Fundamentos de R

Code

1. Introducción

- En el siguiente documento presentaremos los fundamentos básicos de R.
- Usaremos R para implementar los algoritmos básicos de machine learning que estudiaremos el curso.
- Esta, y las clases que vienen, serán desarrolladas en R Markdown.
- A su vez, utilizaremos R Studio para ejecutar R.
- R Studio es una interface poderosa (tecnicamente una Integrated Development Environment -IDE) que facilita la utilizacion de R, permite introducir códigos y diversas herramientas de visualización.
- La ventaja de todos estos paquetes es que son open source.
- Lo primero que debemos hacer es estar seguros que trabajamos en una carpeta conocida.
- Para ello, creemos en nuestro computador una carpeta llamada “Semana 2 - Fundamentos de R”.
- Y seleccionemos dicha carpeta como el directorio de trabajo.
- Para ello vamos a `Session -> Set Working Directory -> Choose Directory` y seleccionamos la carpeta creada.
- R es una calculadora muy poderosa. Lo podemos usar para operaciones complejas, tipo:

Hide

Hide

```
2+2
```

```
[1] 4
```

- En adelante, lo que veamos en recuadro gris son comandos. En recuadro blanco aparecerán los resultados
- Además, hay una serie de funciones prederminadas que pueden implementarse fácilmente:

Hide

Hide

```
log(1)
```

```
[1] 0
```

Hide

Hide

```
15^2
```

```
[1] 225
```

Hide

Hide

```
sqrt(16)
```

```
[1] 4
```

2. Paquetes

- R funciona a base de paquetes. Una colección de funciones en R que pueden compartirse entre usarios es un paquete. Para lo que haremos de machine learning, existen paquetes gratuitos.
- Quizas la primera función que tenemos que aprender a utilizar es la que sirve para instalar paquetes.
- Haremos la prueba con el paquete `class`. Este paquete será usado mas adelante cuando veamos el algoritmo kNN.

Hide

Hide

```
install.packages("class")
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed

  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100 86276  100 86276    0     0   143k      0 --:--:-- --:--:-- --:--:--  143k
tar: Failed to set default locale
```

```
The downloaded binary packages are in
    /var/folders/cw/_9zpljws58jbbv49mcb86l6c0000gn/T//Rtmp4wL8Gz/downloaded_packages
```

- Con esto el paquete queda instalado. Para preservar memoria, R no carga todos los paquetes
- Debemos hacerlo manualmente. Lo podemos hacer de la siguiente forma

Hide

Hide

```
library(class)
```

- Esto es suficiente para que podamos usar el paquete. Por el momento no lo usaremos. Por tanto, revertimos la acción anterior

Hide

Hide

```
detach("package:class", unload=TRUE)
```

- Con eso el paquete ya no nos está quitando espacio.

3. Estructuras de datos en R

- Para trabajar en big data, naturalmente, hay que dominar los datos. Es necesario saber manejarlos y entenderlos.
- Una gran porción del tiempo se dedica a limpiar los datos.
- Empezaremos entendiendo las estructuras con las que R entiende los datos. Para ello, estudiaremos:
    - Vectores
    - Factores
    - Listas
    - Data Frames
    - Matrices

3.1 Vectores

- Los vectores son la estructura fundamental en R.
- Sirven para almacenar un conjunto ordenado de valores llamados elementos.
- Todos los elementos deben ser del mismo tipo.
- Hay diferentes tipos de elementos muy usados en machine learning, como son:
    - `integer` : número sin decimales
    - `double` : número con decimales
    - `character` : texto
    - `logical` : Valores `TRUE` o `FALSE`
- Hay dos valores espaciales:
    - `NULL` indica la ausencia de cualquier valor
    - `NA` es un missing value
- Podemos crear nuestros propios vectores fácilmente
- Para esto, el operador flecha es clave: `<-` . Sirve para asignar valores.
- Por ejemplo, creemos un vector de nombres:

Hide

Hide

```
nombre_sujeto <- c("James", "Radamel", "David")
nombre_sujeto
```

```
[1] "James"   "Radamel" "David"
```

- La última linea nos muestra el vector, que en este caso es `string` .
- Creemos ahora un vector `integer` con el número de goles de cada uno:

Hide

Hide

```
numero_goles <- c(109, 249, 0)
```

- Ahora creemos un vector `logical` indicando si el jugador juega en Francia

Hide

Hide

```
juega_francia <- c(FALSE, TRUE, FALSE)
```

- Podemos acceder a elementos de un vector de acuerdo con su orden en el vector y usando brackets (esto es, [y]).
- Por ejemplo, podemos averiguar cuántos goles tiene Radamel:

Hide

Hide

```
numero_goles[2]
```

```
[1] 249
```

- O si James juega en Francia

Hide

Hide

```
juega_francia[1]
```

```
[1] FALSE
```

- Pero podemos extraer porciones mas grandes de datos. Los dos puntos, `:`, son muy útiles para esto.
- Por ejemplo, si queremos saber cuántos goles tienen Radamel y David podemos escribir:

Hide

Hide

```
numero_goles[2:3]
```

```
[1] 249   0
```

- O los goles de todos, excepto David, excluyendo el tercer elemento del vector:

Hide

Hide

```
numero_goles[-3]
```

```
[1] 109 249
```

- Esto lo podemos hacer también por medio de un vector `logical` indicado si incluimos cada elemento o no:

Hide

Hide

```
numero_goles[c(TRUE, TRUE, FALSE)]
```

```
[1] 109 249
```

- Los vectores son la base de muchas otras estructuras en R. Entenderlos es clave.
- Podemos hacer operaciones típicas entre vectores:

Hide

Hide

```
a <- c(1,2,3)
b <- c(4,5,6)
c <- a+b
c
```

```
[1] 5 7 9
```

- Creamos dos vectores y definimos un tercero como la suma de estos dos.
- Fácilmente hacemos multiplicación escalar:

Hide

Hide

```
d <- 5*a
d
```

```
[1]  5 10 15
```

- Podemos trasponer un vector para obtener un vector columna

Hide

Hide

```
e <- t(a)
e
```

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

- Y podemos multiplicar vectores

Hide

Hide

```
f <- a*b
f
```

```
[1]  4 10 18
```

Hide

Hide

```
g <- e %*% a
g
```

```
     [,1]
[1,]   14
```

3.2 Factores

- En machine learning las variables categóricas son clave.
- Sus elementos son nominales.
- R provee una estructura específica para este propósito.
- Un **factor** es un vector usado para representar variables categóricas u ordinales.
- En el caso de los futbolistas, podemos crear un factor que indique la posición en la que juega

Hide

```
posicion <- factor(c("volante", "delantero", "arquero"))
posicion
```

```
[1] volante   delantero arquero
Levels: arquero delantero volante
```

- Nótese que los niveles nos indican las categorias contenidas en el vector.
- Pero sabemos que nos falta una posición: defensa.
- La podemos incluir de la siguiente forma:

Hide

Hide

```
posicion <- factor(c("volante", "delantero", "arquero"), levels = c("arquero", "defen
sa", "volante", "delantero"))
posicion
```

```
[1] volante   delantero arquero
Levels: arquero defensa volante delantero
```

- De hecho, si la variable `factor` es ordinal a través de `levels` podemos expresar dicho orden.
- Por ejemplo, supongamos que los jugadores se lesionan y que clasificamos el tipo de lesión en tres categorias:

Hide

Hide

```
gravedad_lesion <- factor(c("severa", "moderada", "leve"), levels = c("leve", "modera
da", "severa"), ordered=TRUE)
gravedad_lesion
```

```
[1] severa   moderada leve
Levels: leve < moderada < severa
```

- En este caso la magia la hace el parámetro `ordered` .
- Esto es bastante útil para hacer tests logicos. Por ejemplo, podemos preguntarle a R qué jugadores tienen una lesión severa:

Hide

Hide

```
gravedad_lesion > "moderada"
```

```
[1]  TRUE FALSE FALSE
```

- El pobre James, en este ejemplo.
- Los factores ordenados serán clave en machine learning.

3.3 Listas

- Una lista es una estructura de datos usada para almacenar un conjunto ordenado de elementos.
- La diferencia con un vector es que la lista permite elementos de diferente tipo.
- Consideremos el ejemplo de los futbolistas. Si quisiéramos desplegar toda la informacion de James, tendríamos que hacerlo por separado:

Hide

Hide

```
nombre_sujeto[1]
```

```
[1] "James"
```

Hide

Hide

```
numero_goles[1]
```

```
[1] 109
```

Hide

Hide

```
posicion[1]
```

```
[1] volante
Levels: arquero defensa volante delantero
```

Hide

Hide

```
juega_francia[1]
```

```
[1] FALSE
```

Hide

Hide

```
gravedad_lesion[1]
```

```
[1] severa
Levels: leve < moderada < severa
```

- Esto no parece muy eficiente.
- Con una lista podemos agrupar toda la información de James en un solo objeto.
- Para ello, usamos la función `list()`:

Hide

Hide

```
sujeto1 <- list(nombre = nombre_sujeto[1], numero_goles=numero_goles[1], posicion=pos
icion[1], juega_francia=juega_francia[1], gravedad_lesion=gravedad_lesion[1])
```

- Y el resultado es:

Hide

Hide

```
sujeto1
```

```
$nombre
[1] "James"

$numero_goles
[1] 109

$posicion
[1] volante
Levels: arquero defensa volante delantero

$juega_francia
[1] FALSE

$gravedad_lesion
[1] severa
Levels: leve < moderada < severa
```

- Podemos acceder a los elementos de una lista usando la posicion del elemento o el nombre usado para dicho elemento:

Hide

Hide

```
sujeto1[1]
```

```
$nombre
[1] "James"
```

Hide

Hide

```
sujeto1$nombre
```

```
[1] "James"
```

- Es mucho más confiable recuperar el elemento por medio de su nombre que de su posición.
- Podemos además recuperar vario items de la lista, por medio de un vector de nombres:

Hide

Hide

```
sujeto1[c("nombre", "numero_goles")]
```

```
$nombre
[1] "James"

$numero_goles
[1] 109
```

- Podemos hacer lo mismo para los otros dos jugadores

Hide

Hide

```
sujeto2 <- list(nombre = nombre_sujeto[2], numero_goles=numero_goles[2], posicion=pos
icion[2], juega_francia=juega_francia[2], gravedad_lesion=gravedad_lesion[2])
sujeto3 <- list(nombre = nombre_sujeto[3], numero_goles=numero_goles[3], posicion=pos
icion[3], juega_francia=juega_francia[3], gravedad_lesion=gravedad_lesion[3])
```

- Podríamos construir una base de datos entera por medio de listas y de listas de listas.
- Por ejemplo, una lista para cada jugador y una lista que sea la unión de estas listas.
- Sin embargo, R nos da una estructura de datos especial para hacer esto.

3.4 Data Frames

- La estructura de datos en R mas importante para hacer machine learning son los data frames.
- Es una estructura análoga a un spreadsheet (`.xls`) en `Excel` o base de datos (`.dta`) en `Stata` .
- Se compone de filas y columnas.
- Puede entenderse como una lista de vectores o factores, cada uno con el mismo número de valores.
- Creemos un data frame para nuestros jugadores.
- Para esto, podemos usar los vectores que tenemos o las listas que creamos hace un rato.
- Usemos los vectores:

Hide

Hide

```
jugadores_seleccion <- data.frame(nombre_sujeto, numero_goles, posicion, juega_franci
a, gravedad_lesion, stringsAsFactors=FALSE)
```

- Nótese que incluímos el parametros `stringsAsFactors=FALSE` al final.
- Con esto evitamos que R convierta en `factor` cualquier variable `string` .
- Por ejemplo, no tendría sentido convertir `nombre_sujeto` en una variable categórica.
- Podemos observar nuestra base de datos:

Hide

Hide

```
jugadores_seleccion
```

nombre_sujeto	**numero_goles**	**posicion**
<chr>	<dbl>	<fctr>

James	109	volante
Radamel	249	delantero
David	0	arquero

3 rows | 1-3 of 5 columns

- Es clara la estructura bidimensional (filas y columnas) del data frame.
- Está en formato matriz.
- Una columna para cada vector de características. Y un vector para cada jugador.
- Las columnas son los atributos y las filas los ejemplos, en lenguaje de machine learning.
- Podemos extraer elementos enteros de un data frame.
- Por ejemplo, toda una columna (vector):

Hide

Hide

```
jugadores_seleccion$nombre_sujeto
```

```
[1] "James"   "Radamel" "David"
```

- Pero también podemos extraer varias columnas del data frame:

Hide

Hide

```
jugadores_seleccion[c("nombre_sujeto", "numero_goles")]
```

nombre_sujeto <chr>	**numero_goles** <dbl>
James	109
Radamel	249
David	0

3 rows

- También lo hubieramos podido obtener invocando las posiciones en el data frame:

Hide

Hide

```
jugadores_seleccion[1:2]
```

nombre_sujeto <chr>	**numero_goles** <dbl>
James	109
Radamel	249
David	0

3 rows

- Pero es recomendable hacerlo de la primera forma, porque las bases pueden cambiar de estructura y de orden.
- Podemos obtener elementos específicos de la matriz, simplemente indicando la posición que nos interesa.
- Para esto, usamos la sintaxis `[rows, columns]`. Es decir, primero la fila deseada, luego la columna.
- Por ejemplo, si deseamos saber la posición de Radamel, usamos:

Hide

Hide

```
jugadores_seleccion[2,3]
```

```
[1] delantero
Levels: arquero defensa volante delantero
```

- Podemos obtener particiones de la matriz, indicando las filas y las columnas que deseamos.
- Por ejemplo, si deseamos las filas 1 y 3 (James y David) y las columnas 2 y 4 (numero de goles y si juegan en francia), escribimos:

Hide

Hide

```
jugadores_seleccion[c(1,3), c(2,4)]
```

	numero_goles <dbl>	juega_francia <lgl>
1	109	FALSE
3	0	FALSE

2 rows

- También podemos extraer toda una fila o toda una columna. Por ejemplo, toda la primera columna (nombres):

Hide

Hide

```
jugadores_seleccion[, 1]
```

```
[1] "James"   "Radamel" "David"
```

- O toda la primera fila (datos de James)

Hide

Hide

```
jugadores_seleccion[1,]
```

	nombre_sujeto <chr>	numero_goles <dbl>	posicion <fctr>
1	James	109	volante

1 row | 1-4 of 5 columns

- La clave es dejar en blanco la correspondiente fila o columna

3.5 Matrices

- Además de los `data frames` , R tiene otras estructuras para almacenar valores en una forma tabular.
- Una matriz es una estructura de datos que representa una tabla bidimensional con filas y columnas de datos.
- Suelen ser usadas para operaciones matemáticas, luego suelen tener valores numéricos.
- Para crear una matriz debemos ofrecer un vector de datos a la función matriz y un parámetro especificando el número de filas o de columnas.
- Por ejemplo, creemos una matriz `2x2` que almacene los numeros del 1 al 4.

Hide

Hide

```
m <- matrix(c(1,2,3,4), nrow=2)
m
```

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

- También la podríamos haber definido según el número de columnas:

Hide

Hide

```
m <- matrix(c(1,2,3,4), ncol=2)
m
```

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

- Y obtenemos la misma matriz. Nótese que R carga la primera columna primero, y luego la segunda.
- Este es el default. Lo podemos revertir con la opción `byrow`

Hide

Hide

```
m <- matrix(c(1,2,3,4), ncol=2, byrow=TRUE)
m
```

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

- Veamos otro ejemplo que ilustra el default:

Hide

Hide

```
m <- matrix(c(1,2,3,4,5,6), nrow=2)
m
```

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

Hide

Hide

```
m <- matrix(c(1,2,3,4,5,6), ncol=2)
m
```

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

Hide

Hide

```
m <- matrix(c(1,2,3,4,5,6), nrow=2, byrow=TRUE)
m
```

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

Hide

Hide

```
m <- matrix(c(1,2,3,4,5,6), ncol=2, byrow=TRUE)
m
```

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

- Como con data frames, podemos extraer elementos particulares de una matriz:

Hide

```
m[1,2]
```

```
[1] 2
```

Hide

Hide

```
m[2,]
```

```
[1] 3 4
```

Hide

Hide

```
m[,2]
```

```
[1] 2 4 6
```

4. Trabajando con datos en R

- Uno de los mayores desafíos de cualquier proyecto es trabajar con los datos.
- Obtenerlos, prepararlos y usarlos, sobre todo porque vienen de diferentes fuentes.
- Estaremos lidiando con esto a lo largo de todo el curso.
- Pero veremos a continuación algunas funciones básicas.

4.1 Guardando, cargando y removiendo estructuras de datos

- La función `save()` nos permite guardar una estructura de datos, para poder cargarla despues.
- La función guarda la estructura en la ubicación especificada por el parametro `file`.
- La extensión usada para los archivos de datos es `.RData`.
- Supongamos que tenemos tres objetos: x, y, z.
- Pueden ser vectores, matrices, listas, data frames, etc.
- Los podemos guardar en un archivo llamado `mydata.Rdata`. Probemos con algunos de los objetos que ya tenemos:

Hide

Hide

```
save(jugadores_seleccion, m, sujeto1, file="mydata.RData")
```

- Y podemos revisar en nuestra carpeta que en efecto ahora tenemos el archivo `mydata.RData`.
- Si tenemos abierta una sesión de R, y queremos cargar una estructura de datos que hayamos guardado, usamos la función `load()`.
- Luego de trabajar en una sesión, suelen acumularse muchos objetos.
- La función `ls()` regresa un vector con todas las estructuras de datos actualmente en la memoria:

Hide

Hide

```
ls()
```

```
 [1] "a"                  "b"
 [3] "c"                  "d"
 [5] "e"                  "f"
 [7] "g"                  "gravedad_lesion"
 [9] "juega_francia"      "jugadores_seleccion"
[11] "m"                  "nombre_sujeto"
[13] "numero_goles"       "posicion"
[15] "sujeto1"            "sujeto2"
[17] "sujeto3"
```

- Por cuestiones de memoria, quisiéramos remover algunas de estas esctructuras.
- Lo podemos hacer con la función `rm()`:

Hide

Hide

```
rm(a,b,c,d,e,f,g)
```

- Y verificamos que ya no estén:

Hide

Hide

```
ls()
```

```
[1] "gravedad_lesion"     "juega_francia"
[3] "jugadores_seleccion" "m"
[5] "nombre_sujeto"       "numero_goles"
[7] "posicion"            "sujeto1"
[9] "sujeto2"             "sujeto3"
```

- Incluso podríamos remover todos los objetos de un solo tajo con el código $rm = list(ls())$
- Por obvias razones, hay que tener mucho cuidado al hacer esto!

4.2 Trabajando con archivos CSV

- Muchas bases de datos están en formato de texto.
- Esto tiene muchas ventajas.
- Se pueden leer en cualquier sistema operativo.
- Se exportan fácilmente a otros formatos.
- Suelen usarse los **tabulars**, que son archivos de datos en forma de matriz.
- Cada línea de texto refleja un ejemplo, y cada ejemplo tiene el mismo número de caracteristicas.
- En una fila las características son separadas por medio de un símbolo, llamado delimitador.
- Comunmente la primera fila del archivo se usa para escribir los nombres de las columnas. Esta fila es el **encabezado**.
- El archivo de texto tabular más común es el CSV (Comma-Separated Values).
- El delimitador es una coma.
- Veamos un ejemplo de cómo importar a R un CSV. En la carpeta tenemos un archivo llamado `usedcars.csv`.

Hide

Hide

```
usedcars <- read.csv("usedcars.csv", stringsAsFactors=FALSE)
```

- Esta base contiene datos de carros usados a la venta en USA de una popular página web.
- Podemos ver que los datos han sido cargados. Removamos el objeto

Hide

Hide

```
rm(usedcars)
```

- Por default, R asume que el CSV viene con un header. Si no es así, podemos usa el parámetro `header=FALSE`

Hide

Hide

Hide

```
usedcars <- read.csv("usedcars.csv", stringsAsFactors=FALSE, header=FALSE)
```

Hide

Hide

```
rm(usedcars)
```

- La función `read.csv` es un caso especial de la función `read.table()`.
- Esta nos sirve para leer archivos en muchos otros formatos.
- Por ejemplo, en `.dta`.
- Mas adelante trabajaremos más sobre esto.
- A su vez, es posible escribir un data frame en CSV, por medio de la función `write.csv()`.
- Por ejemplo, guardemos en CSV nuestro data frame `jugadores_seleccion`:

Hide

Hide

```
write.csv(jugadores_seleccion, file="jugadores_seleccion.csv", row.names=FALSE)
```

- Y en efecto podemos verificar que el archivo ha sido creado en la carpeta.

4.3 Explorando y analizando los datos

- Tras recopilar y cargar los datos en R, el siguiente paso es analizarlos y entenderlos.
- Este paso es clave: cuánto más entendamos nuestros datos, mejor sabremos elegir el algoritmo a utilizar.
- Usaremos nuevamente la base de datos de carros usados

Hide

Hide

```
usedcars <- read.csv("usedcars.csv", stringsAsFactors=FALSE)
```

- Exploremos y entendamos estos datos.
- Primero, entendamos la estructura de los datos. Esto con la función `str()` :

Hide

Hide

```
str(usedcars)
```

```
'data.frame':   150 obs. of  6 variables:
 $ year        : int  2011 2011 2011 2011 2012 2010 2011 2010 2011 2010 ...
 $ model       : chr  "SEL" "SEL" "SEL" "SEL" ...
 $ price       : int  21992 20995 19995 17809 17500 17495 17000 16995 16995 16995 ...
 $ mileage     : int  7413 10926 7351 11613 8367 25125 27393 21026 32655 36116 ...
 $ color       : chr  "Yellow" "Gray" "Silver" "Gray" ...
 $ transmission: chr  "AUTO" "AUTO" "AUTO" "AUTO" ...
```

- Vemos que tenemos 150 observaciones y 6 variables.
- `year` , es una variable de números enteros.
- `model` es una variable de texto.
- `price` es de números enteros.
- `mileage` también de enteros.
- `color` de texto.
- `transmisson` también de texto
- Existen diferentes herramientas para describir las variables numéricas.
- La función `summary()` nos da estadística descriptiva.

Hide

Hide

```
summary(usedcars$price)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   3800   11000   13590   12960   14900   21990
```

- También podemos obtener estadística descriptiva de varias variables:

Hide

Hide

```
summary(usedcars[c("year", "mileage")])
```

```
      year         mileage
 Min.   :2000   Min.   :  4867
 1st Qu.:2008   1st Qu.: 27200
 Median :2009   Median : 36385
 Mean   :2009   Mean   : 44261
 3rd Qu.:2010   3rd Qu.: 55124
 Max.   :2012   Max.   :151479
```

- Directamente podemos preguntar por otras medidas descriptivas para las variables. Por ejemplo:

Hide

Hide

```
mean(usedcars$price)
```

```
[1] 12961.93
```

Hide

Hide

```
range(usedcars$price)
```

```
[1]  3800 21992
```

Hide

Hide

```
quantile(usedcars$price)
```

```
     0%     25%     50%     75%    100% 
 3800.0 10995.0 13591.5 14904.5 21992.0 
```

Hide

Hide

```
IQR(usedcars$price)
```

```
[1] 3909.5
```

Hide

Hide

```
var(usedcars$price)
```

```
[1] 9749892
```

Hide

Hide

```
sd(usedcars$price)
```

```
[1] 3122.482
```

- Y así sucesivamente.