

Datos en formato CSV

- Lectura de archivos CSV
- Exportación de datos a CSV

Importar datos desde Excel

- XLConnect
- xlsx

Importar datos en formato ARFF

- foreign
- RWeka

Importar datos de otras fuentes

- Compartir datos mediante el portapapeles
- Obtener datos a partir de una URL
- Datasets integrados

4. Carga de datos

En los capítulos previos hemos usado datos introducidos manualmente en los guiones R o, a lo sumo, generados aleatoriamente mediante funciones como `rnorm()` y `runif()`. En la práctica, la información a la que habremos de aplicar los algoritmos casi siempre se encontrará almacenada en algún tipo de archivo, siendo necesario importarla desde R. También es posible que esa información esté alojada en la web, en el portapapeles o algún otro tipo de recurso.

Este capítulo enumera las funciones que necesitaremos utilizar para importar datos desde fuentes externas, explicando los procedimientos a seguir en cada caso.

4.1 Datos en formato CSV

El formato CSV (*Comma Separated Values*) es uno de los más comunes a la hora de intercambiar datos entre aplicaciones. Un archivo en este formato es un archivo de texto, en el que cada fila es una muestra u ocurrencia y cada columna una variable. Los datos en cada fila se separan entre sí mediante comas, de ahí la denominación del formato.

Un archivo CSV puede incluir o no un encabezado, una primera línea especificando el nombre de cada una de las columnas de datos. El archivo mostrado en la Figura 4.1 cuenta con dicha cabecera. En caso de que el separador para la parte decimal de los números sea la coma, en lugar del punto, los datos de cada fila se separan entre sí mediante puntos y comas. Los datos no numéricos, especialmente cadenas de caracteres, pueden ir delimitados por comillas o no.

En suma, existe una cierta variabilidad en el formato CSV si bien su estructura fundamental es siempre la misma. Por ello R cuenta con varias funciones distintas para operar con este tipo de archivos.

4.1.1 Lectura de archivos CSV

En la mayoría de las ocasiones necesitaremos importar datos almacenados en formato CSV, obteniendo como resultado un *data frame* R. Con este fin podemos recurrir a funciones como `read.table()`, `read.csv()` y `read.csv2()`, siendo estas dos últimas versiones especializadas de la primera, en las que se asigna un valor por defecto concreto a algunos de los parámetros.

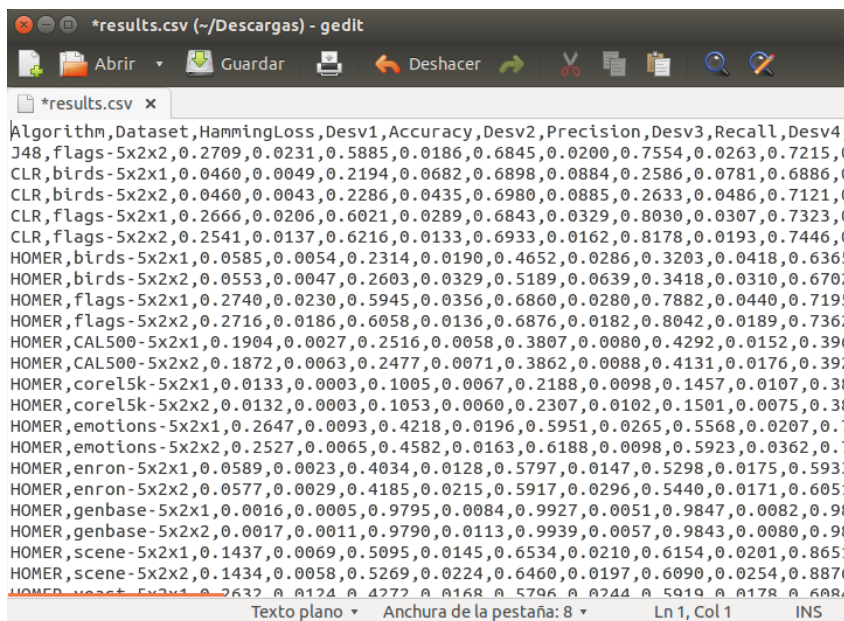


Figura 4.1: VISTA PARCIAL DEL CONTENIDO DE UNA ARCHIVO CSV

Sintaxis 4.1 `read.table(file = archivo[, header = TRUE|FALSE, sep = separadorDatos, dec = separadorDecimal, quote = delimitadorCadenas, stringsAsFactors = TRUE|FALSE])`

Función genérica para la lectura de datos en formato CSV. El único argumento obligatorio es `file`, con el que se facilita el nombre del archivo a leer. Este debe incluir la extensión y, si fuese preciso, también la ruta. El resto de parámetros tienen la siguiente finalidad:

- **header**: De tipo `logical`. Indica a la función si el archivo contiene una cabecera o no.
- **sep**: Determina cuál es el separador de datos en cada fila.
- **dec**: Para datos numéricos, establece cuál es el separador entre parte entera y decimal.
- **quote**: Para datos alfanuméricos, establece el carácter que se usa como delimitador.
- **stringsAsFactors**: Tiene la misma finalidad que en la función `data.frame()` que conocimos en el capítulo previo.

Sintaxis 4.2 `read.csv(archivo)`


Es una implementación especializada de `read.table()` en la que se asume que los parámetros `header`, `sep` y `dec` toman los valores `TRUE`, `" , "` y `" . "`, respectivamente. Acepta los mismos parámetros que `read.table()`.

Sintaxis 4.3 `read.csv2(archivo)`

Es una implementación especializada de `read.table()` en la que se asume que los parámetros `header`, `sep` y `dec` toman los valores `TRUE`, `" ; "` y `" , "`, respectivamente. Acepta los mismos parámetros que `read.table()`.

Sintaxis 4.4 `read.delim(archivo)`

Es una implementación especializada de `read.table()` en la que se asume que los parámetros `header`, `sep` y `dec` toman los valores `TRUE`, `"\t"` y `"."`, respectivamente. Acepta los mismos parámetros que `read.table()`.

 Los archivos de datos utilizados en los ejercicios de este capítulo han de ser descargados previamente desde la página web asociada al curso o bien desde <https://github.com/fcharte/CursoUNIA14>. Se asume que en la ruta actual, que podemos modificar con la función `setwd()` como ya sabemos, se habrá creado un subdirectorio `data` y que en él estarán almacenados los archivos de datos.

En el siguiente ejercicio se carga el contenido de un archivo CSV, con datos sobre el rendimiento de diversos algoritmos, y se muestra información sobre su estructura y parte de los datos leídos:

Ejercicio 4.1 Lectura del contenido de un archivo CSV

```
> results <- read.csv('data/results.csv')
> class(results)

[1] "data.frame"

> str(results)

'data.frame':      188 obs. of  34 variables:
 $ Algorithm : Factor w/ 6 levels "BR-J48","CLR",...: 1 1 1 1 1
   ...
 $ Dataset   : Factor w/ 32 levels "bibtex-5x2x1",...: 5 6 9 10 11
   ...
 $ HammingLoss : num 0.163 0.163 ...
 $ Desv1       : num 0.0015 0.002 0.0001 0.0001 0.0111 ...
 $ Accuracy    : num 0.213 0.214 ...
 $ Desv2       : num 0.0095 0.0063 0.0025 0.0015 0.0246 ...
 $ Precision   : num 0.439 0.44 ...
 $ Desv3       : num 0.0151 0.006 0.014 0.0155 0.0269 ...
 $ Recall      : num 0.297 0.296 ...
 $ Desv4       : num 0.0171 0.013 0.0027 0.0017 0.0354 ...
 $ FMeasure    : num 0.346 0.346 ...
 $ Desv5       : num 0.0126 0.009 0.0109 0.0047 0.007 ...
 $ SubsetAccuracy : num 0 0 ...
 $ Desv6       : num 0 0 0.0017 0.001 0.0278 ...
 $ MacroFMeasure : num 0.295 0.292 ...
 $ Desv7       : num 0.007 0.0093 0.0105 0.0144 0.02 ...
 $ MacroPrecision : num 0.251 0.249 ...
 $ Desv8       : num 0.0161 0.0213 0.0468 0.0403 0.0147 ...
 $ MacroRecall  : num 0.119 0.12 ...
 $ Desv9       : num 0.0069 0.0078 0.0013 0.0011 0.0248 ...
 $ MicroFMeasure : num 0.348 0.349 ...
 $ Desv10      : num 0.0122 0.0091 0.0038 0.0022 0.023 ...
 $ MicroPrecision : num 0.433 0.435 ...
```

```

$ Desv11 : num 0.0168 0.0078 0.0183 0.0197 0.0184 ...
$ MicroRecall : num 0.292 0.292 ...
$ Desv12 : num 0.0154 0.0132 0.0026 0.0016 0.0298 ...
$ OneError : num 0.709 0.731 ...
$ Desv13 : num 0.0339 0.0257 0.0096 0.013 0.0642 ...
$ Coverage : num 169 169 ...
$ Desv14 : num 1.13 0.765 ...
$ RankingLoss : num 0.314 0.318 ...
$ Desv15 : num 0.0134 0.0105 0.0017 0.0034 0.0374 ...
$ AveragePrecision: num 0.347 0.344 ...
$ Desv16 : num 0.013 0.0068 0.0067 0.0059 0.0341 ...

> head(results[,c('Algorithm', 'Dataset', 'Accuracy')])

```

	Algorithm	Dataset	Accuracy
1	BR-J48	CAL500-5x2x1	0.2134
2	BR-J48	CAL500-5x2x2	0.2136
3	BR-J48	corel5k-5x2x1	0.0569
4	BR-J48	corel5k-5x2x2	0.0604
5	BR-J48	emotions-5x2x1	0.4343
6	BR-J48	emotions-5x2x2	0.4527

En este caso podemos comprobar que el *data frame* obtenido cuenta con 34 variables (columnas) y 188 observaciones (filas). Una vez leídos los datos, operariamos sobre ellos como lo haríamos con cualquier otro *data frame*, según se explicó en el capítulo previo.

4.1.2 Exportación de datos a CSV

Si necesitamos guardar un *data frame* podemos usar, como con cualquier otro objeto, la función `save()` que conocimos en un capítulo previo. Esta almacena el contenido del objeto en formato binario. Dicho formato es más compacto y rápido de procesar, pero no resulta útil en caso de que se precise leer la información desde otras aplicaciones.

Las funciones `read.table()`, `read.csv()` y `read.csv2()` antes descritas cuentan con las complementarias `write.table()`, `write.csv()` y `write.csv2()`.

Sintaxis 4.5 `write.table(objeto, file = archivo[,
sep = separadorDatos, dec = separadorDecimal,
quote = delimitadorCadenas, col.names = TRUE|FALSE,
append = TRUE|FALSE])`

Función genérica para la escritura de datos en formato CSV. El único argumento obligatorio, aparte del objeto a almacenar, es `file`, con el que se facilita el nombre del archivo en el que se escribirá. El contenido de este se perderá a menos que se dé el valor `TRUE` al parámetro `append`. El parámetro `col.names` determina si se agregará o no una fila como cabecera, con los nombres de las columnas. El resto de parámetros son equivalentes a los de `read.table()`.

Sintaxis 4.6 `write.csv(objeto, file = archivo)`

Es una implementación especializada de `write.table()` en la que se asume que los parámetros `sep` y `dec` toman los valores `","` y `"."`, respectivamente. Acepta los mismos parámetros que `write.table()`.

Sintaxis 4.7 `write.csv2(archivo)`

Es una implementación especializada de `write.table()` en la que se asume que los parámetros `sep` y `dec` toman los valores `";"` y `","`, respectivamente. Acepta los mismos parámetros que `write.table()`.

4.2 Importar datos desde Excel


Microsoft Excel es una de las aplicaciones más utilizadas para analizar datos y elaborar representaciones gráficas. Por ello es bastante habitual encontrarse con la necesidad de importar a R información alojada en una hoja de cálculo Excel. Para esta tarea podemos recurrir a múltiples paquetes disponibles en CRAN. En esta sección se explica cómo usar dos de ellos.

4.2.1 XLConnect

Este paquete cuenta con varias funciones capaces de leer el contenido de un archivo Excel y generar un *data frame* a partir de todo o parte del contenido de una de sus hojas. Una de esas funciones es `readWorksheetFromFile()`:

Sintaxis 4.8 `readWorksheetFromFile(archivo, sheet = hojaALeer[, header = TRUE|FALSE, region = rango])`

Abre el archivo Excel indicado por el primer parámetro, recupera el contenido de la hoja indicada por `sheet` y genera un *data frame* con esa información. Opcionalmente puede indicarse si los datos cuentan con una cabecera o no, mediante el parámetro `header`. Asimismo, puede facilitarse un rango de celdillas a leer mediante el parámetro `region`. Este contendrá una cadena con un rango del tipo `"B12:D18"`.

 Las funciones del paquete `XLConnect` permiten operar con archivos Excel en el formato heredado `.xls` y también con el nuevo formato `.xlsx`.

En el siguiente ejercicio se utiliza esta función para obtener información sobre subastas en eBay almacenadas en una hoja de cálculo Excel. A continuación se usan las funciones `str()` y `tail()` para tener una idea sobre la estructura de los datos, como hemos hecho en ejemplos previos.

Ejercicio 4.2 Lectura del contenido de una hoja Excel

```
> if(!is.installed('XLConnect'))
+   install.packages('XLConnect')
> library('XLConnect')
> ebay <- readWorksheetFromFile('data/eBayAuctions.xls', sheet=1)
> class(ebay)

[1] "data.frame"
```

```
> str(ebay)

'data.frame':      1972 obs. of  8 variables:
 $ Category : chr "Music/Movie/Game" "Music/Movie/Game" ...
 $ currency : chr "US" "US" ...
 $ sellerRating: num 3249 3249 ...
 $ Duration : num 5 5 5 5 5 ...
 $ endDay : chr "Mon" "Mon" ...
 $ ClosePrice : num 0.01 0.01 0.01 0.01 0.01 ...
 $ OpenPrice : num 0.01 0.01 0.01 0.01 0.01 ...
 $ Competitive.: num 0 0 0 0 0 ...

> tail(ebay)

      Category currency sellerRating Duration endDay ClosePrice
1967 Automotive      US          142         7    Sat      521.55
1968 Automotive      US         2992         5    Sun      359.95
1969 Automotive      US          21         5    Sat      610.00
1970 Automotive      US         1400         5    Mon      549.00
1971 Automotive      US          57         7    Fri      820.00
1972 Automotive      US         145         7    Sat      999.00
      OpenPrice Competitive.
1967      200.00           1
1968      359.95           0
1969      300.00           1
1970      549.00           0
1971      650.00           1
1972      999.00           0
```

Como puede apreciarse, tenemos datos sobre 1972 transacciones y por cada una de ellas tenemos 8 variables distintas: la categoría asociada al objeto puesto en venta, la moneda de pago, la calificación del vendedor, el tiempo que el objeto ha estado en venta, el día de finalización de la subasta, el precio de salida y precio final y un indicador de competitividad.

4.2.2 xlsx

Este paquete también es capaz de trabajar con archivos Excel en los dos formatos habituales, `.xls` y `.xlsx`, ofreciendo funciones que permiten tanto escribir como leer datos de sus hojas. Las dos funciones fundamentales son `write.xlsx()` y `read.xlsx()`. También están disponibles las funciones `write.xlsx2()` y `read.xlsx2()`, funcionalmente equivalentes a las anteriores pero que ofrecen un mejor rendimiento cuando se trabaja con hojas de cálculo muy grandes.

Sintaxis 4.9 `write.xlsx(objeto, archivo[, sheetName = nombreHoja, append = TRUE|FALSE, col.names = TRUE|FALSE, row.names = TRUE|FALSE])`

Crea un archivo Excel con el nombre indicado por el parámetro `archivo`, a menos que se dé el valor `TRUE` al parámetro `append` en cuyo caso se abre un archivo existente para añadir datos. El objeto a escribir ha de ser un *data frame*.

Opcionalmente puede establecerse un nombre para la hoja en la que se introducirá la información, mediante el parámetro `sheetName`. La adición de una línea de cabecera y de los nombres de cada fila están regidos por los parámetros `col.names` y `row.names`, respectivamente.

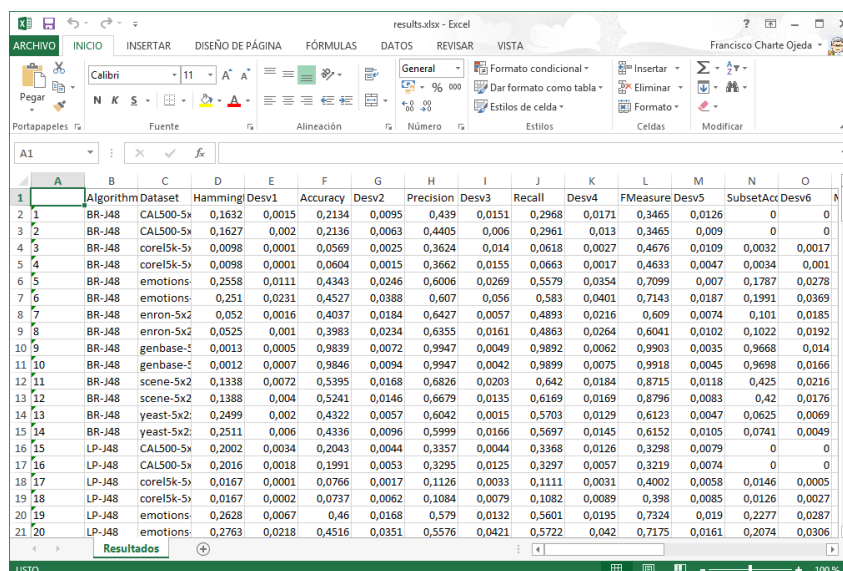
Sintaxis 4.10 `read.xlsx(archivo, sheetIndex = hojaALeer[, sheetName = hojaALeer, header = TRUE|FALSE])`

Abre el archivo Excel indicado por el primer parámetro, recupera el contenido de la hoja indicada por `sheetIndex` (índice numérico de la hoja en el libro Excel) o `sheetName` (nombre de la hoja) y genera un *data frame* con esa información. Opcionalmente puede indicarse si los datos cuentan con una cabecera o no, mediante el parámetro `header`.

En el ejercicio siguiente se utiliza la función `write.xlsx()` para guardar en formato Excel los datos que habíamos obtenido antes de un archivo CSV. La Figura 4.2 muestra el archivo generado por R abierto en Excel.

Ejercicio 4.3 Creación de un archivo Excel con datos desde R

```
> if(!is.installed('xlsx'))
+   install.packages('xlsx')
> library('xlsx')
> write.xlsx(results, file = 'data/results.xlsx',
+           sheetName = 'Resultados')
```



	Algorithm	Dataset	Hamming	Desv1	Accuracy	Desv2	Precision	Desv3	Recall	Desv4	FMeasure	Desv5	SubsetAcc	Desv6	
1	BR-J48	CAL500-5x	0.1632	0.0015	0.2134	0.0095	0.439	0.0151	0.2968	0.0171	0.3465	0.0126	0	0	
2	BR-J48	CAL500-5x	0.1627	0.002	0.2136	0.0063	0.4405	0.006	0.2961	0.013	0.3465	0.009	0	0	
3	BR-J48	corel5k-5x	0.0098	0.0001	0.0569	0.0025	0.3624	0.014	0.0618	0.0027	0.4676	0.0109	0.0032	0.0017	
4	BR-J48	corel5k-5x	0.0098	0.0001	0.0604	0.0015	0.3662	0.0155	0.0663	0.0017	0.4633	0.0047	0.0034	0.001	
5	BR-J48	emotions	0.2558	0.0111	0.4343	0.0246	0.6006	0.0269	0.5579	0.0354	0.7099	0.007	0.1787	0.0278	
6	BR-J48	emotions	0.251	0.0231	0.4527	0.0388	0.607	0.056	0.583	0.0401	0.7143	0.0187	0.1991	0.0369	
7	BR-J48	enron-5x2	0.052	0.0016	0.4037	0.0184	0.6427	0.0057	0.4893	0.0216	0.609	0.0074	0.101	0.0185	
8	BR-J48	enron-5x2	0.0525	0.001	0.3983	0.0234	0.6355	0.0161	0.4863	0.0264	0.6041	0.0102	0.1022	0.0192	
9	BR-J48	genbase-5	0.0013	0.0005	0.9839	0.0072	0.9947	0.0049	0.9892	0.0062	0.9903	0.0035	0.9668	0.014	
10	BR-J48	genbase-5	0.0012	0.0007	0.9846	0.0094	0.9947	0.0042	0.9899	0.0075	0.9918	0.0045	0.9698	0.0166	
11	BR-J48	scene-5x2	0.1338	0.0072	0.5395	0.0168	0.6826	0.0203	0.642	0.0184	0.8715	0.0118	0.425	0.0216	
12	BR-J48	scene-5x2	0.1388	0.004	0.5241	0.0146	0.6679	0.0135	0.6169	0.0169	0.8796	0.0083	0.42	0.0176	
13	BR-J48	yeast-5x2	0.2499	0.002	0.4322	0.0057	0.6042	0.0015	0.5703	0.0129	0.6123	0.0047	0.0625	0.0069	
14	BR-J48	yeast-5x2	0.2511	0.006	0.4336	0.0096	0.5999	0.0166	0.5697	0.0145	0.6152	0.0105	0.0741	0.0049	
15	LP-J48	CAL500-5x	0.2002	0.0034	0.2043	0.0044	0.3357	0.0044	0.3368	0.0126	0.3298	0.0079	0	0	
16	LP-J48	CAL500-5x	0.2016	0.0018	0.1991	0.0053	0.3295	0.0125	0.3297	0.0057	0.3219	0.0074	0	0	
17	LP-J48	corel5k-5x	0.0167	0.0001	0.0766	0.0017	0.1126	0.0033	0.1111	0.0031	0.4002	0.0058	0.0146	0.0005	
18	LP-J48	corel5k-5x	0.0167	0.0002	0.0737	0.0062	0.1084	0.0079	0.1082	0.0089	0.398	0.0085	0.0126	0.0027	
19	LP-J48	emotions	0.2628	0.0067	0.46	0.0168	0.579	0.0132	0.5601	0.0195	0.7324	0.019	0.2277	0.0287	
20	LP-J48	emotions	0.2763	0.0218	0.4516	0.0351	0.5576	0.0421	0.5722	0.042	0.7175	0.0161	0.2074	0.0306	

Figura 4.2: VISTA PARCIAL DEL ARCHIVO EXCEL CREADO DESDE R

La lectura de una hoja Excel con `read.xlsx()` tiene prácticamente la misma sintaxis que la usada antes con `readWorksheetFromFile()` y, obviamente, el resultado es el mismo, como se aprecia en el siguiente ejercicio:

Ejercicio 4.4 Lectura del contenido de una hoja Excel

```
> ebay <- read.xlsx('data/eBayAuctions.xls', sheetIndex=1)
> class(ebay)

[1] "data.frame"

> str(ebay)

'data.frame':      1972 obs. of  8 variables:
 $ Category : Factor w/ 18 levels "Antique/Art/Craft",...: 14 14
   14 14 14 ...
 $ currency : Factor w/ 3 levels "EUR","GBP","US": 3 3 3 3 3 ...
 $ sellerRating: num 3249 3249 ...
 $ Duration : num 5 5 5 5 5 ...
 $ endDay : Factor w/ 7 levels "Fri","Mon","Sat",...: 2 2 2 2 2
   ...
 $ ClosePrice : num 0.01 0.01 0.01 0.01 0.01 ...
 $ OpenPrice : num 0.01 0.01 0.01 0.01 0.01 ...
 $ Competitive.: num 0 0 0 0 0 ...

> tail(ebay)


      Category currency sellerRating Duration endDay ClosePrice
1967 Automotive      US          142         7    Sat     521.55
1968 Automotive      US        2992         5    Sun     359.95
1969 Automotive      US          21         5    Sat     610.00
1970 Automotive      US       1400         5    Mon     549.00
1971 Automotive      US          57         7    Fri     820.00
1972 Automotive      US         145         7    Sat     999.00
      OpenPrice Competitive.
1967      200.00           1
1968      359.95           0
1969      300.00           1
1970      549.00           0
1971      650.00           1
1972      999.00           0
```



En <http://fcharte.com/Default.asp?busqueda=1&q=Excel+y+R> podemos encontrar un trío de artículos dedicados al intercambio de datos entre R y Excel que pueden sernos útiles si trabajamos habitualmente con estas dos herramientas.

4.3 Importar datos en formato ARFF

El formato ARFF (*Attribute-Relation File Format*) fue creado para el software de minería de datos WEKA¹, siendo actualmente utilizado por muchos otros paquetes de software. Es muy probable que al trabajar con R necesitemos obtener información alojada en archivos `.arff`, que es la extensión habitual para datos con dicho formato.

 Un archivo `.arff` es básicamente un archivo en formato CSV con una cabecera compuesta de múltiples líneas, definiendo cada una de ellas el nombre y tipo de los atributos (las columnas) de cada fila. Las secciones de cabecera y de datos están señaladas mediante etiquetas.

Al igual que ocurría con las hojas Excel, existen varios paquetes que nos ofrecen funciones capaces de leer el contenido de archivos en formato ARFF. En los siguientes apartados se describen dos de esos paquetes.

4.3.1 foreign

Este paquete ofrece múltiples funciones del tipo `read.XXX()`, representando `XXX` un formato de archivo de datos como puede ser `spss`, `dbf`, `octave` y `arff`. En total hay una decena de funciones de lectura, entre ellas `read.arff()`, así como algunas funciones de escritura, incluyendo `write.arff()`.

Sintaxis 4.11 `read.arff(archivo)`

Lee el contenido del archivo ARFF indicado y genera un *data frame* que devuelve como resultado.

Sintaxis 4.12 `write.arff(objeto, file = archivo)`

Escribe el objeto entregado como parámetro, normalmente será un *data frame*, en el archivo ARFF indicado por el parámetro `file`.

4.3.2 RWeka

Este paquete actúa como una interfaz completa entre R y la funcionalidad ofrecida por el software WEKA, incluyendo el acceso a los algoritmos de obtención de reglas, agrupamiento, clasificación, etc. No es necesario tener instalado WEKA, al instalar el paquete `RWeka` también se instalarán las bibliotecas Java que forman el núcleo de dicho software.

Al igual que `foreign`, el paquete `RWeka` también aporta las funciones `read.arff()` y `write.arff()`. El siguiente ejercicio muestra cómo utilizar la primera de ellas para leer el dataset `Coverttype`².

Ejercicio 4.5 Carga de un dataset en formato ARFF con el paquete `RWeka`

```
> if(!is.installed('RWeka'))
+   install.packages('RWeka')
> library('RWeka')
> coverttype <- read.arff('data/coverttype.arff')
> class(coverttype)
```

¹<http://www.cs.waikato.ac.nz/ml/weka/>

²Este dataset contiene una docena de variables cartográficas a partir de las cuales se trata de predecir el tipo cubierta forestal del terreno. Más información sobre este dataset en <https://archive.ics.uci.edu/ml/datasets/Coverttype>.

```
[1] "data.frame"

> str(covertime)

'data.frame':      581012 obs. of  13 variables:
 $ elevation : num 2596 2590 ...
 $ aspect    : num 51 56 139 155 45 ...
 $ slope     : num 3 2 9 18 2 ...
 $ horz_dist_hydro: num 258 212 268 242 153 ...
 $ vert_dist_hydro: num 0 -6 65 118 -1 ...
 $ horiz_dist_road: num 510 390 3180 3090 391 ...
 $ hillshade_9am : num 221 220 234 238 220 ...
 $ hillshade_noon : num 232 235 238 238 234 ...
 $ hillshade_3pm : num 148 151 135 122 150 ...
 $ horiz_dist_fire: num 6279 6225 ...
 $ wilderness_area: Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1
 ...
 $ soil_type : Factor w/ 40 levels "1","2","3","4",...: 29 29 12
 30 29 ...
 $ class : Factor w/ 7 levels "1","2","3","4",...: 5 5 2 2 5 ...

> head(covertime)

  elevation aspect slope horz_dist_hydro vert_dist_hydro
1     2596     51     3           258             0
2     2590     56     2           212            -6
3     2804    139     9           268             65
4     2785    155    18           242            118
5     2595     45     2           153             -1
6     2579    132     6           300            -15
  horiz_dist_road hillshade_9am hillshade_noon hillshade_3pm
1             510           221           232           148
2             390           220           235           151
3            3180           234           238           135
4            3090           238           238           122
5             391           220           234           150
6              67           230           237           140
  horiz_dist_fire wilderness_area soil_type class
1            6279             1         29     5
2            6225             1         29     5
3            6121             1         12     2
4            6211             1         30     2
5            6172             1         29     5
6            6031             1         29     2
```

En la información facilitada por la función `str()` puede apreciarse que el dataset cuenta con 581 012 observaciones, por lo que su carga puede precisar un cierto tiempo dependiendo de la potencia del equipo donde se esté trabajando.

4.4 Importar datos de otras fuentes

Aunque en la mayoría de los casos las funciones `read.csv()`, `read.xlsx()` y `read.arff()` serán suficientes para importar los datos con los que tengamos que trabajar, habrá ocasiones en que la información no se encuentre en un archivo que podamos leer. En esta sección se describen tres de esos casos.

4.4.1 Compartir datos mediante el portapapeles

Si los datos que necesitamos están en una aplicación que no nos permite exportar a CSV o algún otro formato más o menos estándar, por regla general siempre podremos recurrir al uso del portapapeles. En R el portapapeles se trata como si fuese un archivo cualquiera, lo único característico es su nombre: `clipboard`. También podemos usar este recurso en sentido inverso, copiando datos desde R al portapapeles para que otra aplicación pueda recuperarlos.

Funciones que ya conocemos, como `read.delim()` y `write.table()`, pueden utilizarse para importar y exportar datos al portapapeles. Si tenemos una hoja de cálculo Excel abierta, tras seleccionar un rango de celdillas y copiarlo al portapapeles la función `read.delim()` sería la adecuada para obtener su contenido desde R, ya que por defecto Excel usa tabuladores para separar los datos. De igual manera, podríamos copiar en el portapapeles un *data frame* R mediante la función `write.table()`, especificando que el separador debe ser el tabulador.

El siguiente ejemplo muestra cómo usar el portapapeles, en este caso exportando a él parte de un *data frame* que después es importado desde el propio R en otra variable:

Ejercicio 4.6 Copiar información a y desde el portapapeles

```
> write.table(results[1:100,], 'clipboard', sep='\t')  
> partial.results <- read.delim('clipboard')
```

4.4.2 Obtener datos a partir de una URL

Si los datos con los necesitamos trabajar están alojados en un sitio web, como puede ser un repositorio de **GitHub**, no es necesario que descargemos el archivo a una ubicación local para a continuación leerlo. Gracias a la función `getURL()` del paquete **RCurl** podemos leer directamente desde la URL. Esto nos permitirá trabajar siempre con la última versión disponible de los datos, sin necesidad de comprobar manualmente si ha habido cambios desde la última vez que los descargamos desde el repositorio.

Sintaxis 4.13 `getURL(URL[, write = funcionLectura])`

Descarga la información especificada por el parámetro URL. Opcionalmente puede facilitarse una función a la que se irá invocando repetidamente a medida que lleguen bloques de datos con la respuesta. Se aceptan muchos otros parámetros cuya finalidad es controlar la solicitud HTTP y la gestión de la respuesta.

El valor devuelto por la función `getURL()` representa la información descargada. Podemos usarlo como parámetro para crear un objeto `textConnection`, del que podemos leer como si de un archivo cualquiera se tratase. Esto es lo que se hace

en el siguiente ejercicio a fin de obtener el contenido de un archivo CSV desde un repositorio GitHub:

Ejercicio 4.7 Lectura de un archivo alojado en un repositorio GitHub

```
> if(!is.installed('RCurl'))
+   install.packages('RCurl')
> library('RCurl')

> url <- getURL(
+   'https://raw.githubusercontent.com/fcharte/CursoUNIA14/
+   master/data/results.csv', ssl.verifypeer = FALSE)

> results2 <- read.csv(textConnection(url))

> str(results2)

'data.frame':      188 obs. of  34 variables:
 $ Algorithm      : Factor w/ 6 levels "BR-J48","CLR",...: 1 1 1 1 1 ...
 $ Dataset        : Factor w/ 32 levels "bibtex-5x2x1",...: 5 6 9 10 11 ...
 $ HammingLoss    : num  0.163 0.163 ...
 $ Desv1          : num  0.0015 0.002 0.0001 0.0001 0.0111 ...
 $ Accuracy       : num  0.213 0.214 ...
 $ Desv2          : num  0.0095 0.0063 0.0025 0.0015 0.0246 ...
 $ Precision      : num  0.439 0.44 ...
 $ Desv3          : num  0.0151 0.006 0.014 0.0155 0.0269 ...
 $ Recall         : num  0.297 0.296 ...
 $ Desv4          : num  0.0171 0.013 0.0027 0.0017 0.0354 ...
 $ FMeasure       : num  0.346 0.346 ...
 $ Desv5          : num  0.0126 0.009 0.0109 0.0047 0.007 ...
 $ SubsetAccuracy : num  0 0 ...
 $ Desv6          : num  0 0 0.0017 0.001 0.0278 ...
 $ MacroFMeasure  : num  0.295 0.292 ...
 $ Desv7          : num  0.007 0.0093 0.0105 0.0144 0.02 ...
 $ MacroPrecision : num  0.251 0.249 ...
 $ Desv8          : num  0.0161 0.0213 0.0468 0.0403 0.0147 ...
 $ MacroRecall    : num  0.119 0.12 ...
 $ Desv9          : num  0.0069 0.0078 0.0013 0.0011 0.0248 ...
 $ MicroFMeasure  : num  0.348 0.349 ...
 $ Desv10         : num  0.0122 0.0091 0.0038 0.0022 0.023 ...
 $ MicroPrecision : num  0.433 0.435 ...
 $ Desv11         : num  0.0168 0.0078 0.0183 0.0197 0.0184 ...
 $ MicroRecall    : num  0.292 0.292 ...
 $ Desv12         : num  0.0154 0.0132 0.0026 0.0016 0.0298 ...
 $ OneError       : num  0.709 0.731 ...
 $ Desv13         : num  0.0339 0.0257 0.0096 0.013 0.0642 ...
 $ Coverage       : num  169 169 ...
 $ Desv14         : num  1.13 0.765 ...
 $ RankingLoss    : num  0.314 0.318 ...
 $ Desv15         : num  0.0134 0.0105 0.0017 0.0034 0.0374 ...
 $ AveragePrecision: num  0.347 0.344 ...
 $ Desv16         : num  0.013 0.0068 0.0067 0.0059 0.0341 ...
```

4.4.3 Datasets integrados

Muchos paquetes R incorporan datasets propios preparados para su uso. La instalación base de R aporta un paquete, llamado **datasets**, con docenas de bases de datos. Podemos usar la función **data()** para obtener una lista completa de todos los datasets disponibles, así como para cargar cualquiera de ellos.

Sintaxis 4.14 `data([dataset, package = nombrePaquete])`

Ejecutada sin parámetros, esta función abre un documento enumerando todos los datasets disponibles en los paquetes actualmente cargados en la sesión de trabajo. Si se facilita uno o más nombres de datasets, estos son cargados en memoria y quedan preparados para su uso. Opcionalmente puede especificarse el paquete en que están alojados los datasets.

Una vez se ha cargado un dataset, podemos usarlo como haríamos con cualquier *data frame*. En el siguiente ejercicio se utiliza el conocido dataset **iris**:

Ejercicio 4.8 Lectura de un archivo alojado en un repositorio GitHub

```
> library(datasets)
> data(iris)

> str(iris)

'data.frame':      150 obs. of  5 variables:
 $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...
 1 1 1 1 1 1 ...

> head(iris)

  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
```