x),  sqrt(x),
factorial(x),  choose(n,x),   gamma(x),  lgamma(x),
floor(x), ceiling(x), trunc(x), round(x,digits=0),
signif(x,digits=6), cos(x),  sin(x),  tan(x),  acos(x)
asin(x),  atan(x), acosh(x), asinh(x), atanh(x),
abs(x)

> trunc(x)
[1] 1
> round(x)
[1] 2
exp(x)
[1] 6.049647
```

Índice

Pidiendo ayuda en R

Si se quiere saber más acerca de una función, además de las opciones de menú propias de R, desde la línea de comandos, se puede acceder a información acerca de:
forma de uso de una función, número y tipo de los parámetros, *help*() o bien algo relacionado con un concepto *apropos*() o bien muestras de ejecución *example*().
Ej. El comando **ls()** muestra los objetos ya creados en el entorno.

```
> ? ls
> help(ls)
> apropos("vector")
> args(vector)
> example(log)
> example(plot)
```

Índice

Salvar e iniciar sesiones anteriores

- Desde el menú `Archivo> guardar` se puede guardar una imagen del espacio de trabajo realizado hasta ese momento. La extensión para estos archivos es `.RData`.
- Para salir de R podemos seleccionar Salir del menú `Archivo`, o bien ejecutar la orden **q()**. Antes de salir, R pregunta si se quiere guardar el actual espacio de trabajo.
- Podemos retomar el espacio de trabajo de otro día abriendo desde el menú `Archivo> cargar` área de trabajo para restaurar los objetos creados en otras sesiones.
- Es posible recuperar los últimos comandos ejecutados en la sesión con **history()**.

Índice

Instalación de paquetes (librerías) de R

- Desde R en `Paquetes > Instalar paquetes...` ... nos pide seleccionar el CRAN mirror, por comodidad podemos elegir el que hay en España. Vamos a instalar el paquete `ISLR`.
- También podemos utilizar la función **install.packages()**

La instalación no implica que los paquetes ya puedan ser utilizados. Es necesario **cargar** las librerías antes de empezar a usarlas. Lo mismo ocurre con las librerías existentes en la versión local de R.

Incorporando las librerías en la sesión de R

- Desde el menú en `Paquetes > cargar paquetes...` se selecciona la librería `ISLR` de entre las disponibles
- bien desde la línea de comandos utilizando la función **library()**

Qué contiene?
Para saberlo hay que utilizar las funciones **library** y **help**

```
library(help="ISLR")
```

Se abre una ventana de ayuda sobre la librería en cuestión.
Cuestión: ¿Qué es Auto?

Índice

Los objetos

R tiene 5 clases de objetos básicos simples

- character
- numeric (real)
- integer
- complex
- logical (TRUE/FALSE) o bien (T/F)

Los objetos básicos no simples para R son:

- el vector: solo puede contener objetos del mismo tipo.
- listas: puede contener objetos de distinto tipo.
- dataframe: tablas de datos son estructuras similares a una relación de Base de Datos.
- factores : sirven para representar datos categóricos.
- funciones : permiten extender las capacidades de R fácilmente.
- definidos por el usuario.

Sobre números

Los números en R por defecto se tratan como objetos de tipo *numeric* reales de doble precisión. Si se quiere explícitamente un entero hay que especificar el sufijo L .
Para conocer el tipo de un objeto se puede usar la función *class*().

```
> class(1)
[1] "numeric"
> class(1L)
[1] "integer"
```

Existe un número especial *Inf* que representa infinito

```
 1/0
```

El número *Inf* se puede utilizar en los cálculos

```
  1 / Inf
```

El valor *NaN* representa un valor indefinido, ("not a number"); también se puede considerar como valor perdido ("missing value").

Atributos de los objetos

Los objetos de R tienen características o atributos (no todos las mismas) como

- class
- length
- dimensiones (matrices, arrays)
- nombres
- atributos definidos por usuario

Algunos de esos atributos se pueden consultar mediante funciones *length*(), *mode*(), *dim*(),

```
> class(mensaje)
[1] "character"
```

Atributos de los objetos

Los atributos son distintos según el tipo de objeto.

Algunos de los objetos más complejos, como la salida de un método, se puede consultar con *attributes*()

Atributos	Objetos
mode	todos
storage.mode	todos de modo numérico
length	todos
names	vectores y listas
dim	matrices y arrays
dimnames	matrices y arrays
levels	factores
class	cualquier clase

Índice

Vectores

Cómo se crea un vector: Utilizando la función *c*()

```
> x <- c(0.1, 0.6)     ## numeric
> x <- c(TRUE, FALSE)     ## logical
> x <- c(T, F)         ## logical
> x <- c("a", "b", "c") ## character
> x <- c(1+0i, 2+4i)     ## complex
```

o bien usando la función *vector*()

```
> v <- vector("numeric", length = 10)
> v
 [1] 0 0 0 0 0 0 0 0 0 0
```

Creando secuencias

El operador **:** se usa para crear una secuencia de enteros crecientes o decrecientes

```
> x <- 100:120
> x
 [1] 100 101 102 103 104 105 106 107 108 109 110 111 1
[20] 119 120
```

x es un vector, cuyo primer valor es 100 ... cada línea identifica la posición del primer elemento

```
> y <- 20:10
> y
 [1] 20 19 18 17 16 15 14 13 12 11 10
```

Creando secuencias, con salto diferentes

Si la secuencia es regular pero con saltos distintos de 1 o -1 se usa el comando **seq()**

```
> seq(1, 9, by = 2)
> seq(0, 1, length.out = 11)
> s <- seq(1.575, 5.125, by = 0.05)
```

x es un vector, cuyo primer valor es 100 ... cada línea identifica la posición del primer elemento

```
> y <- 20:10
> y
 [1] 20 19 18 17 16 15 14 13 12 11 10
```

Cambiando tipos de los objetos

Coerción de forma implícita:
Cuando se mezclan diferentes tipos de objetos en un vector, si fuerzan (coerción) todos los componentes a un mismo tipo.

```
> y <- c(1.7, "a")     ## character
> y <- c(TRUE, 2)     ## numeric
> y <- c("a", TRUE) ## character
```

Coerción de forma explícita:

```
> n <- 0:7    # enteros
> as.logical(n)
[1] FALSE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
> as.character(n)
[1] "0" "1" "2" "3" "4" "5" "6" "7"
```

Cambiando tipos de los objetos

Coerción de forma explícita: (con menos éxito)
Cuando no procede la coerción, devuelve NAs.

```
> m <- c("a", "b", "c")
> as.numeric(m)
[1] NA NA NA
Warning message:
NAs introducidos por coercion
> as.logical(m)
[1] NA NA NA
> as.complex(m)
[1] NA NA NA
Warning message:
NAs introducidos por coercion
```

Listas

Las listas son un tipo especial de vector que contiene elementos de diferentes tipos, son un tipo frecuente en R. Se usas por ejemplo en los resultados de los métodos de aprendizaje que devuelven un conjunto de valores numéricos y descriptivos.

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

[[2]]
[1] "a"

[[3]]
[1] TRUE

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

Índice

Matrices

Matrices son vectores con un atributo dimensión. El atributo dimensión es un vector de enteros de longitud 2 (nrow, ncol)

```
m <- matrix(nrow = 2, ncol = 3)
 m
[,1] [,2] [,3]
[1,] NA NA NA
[2,] NA NA NA
 dim(m)
[1] 2 3
attributes(m)
$dim
[1] 2 3
```

Matrices +

Las matrices se pueden construir y rellenar de forma cómoda empezando por la coordenada (1,1) con una secuencia de valores, bien por filas bien por columnas. Por defecto es por columnas. *args(matrix) byrow = FALSE*

```
 m <- matrix(1:6, nrow = 2, ncol = 3)
 m
      [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

 m <- matrix(1:6, nrow = 2, ncol = 3, byrow = TRUE)
 m
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

Matrices ++

Las matrices se pueden crear apartir de un vector y luego añadir el atributo de dimensión

```
 m <- 1:10
 m
[1] 1 2 3 4 5 6 7 8 9 10

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

Matrices +++. Pegando por filas o columnas

Las matrices se pueden crear apartir de dos vectores pegándolas por filas o por columnas, column-binding or row-binding con **cbind()** y **rbind()**.

```
> x <- 1:3
> y <- 10:12
> cbind(x, y)
     x  y
[1,] 1 10
[2,] 2 11
[3,] 3 12
> m1 <- cbind(x, y)

> m2 <- rbind(x, y)
> m2
  [,1] [,2] [,3]
x    1    2    3
y   10   11   12
```

Índice

Dataframe

Los Dataframes se usan para almacenar datos de tablas

- Se representan como un tipo especial de lista donde cada componente de la lista tiene la misma longitud.
- Cada elemento de la lista puede verse como una columna y la longitud de cada componente es el número de filas.
- A diferencia de las matrices se pueden almacenar diferentes clases de objetos en cada columna (como las listas). En matrices todos los componentes son del mismo tipo.
- Los Data frames tienen un tipo especial de atributo el **row.names**
- Los Data frames se crean utilizando la funcion **read.table()** o **read.csv()**
- Un data frame puede convertirse en una matriz mediante **data.matrix()**

Dataframe

Vamos a trabajar con el dataframe Auto del paquete ISLR

- Para tener una descripción detallada de los datos que contiene **summary(Auto)**
- Para ver los primeros y los últimos datos de la tabla, usar **head()** y **tail()** respectivamente.

Dataframe.

Creando un dataframe

```
> dd <- data.frame(col1 = 1:4, col2 = c(T, T, F, F))
> dd
  col1  col2
1    1  TRUE
2    2  TRUE
3    3 FALSE
4    4 FALSE
>dim(dd)
[1] 4 2
> nrow(dd)
[1] 4
> ncol(dd)
[1] 2
```

Dando nombres, otros atributos

Los objetos de R pueden tener nombres, (además del identificador) lo que les hace más descriptivos

```
> notas <- c(3.5,7.2,7.3)
> names(notas)
NULL
> names(notas) <- c("part1", "part2", "teo")
> notas
part1 part2   teo
  3.5   7.2   7.3
> names(notas)
[1] "part1" "part2" "teo"
```

Nombres para los objetos

Las listas también pueden tener nombres para los componentes

```
> l <- list(a = 1, b = 2, c = 3)
> l
$a
[1] 1
$b
[1] 2
$c
[1] 3
```

y también las matrices. Filas + columnas

```
> m <- matrix(1:4, nrow = 2, ncol = 2)
> dimnames(m) <- list(c("a", "b"), c("c", "d"))
> m
  c d
a 1 3
b 2 4
```

Índice

Datos perdidos

Los datos perdidos *missing values* se notan mediante NA o NaN , están indefinidos para cualquier operación matemática.

- se usa is.na() para comprobar si un objeto contiene algún NA
- se usa is.nan() para comprobar si un objeto contiene algún NaN
- puede haber valores NA en cualquier tipo de dato, hay enteros NA, caracter NA etc.
- Todos los NaN (not a number) tienen el valor NA (desconocido) pero al revés no es cierto

```
> x <- c(1, 2, NA, 10, 3)
> is.na(x)
[1] FALSE FALSE TRUE FALSE FALSE
> is.nan(x)
[1] FALSE FALSE FALSE FALSE FALSE
> z <- c(1, 2, NaN, NA, 4)
> is.na(z)
[1] FALSE FALSE TRUE TRUE FALSE
> is.nan()
[1] FALSE FALSE TRUE FALSE FALSE
```

Índice

Indexando o extrayendo subconjuntos

Se puede acceder a componentes parciales de objetos, para ello existe un conjunto de operadores que permiten extraer subconjuntos

- []
  siempre devuelve un objeto del mismo tipo que el objeto original, y permite seleccionar uno o más componentes.
- `[ [ ] ]`
  se usa para extraer componentes de una lista o un dataframe; tan solo puede usarse para extraer un único componente y el tipo del objeto devuelto es el tipo del componente del objeto original.
- $
  se usa para extraer componentes de una lista o dataframe de forma similar al anterior, pero solo con nombres (no posiciones).

Indexando en vectores

```
> letras <- c("a", "b", "c", "c", "d", "a")
> letras[1]
[1] "a"
> letras[2]
[1] "b"
> letras[1:4]
[1] "a" "b" "c" "c"
> letras[ letras > "a"]
[1] "b" "c" "c" "d"
> u <- letras > "a"
> u
[1] FALSE TRUE TRUE TRUE TRUE FALSE
> letras[u]
[1] "b" "c" "c" "d"
```

Indexando en listas

```
> l
[[1]]
[1] 1

[[2]]
[1] "a"

[[3]]
[1] TRUE

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

> l[[3]]
[1] TRUE
> l2 <- list(orden = 1:4, bar = 0.6, baf = "hello")
> l2 [c(1, 3)]
$orden
[1] 1 2 3 4
$baf
[1] "hello"
```

Indexando en una matriz

Las matrices se pueden indexar en la forma habitual (i, j) filas, columnas.

```
> m <- matrix(1:6, 2, 3)
> m[1, 2]
[1] 3
> m[2, 1]
[1] 2
```

Los índices pueden estar ausentes

```
> m[1, ]
[1] 1 3 5
> m[, 2]
[1] 3 4
```

Indexando en una matriz

Por defecto, cuando se selecciona un único componente de la matriz, éste devuelve un vector de length 1 en lugar de una matriz de $1x1$. Este comportamiento se puede modificar fijando `drop=FALSE`

```
> mm <- matrix(1:6, 2, 3)
> mm[1, 2]
[1] 3
> mm[1, 2, drop = FALSE]
     [,1]
[1,]    3
```

De forma similar, extraer una columna o una fila en una matriz, devuelven un vector no una matriz por defecto

```
> mm <- matrix(1:6, 2, 3)
> mm[1, ]
[1] 1 3 5
> mm[1, , drop = FALSE]
     [,1] [,2] [,3]
[1,]    1    3    5
```

Indexando ++

- en dataframe

```
> dd[[2]]
[1]  TRUE  TRUE FALSE FALSE
> dd$col1
[1] 1 2 3 4
```

- de forma parcial

A la hora de indexar utilizando el nombre se puede hacer de forma aproximada o relajada, mientras no sea ambiguo

```
> ll <- list(aardvark = 1:5)
> ll$a
[1] 1 2 3 4 5
> ll[["a"]]
NULL
> ll[["a", exact = FALSE]]
[1] 1 2 3 4 5
```

Índice

Eliminando datos NA, perdidos

Una tarea frecuente es eliminar datos perdidos

```
> x <- c(1, 2, NA, 4, NA, 5)
> bad <- is.na(x)
> x[!bad]
[1] 1 2 4 5
```

Qué pasa cuando tenemos diferentes variables y queremos eliminar los NA de varios simultáneamente?

```
> varx <- c(1, 2, NA, 4, NA, 5)
> vary <- c("a", NA, "c",  "d", NA, "f")
> good <- complete.cases(varx, vary)
> good
[1]  TRUE FALSE FALSE  TRUE FALSE  TRUE
> varx[good]
[1] 1 4 5
> vary[good]
[1] "a" "d" "f"
```